**Scrapy一目了然**

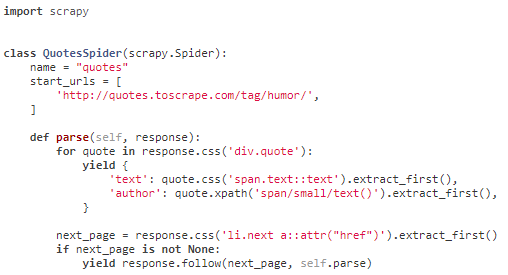
Scrapy是一个应用程序框架，用于抓取网站并提取可用于广泛的有用应用程序的结构化数据，如数据挖掘，信息处理或历史档案。

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。

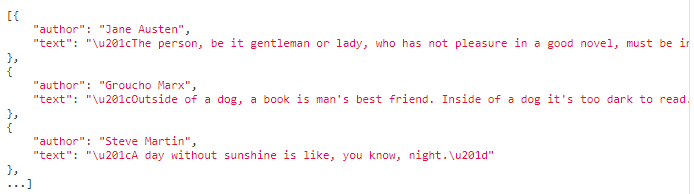
下面是一个蜘蛛的代码，它可以在分页之后从网站[http://quotes.toscrape.com](http://quotes.toscrape.com/)摘录出名言 ：



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)。

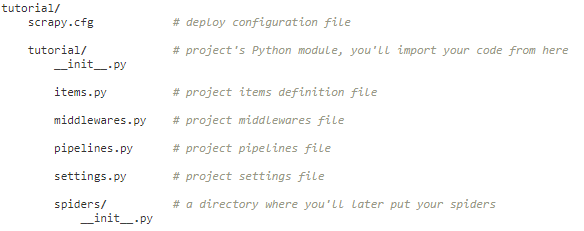
如果您是编程新手，想从Python开始，那么您可能会对联机丛书“ [Learn Python The Hard Way”感兴趣](https://learnpythonthehardway.org/book/)。您还可以查看[非程序员的Python资源列表](https://wiki.python.org/moin/BeginnersGuide/NonProgrammers)。

## 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。

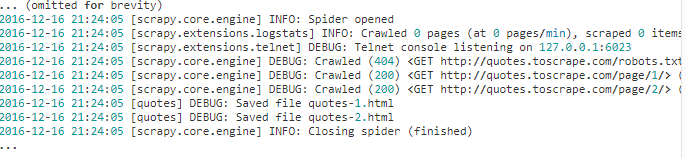
该[**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse)方法通常解析响应，将提取的数据提取为字符串，并查找新的URL并[**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)根据它们创建新的请求（）。

### 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**

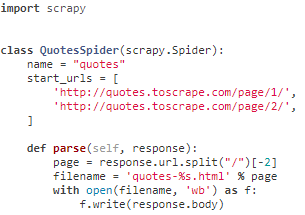
如果您想知道为什么我们还没有解析HTML，请坚持下去，我们会很快回覆。

#### 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：

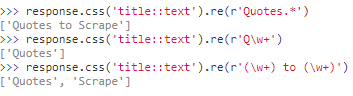
>>> response**.**css('title::text')[0]**.**extract()

'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：

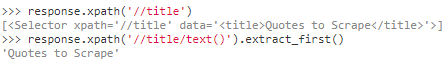


为了找到合适的CSS选择器来使用，你可能会发现在你的web浏览器中使用shell打开响应页面很有用view(response)。您可以使用浏览器开发工具或Firebug等扩展（请参阅关于[使用Firebug进行抓取](https://doc.scrapy.org/en/latest/topics/firebug.html#topics-firebug)和[使用Firefox进行抓取的部分](https://doc.scrapy.org/en/latest/topics/firefox.html#topics-firefox)）。

