命令行工具

版本0.10中的新功能。

Scrapy通过`scrapy`命令行工具进行控制，在此称为“Scrapy工具”，以区别于子命令，我们称之为“命令”或“Scrapy命令”。

Scrapy工具提供了多个命令，用于多种用途，每个命令都接受一组不同的参数和选项。

（该命令已在1.0中删除，有利于独立。请参阅部署项目。）`scrapy deploy` `scrapyd-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`（全系统），
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（参见下一节）。

这些文件中的设置按列出的优先顺序合并：用户定义的值的优先级高于系统范围的默认值，项目范围的设置将在定义时覆盖所有其他值。

Scrapy也了解并且可以通过许多环境变量进行配置。目前这些是：

- `SCRAPY_SETTINGS_MODULE`（请参阅指定设置）
- `SCRAPY_PROJECT`
- `SCRAPY_PYTHON_SHELL`（见Scrapy 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，让我们先了解一下Scrapy项目的目录结构。

虽然可以修改，但默认情况下所有Scrapy项目都具有相同的文件结构，类似于：

```
scrapy.cfg
myproject/
    __init__.py
    items.py
    middlewares.py
    pipelines.py
    settings.py
    spiders/
        __init__.py
        spider1.py
        spider2.py
        ...
```

`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。这是一个例子：

```
[settings]
default = myproject.settings
```

使用`scrapy`工具

您可以从没有参数的Scrapy工具开始，它将打印一些使用帮助和可用命令：

```
Scrapy X.Y - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  crawl         Run a spider
  fetch         Fetch a URL using the Scrapy downloader
[...]
```

如果您在Scrapy项目中，第一行将打印当前活动的项目。在这个例子中，它是从项目外部运行的。如果从项目内部运行，它将打印出如下内容：

```
Scrapy X.Y - project: myproject

Usage:
  scrapy <command> [options] [args]

[...]
```

创建项目

您通常使用该`scrapy`工具做的第一件事是创建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`将是相同的`myproject`。

接下来，进入新项目目录：

```
cd project_dir
```

您已准备好使用该`scrapy`命令从那里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，要创建一个新蜘蛛：

```
scrapy genspider mydomain mydomain.com
```

某些Scrapy命令（如`crawl`）必须从Scrapy项目内部运行。有关必须从项目内部运行哪些命令以及哪些命令不能运行，请参阅下面的命令参考。

还要记住，某些命令在从项目内部运行时可能会略有不同的行为。例如，`user_agent`如果获取的url与某个特定的spider相关联，则fetch命令将使用spider-overridden行为（例如覆盖用户代理的属性）。这是故意的，因为该`fetch`命令旨在用于检查蜘蛛如何下载页面。

可用的工具命令

本节包含可用内置命令的列表，其中包含说明和一些用法示例。请记住，您始终可以通过运行以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
```

有两种命令，一种只能在Scrapy项目内部工作（特定于项目的命令）和那些在没有活动的Scrapy项目（全局命令）的情况下工作的命令，尽管从项目内部运行时它们可能表现略有不同（因为他们会使用项目覆盖设置）。

全局命令：

- `startproject`
- `genspider`
- `settings`
- `runspider`
- `shell`
- `fetch`
- `view`
- `version`

仅限项目的命令：

- `crawl`
- `check`
- `list`
- `edit`
- `parse`
- `bench`

startproject命令

- 句法： `scrapy startproject <project_name> [project_dir]`
- 需要项目：*没有*

`project_name`在`project_dir`目录下创建一个名为的新Scrapy项目。如果`project_dir`没有指定，`project_dir`将是相同的`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`如果从项目内部调用，则在当前文件夹或当前项目的文件夹中创建新的蜘蛛。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：

```
$ scrapy genspider -l
Available templates:
  basic
  crawl
  csvfeed
  xmlfeed

$ scrapy genspider example example.com
Created spider 'example' using template 'basic'

$ scrapy genspider -t crawl scrapyorg scrapy.org
Created spider 'scrapyorg' using template 'crawl'
```

这只是一个方便的快捷方式命令，用于根据预定义的模板创建蜘蛛，但肯定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。

抓取

- 句法： `scrapy crawl <spider>`
- 需要项目：*是的*

使用蜘蛛开始抓取。

用法示例：

```
$ scrapy crawl myspider
[ ... myspider starts crawling ... ]
```

检查

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

运行合同检查。

用法示例：

```
$ scrapy check -l
first_spider
  * parse
  * parse_item
second_spider
  * parse
  * parse_item

$ scrapy check
[FAILED] first_spider:parse_item
>>> 'RetailPricex' field is missing

[FAILED] first_spider:parse
>>> Returned 92 requests, expected 0..4
```

列表

- 句法： `scrapy list`
- 需要项目：*是的*

列出当前项目中的所有可用蜘蛛。输出是每行一个蜘蛛。

用法示例：

```
$ scrapy list
spider1
spider2
```

编辑

- 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或（如果未设置）`EDITOR`设置编辑给定的蜘蛛。

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：

