Scrapy Documentation

Scrapy developers

2019 年 06 月 28 日

第一步

1 获得帮助 **3**

2 第一步 **5**

2.1 预览 Scrapy 5
2.2 安装指南 8
2.3 Scrapy 教程 12
2.4 例子 27

3 基本概念 **29**

3.1 命令行工具 29
3.2 爬虫器 41
3.3 选择器 55
3.4 Items 74
3.5 Item 加载器 79
3.6 Scrapy shell 93
3.7 Item 管道 99
3.8 Feed 导出 104
3.9 请求和响应 110
3.10 链接提取 124
3.11 设置 126
3.12 异常 154

4 内置服务 **157**

4.1 日志 157
4.2 统计数据集合 161
4.3 发送邮件 163
4.4 远程控制台 166
4.5 Web Service 170

5 解决具体问题 **171**
5.1 常见问答 . . . 171
5.2 Debug 爬虫器 . . . 177
5.3 爬虫器约束 . . . 180
5.4 常见实践 . . . 182
5.5 并发爬虫 . . . 187
5.6 使用浏览器的开发工具进行抓取 . . . 190
5.7 内存泄漏调试 . . . 196
5.8 下载并处理文件和图片 . . . 202
5.9 部署爬虫器 . . . 211
5.10 爬虫器节流 . . . 212
5.11 爬虫器硬件性能 . . . 214
5.12 作业: 暂停并恢复爬行 . . . 217

6 Scrapy 扩展 **221**
6.1 框架体系 . . . 221
6.2 下载器中间件 . . . 224
6.3 爬虫器中间件 . . . 240
6.4 扩展 . . . 247
6.5 核心 API . . . 253
6.6 信号 . . . 257
6.7 Item 导出文件 . . . 262

7 其他 **271**
7.1 Scrapy 更新 . . . 271
7.2 贡献 Scrapy 代码 . . . 330
7.3 版本控制和 API 稳定性 . . . 335

Python 模块索引 **337**

索引 **339**

Scrapy 是一个快速、高效率的网络爬虫框架，用于抓取 web 站点并从页面中提取结构化的数据。Scrapy 被

Scrapy是一个快速、高效率的开源爬虫框架，用于抓取Web站点并从页面中提取结构化的数据。Scrapy被广泛用于数据挖掘、监测和自动化测试。

第1章

获得帮助

遇到困难了？我们乐意帮你解决！

- 试试*FAQ* – 这里有一些常见问题的解答。
- 寻找具体信息？试试 genindex or modindex。
- 在 StackOverflow using the scrapy tag 里提问或搜索问题。
- 在 Scrapy subreddit 里提问或搜索问题。
- 在 scrapy-users mailing list 文档里搜索问题。
- 在 #scrapy IRC channel 上提问。
- 提交 Scrapy 错误报告请点击 issue tracker。

第2章

2.1预测Scrapy

Scrapy是一个用于抓取和提取网点的结构化数据资料的使用范围，可用于广泛的有使用的应用程序，如数据资料的挖掘、信息处理或历史档案

尽管如此，Scrapy最初是为网页抓取而设计的，但它也可以用API提取数据(例如Amazon Associates网络服务接口)或者说作一个通用网络爬虫

2.1.1爬爬虫虫例子

为了向你展示带来了什么，我们将用一个Scrapy最简单的方法带你浏览一个Scrapy爬虫器的例子子

下面是这个爬虫器的代码码，它可以从http://quotes.toscrape.com网站上抓取名人名，根据上述的页码：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]
```

(下页继续)

(续上页)

```
    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
```

```
                'author': quote.xpath('span/small/text()').get(),
            }

        next_page = response.css('li.next a::attr("href")').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

把编码放到一个Python文件中，文件名如引述_ Spider . py然后用runpider命令运行这个爬虫器：

```
scrapy runspider quotes_spider.py -o quotes.json
```

当它执行完毕后，你会发现目录下多了一个名为 quotes.json 的文件，文件内容是以 JSON 格式储存的，其中包含名言和作者，格式如下（为了美观进行了排版）：

```
[{
    "author": "Jane Austen",
    "text": "\u201cThe person, be it gentleman or lady, who has not pleasure in a good␣
↪novel, must be intolerably stupid.\u201d"
},
{
    "author": "Groucho Marx",
    "text": "\u201cOutside of a dog, a book is man's best friend. Inside of a dog it's␣
↪too dark to read.\u201d"
},
{
    "author": "Steve Martin",
    "text": "\u201cA day without sunshine is like, you know, night.\u201d"
},
...]
```

在此期间发生了什么?