[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。

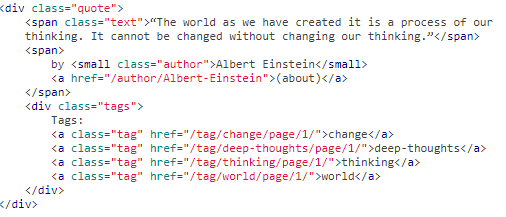
虽然可能不如CSS选择器那么受欢迎，但XPath表达式提供了更多的功能，因为除了浏览结构之外，它还可以查看内容。使用XPath，您可以选择如下内容：选择包含文本“下一页”的链接。这使得XPath非常适合抓取任务，并且即使您已经知道如何构建CSS选择器，我们也鼓励您学习XPath，这会使抓取更容易。

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.”'}

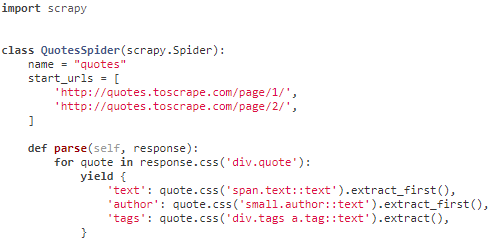
... a few more of these, omitted for brevity

>>>

### 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()

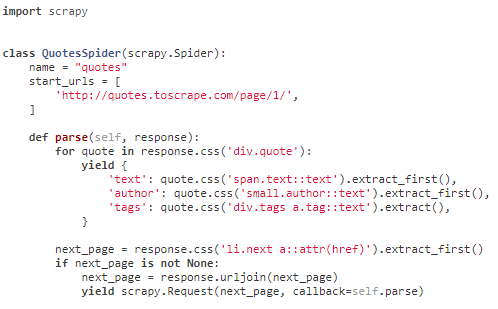
'<a href="/page/2/">Next <span aria-hidden="true">→</span></a>'

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。

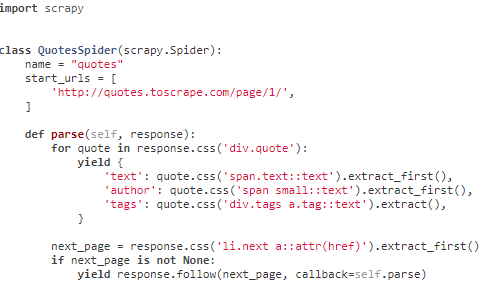
您在这里看到的是Scrapy的以下链接机制：当您在回调方法中产生请求时，Scrapy会安排发送请求并注册一个回调方法，以便在请求结束时执行。

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'):

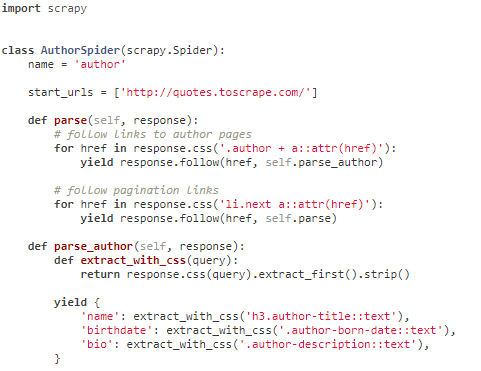
**yield** response**.**follow(a, callback**=**self**.**parse)

**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)。

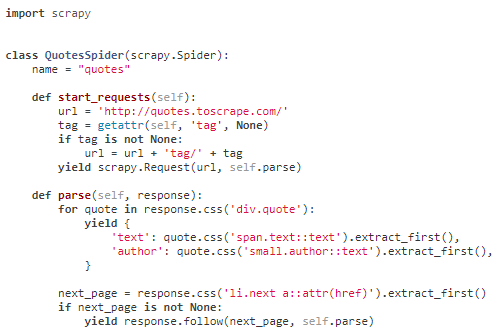
## 使用蜘蛛参数

-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。

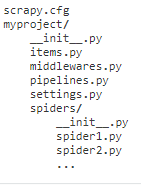
Scrapy也理解并可以通过一些环境变量进行配置。目前这些是：

* 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：



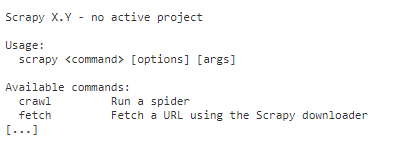
scrapy.cfg文件所在的目录称为项目根目录。该文件包含定义项目设置的python模块的名称。这里是一个例子：