```
$ scrapy edit spider1
```

获取

- 句法： `scrapy fetch <url>`
- 需要项目：*没有*

使用Scrapy下载程序下载给定的URL，并将内容写入标准输出。

这个命令的有趣之处在于它获取了蜘蛛下载它的页面。例如，如果蜘蛛具有`USER_AGENT`覆盖用户代理的属性，则它将使用该属性。

因此，此命令可用于“查看”您的蜘蛛如何获取某个页面。

如果在项目外部使用，则不会应用特定的每蜘蛛行为，它将仅使用默认的Scrapy下载程序设置。

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标头而不是响应的正文
- `--no-redirect`：不要遵循HTTP 3xx重定向（默认是遵循它们）

用法示例：

```
$ scrapy fetch --nolog http://www.example.com/some/page.html
[ ... html content here ... ]

$ scrapy fetch --nolog --headers http://www.example.com/
{'Accept-Ranges': ['bytes'],
 'Age': ['1263   '],
 'Connection': ['close     '],
 'Content-Length': ['596'],
 'Content-Type': ['text/html; charset=UTF-8'],
 'Date': ['Wed, 18 Aug 2010 23:59:46 GMT'],
 'Etag': ['"573c1-254-48c9c87349680"'],
 'Last-Modified': ['Fri, 30 Jul 2010 15:30:18 GMT'],
 'Server': ['Apache/2.2.3 (CentOS)']}
```

视图

- 句法：`scrapy view <url>`
- 需要项目：*没有*

在浏览器中打开给定的URL，因为您的Scrapy蜘蛛会“看到”它。有时蜘蛛会看到不同于普通用户的页面，因此可以用来检查蜘蛛“看到”的内容并确认它是您所期望的。

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`：不要遵循HTTP 3xx重定向（默认是遵循它们）

用法示例：

```
$ scrapy view http://www.example.com/some/page.html
[ ... browser starts ... ]
```

外壳

- 句法：`scrapy shell [url]`
- 需要项目：*没有*

为给定的URL启动Scrapy shell（如果给定），如果没有给出URL，则为空。还支持UNIX样式的本地文件路径，相对于 `./` 或 `../` 前缀或绝对文件路径。有关详细信息，请参阅Scrapy 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`：不要遵循HTTP 3xx重定向（默认是遵循它们）；这只会影响您在命令行中作为参数传递的URL; 一旦进入shell，`fetch(url)`默认情况下仍会遵循HTTP重定向。

用法示例：

```
$ scrapy shell http://www.example.com/some/page.html
[ ... scrapy shell starts ... ]

$ scrapy shell --nolog http://www.example.com/ -c '(response.status, response.url)'
(200, 'http://www.example.com/')

# shell follows HTTP redirects by default
$ scrapy shell --nolog http://httpbin.org/redirect-to?url=http%3A%2F%2Fexample.com%2F -c
'(response.status, response.url)'
(200, 'http://example.com/')

# you can disable this with --no-redirect
# (only for the URL passed as command line argument)
$ scrapy shell --no-redirect --nolog http://httpbin.org/redirect-to?
url=http%3A%2F%2Fexample.com%2F -c '(response.status, response.url)'
(302, 'http://httpbin.org/redirect-to?url=http%3A%2F%2Fexample.com%2F')
```

解析

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

获取给定的URL并使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（可能重复）
- `--callback`或`-c`：用作解析响应的回调的spider方法
- `--meta`或`-m`：将传递给回调请求的其他请求元。这必须是有效的json字符串。示例：-meta ='{“foo”：“bar”}'
- `--pipelines`：通过管道处理项目
- `--rules`或`-r`：使用`CrawlSpider`规则来发现用于解析响应的回调（即蜘蛛方法）
- `--noitems`：不显示刮下的物品
- `--nolinks`：不显示提取的链接
- `--nocolour`：避免使用pygments为输出着色
- `--depth`或`-d`：递归请求的深度级别（默认值：1）
- `--verbose`或`-v`：显示每个深度级别的信息

用法示例：

```
$ scrapy parse http://www.example.com/ -c parse_item
[ ... scrapy log lines crawling example.com spider ... ]

>>> STATUS DEPTH LEVEL 1 <<<
# Scraped Items  ------------------------------------------------------------
[{'name': u'Example item',
 'category': u'Furniture',
 'length': u'12 cm'}]

# Requests  -----------------------------------------------------------------
[]
```

设置

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

获取Scrapy设置的值。

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：

```
$ scrapy settings --get BOT_NAME
scrapybot
$ scrapy settings --get DOWNLOAD_DELAY
0
```

runspider

- 句法： `scrapy runspider <spider_file.py>`
- 需要项目：*没有*

在Python文件中运行自包含的蜘蛛，而无需创建项目。

用法示例：

```
$ scrapy runspider myspider.py
[ ... spider starts crawling ... ]
```

版本

- 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。

长凳

版本0.17中的新功能。

- 句法：`scrapy bench`
- 需要项目：*没有*

运行快速基准测试。基准测试。

自定义项目命令

您还可以使用该`COMMANDS_MODULE`设置添加自定义项目命令 。有关如何实现命令的示例，请参阅scrapy /命令中的 Scrapy命令。

COMMANDS_MODULE

默认值:(`''`空字符串)

用于查找自定义Scrapy命令的模块。这用于为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`文件的入口点中添加一个部分来从外部库添加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',
    ],
  },
 )
```