[settings]

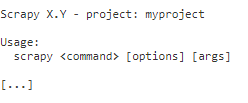
default **=** myproject**.**settings

## 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：

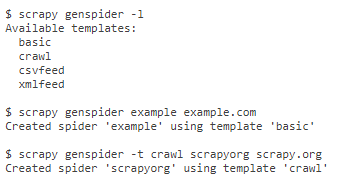
$ scrapy startproject myproject

### 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：

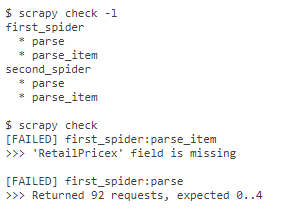


### check

* 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。

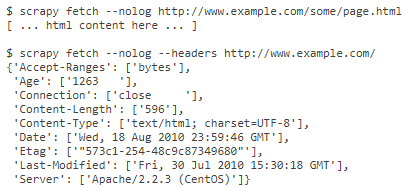
所以这个命令可以用来“看”你的蜘蛛如何获取某个页面。

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

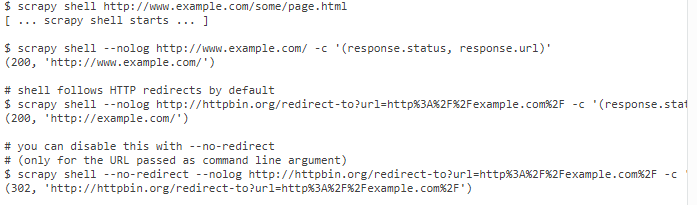
* 句法： scrapy shell [url]
* 需要项目：*没有*

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

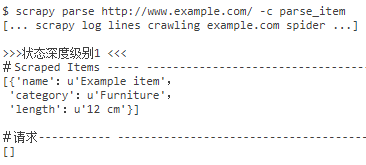
* 句法： scrapy parse <url> [options]
* 需要项目：*是的*

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：



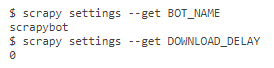
### settings

* 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 。

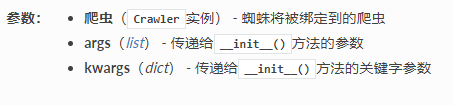
**from\_crawler(crawler, \*args, \*\*kwargs)**

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 |

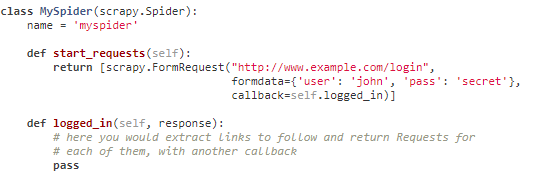


**start\_requests（）**

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 |

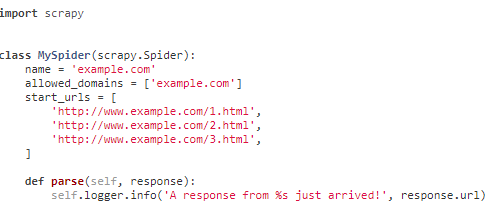
**log(message[, level, component])**

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)。

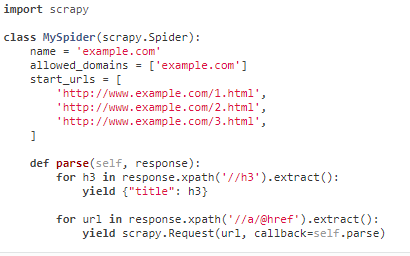
**closed(reason)**

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)：



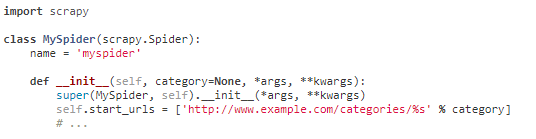
## 蜘蛛的论点

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)：

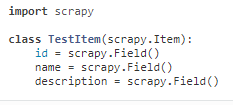


蜘蛛参数也可以通过Scrapyd schedule.jsonAPI 传递。请参阅[Scrapyd文档](https://scrapyd.readthedocs.io/en/latest/)。

## 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。

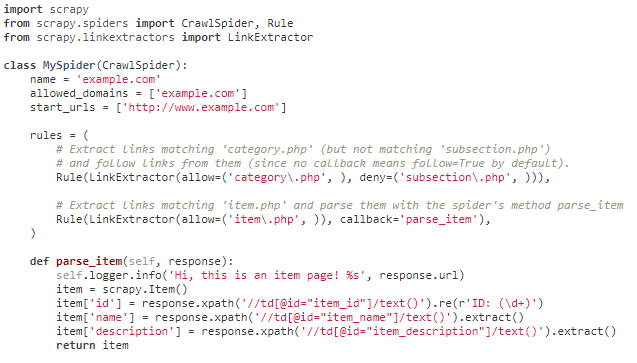
**follow**是一个布尔值，用于指定是否应该遵循通过此规则提取的每个响应之间的链接。如果**callback**是无**follow**默认值**True**，则默认为**False**。

**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)**

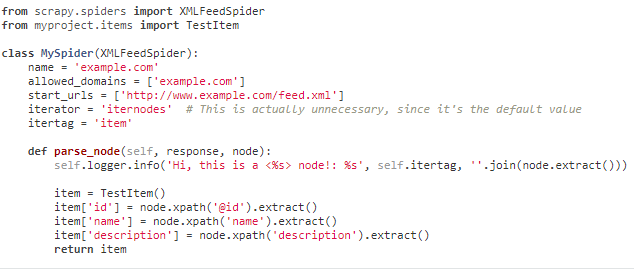
为与提供的标签名称（**itertag**）匹配的节点调用此方法。接收响应并[**Selector**](https://doc.scrapy.org/en/latest/topics/selectors.html#scrapy.selector.Selector)为每个节点分配一个响应 。覆盖此方法是强制性的。否则，你的蜘蛛将无法工作。此方法必须返回一个[**Item**](https://doc.scrapy.org/en/latest/topics/items.html#scrapy.item.Item)对象，一个 [**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)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

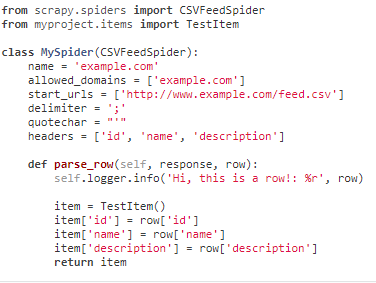
CSV文件中列名称的列表。

**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。

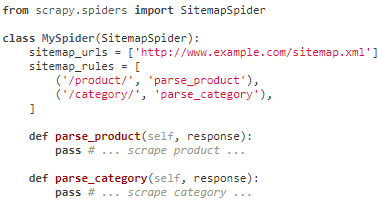
默认是**sitemap\_alternate\_links**禁用的。

#### 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 ：



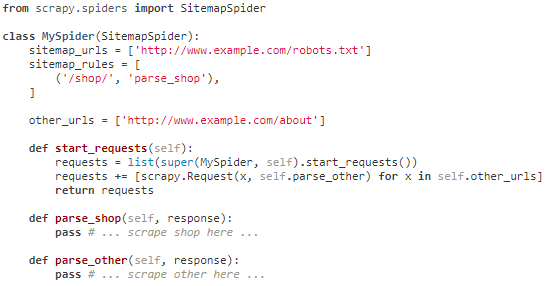
使用不同的回叫处理某些回叫和其他网址的某些网址：



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

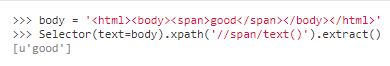
### 构建选择器

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：



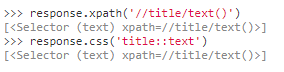
然后，在加载shell之后，您将有可用的响应作为response shell变量，并将其附加的选择器作为response.selector属性。

由于我们正在处理HTML，选择器将自动使用HTML解析器。

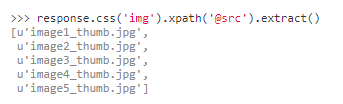
因此，通过查看该页面的[HTML代码](https://doc.scrapy.org/en/latest/topics/selectors.html#topics-selectors-htmlcode)，我们构建一个用于选择标题标签内的文本的XPath：



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：

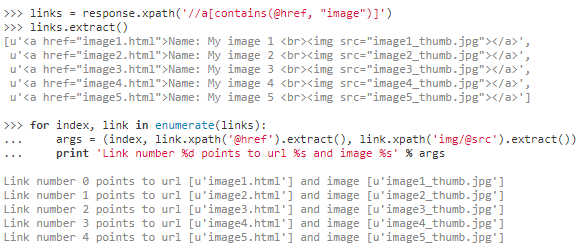


现在我们要获取基本URL和一些图像链接：



### 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

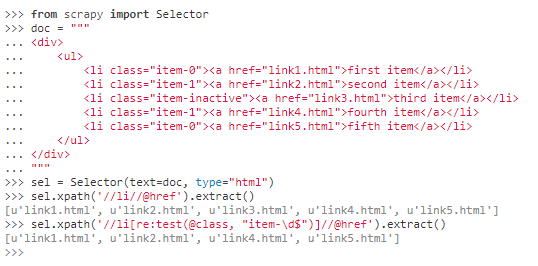
构建在[lxml](http://lxml.de/)之上，Scrapy选择器还支持一些[EXSLT](http://exslt.org/)扩展，并提供这些预先注册的名称空间以用于XPath表达式：

| **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**

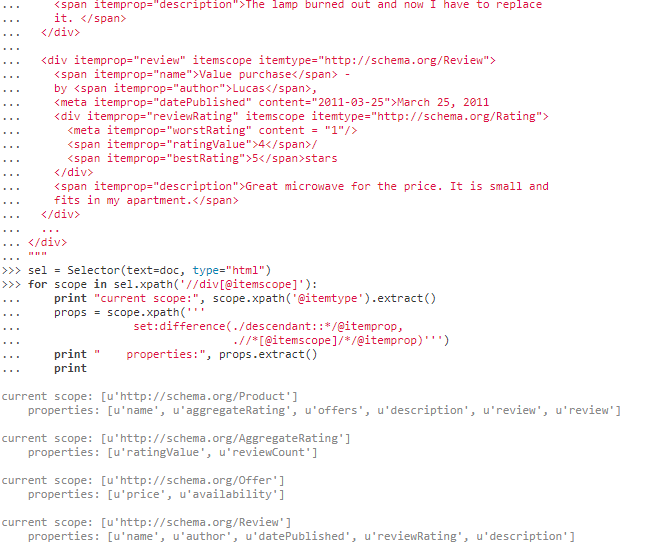
C库libxslt本身不支持EXSLT正则表达式，所以[lxml](http://lxml.de/)的实现在Python re模块中使用钩子。因此，在XPath表达式中使用正则表达式函数可能会增加一点性能损失。

#### 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

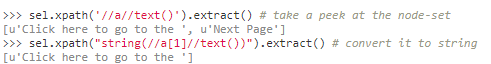
当您需要使用文本内容作为[XPath字符串函数的](https://www.w3.org/TR/xpath/#section-String-Functions)参数时，请避免使用.//text()和使用.。

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。

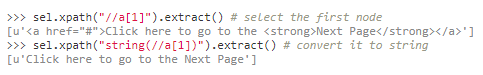
例：



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：

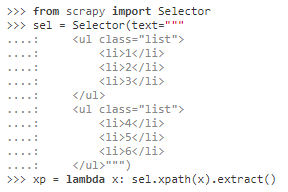


#### 当心//节点[1]和（//节点）[1]之间的区别

//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。

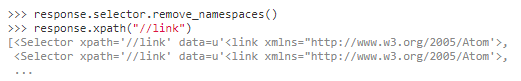
首先，我们用我们想要刮取的url打开shell：

$ 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。