当你运行病爬虫器报价_蜘蛛. py命令之后，Scrapy会找到一定义的爬虫器，以后通过爬虫引杆运行器

侦虫是通发请求给开始_ urls属性中定的url(在本例中,就是幽默分类下的那些url)然后调用默认的回调函数解析，并把响应的对象编码为参传给它。在解析回调函数中，我们用CSS选择器遍

历引《元素》，并把解析性的引名言和作者生成过成器返，找寻并请求一个个链接并继续使用解析方法作回调数

你可以注意到Scrapy的一个主要矛盾：请求的调度是异步的。这意味的Scrapy不需要等待一个请求被完成处

理事务，它可以同时性的发另一个请求或者说这些许其他人的事情。这同样也明白了当些许失败者的处理事务

虽然这使你可以非常快速地进行爬虫(同时可并发多个请求,在可承担者的压力下)，不过Scrapy同样提供了更好的爬虫方法，简单地配置即可。你也可以配置一些其他的数据，比如说下载器配置一个延迟时间，限制

注解: 本例中导出 了一个 JSON 文件, 你可以轻易地更改输出格式（比如 XML 或者 CSV）或者储存到后端（比如 FTP 或者 Amazon S3 ）。你同样可以定义一个*item 管道* 把这些 item 储存到数据库中。

2.1.2 还有什么?

你已经知道了如何用 Scrapy 从一个站点提取和储存 item，但是仅仅是很浅显的了解它。Scrapy 还为爬虫提供了很多强大的功能，比如:

- 用内置的 CSS 选择器和 XPath 语法从 HTML/XML 源中选择和解析数据。甚至可以在其中使用正则表达式来辅助解析。
- 一个交互控制台 (IPython aware)，可以用 CSS 选择器和 XPath 语法来解析数据，在编写和调试爬虫器的时候十分方便。
- 内置支持生成输出 多种格式文件 (JSON, CSV, XML) 和多种后端存储 (FTP, S3, 本地文件系统)。
- 强大的编码支持和检测, 用于处理外部的，不标准的或者损坏的编码。
- 强大的扩展, 允许你植入自己写的函数，信号 和定义好的 API(中间件, 扩展, 以及管道)。
- 广泛的内置扩展功能和中间件处理:
  - 处理 cookies 和 session
  - HTTP 特性包括压缩、身份验证以及缓存
  - 模拟 user-agent
  - robots.txt 协议
  - 爬虫深度限制
  - 更多
- 一个远程控制台 用于连接运行在 Scrapy 进程的 Python 控制台, 以便于自省和调试爬虫程序。
- 还有其他好东西，比如从 Sitemaps and XML/CSV 源导入可复用的爬虫, 一个可以自动下载图片（或者其他和 items 关联的媒体文件）的媒体中间件, 一个 DNS 缓存解析器, 以及更多!

2.1 . 3下一步?

你下一步要做的就是下载Scrapy，跟着教程去创一个完整的Scrapy工作流程并且加进社区感谢您的关注!

2.2安装指南

2.2 . 1安装Scrapy

Scrapy运行在CPython (默认Python实现)和PyPy (从Py Py 5.9开始)下的Python 2.7和Python 3.4以上版本上。

如果您使用的是阿纳康达或者Miniconda，则可以从conda - forge通道安装该软件包，该通道具有Linux、Windows和OS X的最新软件包。

使用conda安装Scrapy，运行：

```
conda install -c conda-forge scrapy
```

Alternatively, if you' re already familiar with installation of Python packages, you can install Scrapy and its dependencies from PyPI with:

```
pip install Scrapy
```

注意，有时这可能需要解决一些Scrapy依赖项的编译问题，这取决于您的操作系统，所以一定要检查Platform特定的安装注释。

我们强烈建议您将Scrapy安装在专用的virtualenv中，以避免与您的系统包发生冲突。

对于更加详细和平台具体的说明，以及故障排除信息，一一研读。

好知道的事情

Scrapy是用纯Python编写的，依赖于几个关键的Python包(除其他外)：

lxml，一个高效的XML和HTML解析器

· parsel，一个写在lxml之上的HTML / XML数据抽取库。

· w3lib，一个用于处理URL和网页编码的多功能助手

•扭曲，一个异步的网络框架

·密码学和py openssl开发包，以应对各种网络级的安全需求

Scrapy测试的最小版本是：

第2章第一步

·扭曲14.0

· lxml 3.4

· pyOpenSSL0.14

Scrapy可以与这些包的旧版本一起工作，但不能保证它会继续工作，因为它没有被测试。

其中一些包本身依赖于非Python包，这些包可能需要额外的安装步骤，具体取决于您的平台。请在下面查看平台的具体指南。

如果遇到与这些依赖项相关的任何麻烦，请参考它们各自的安装说明：

· lxml安装

·密码学安装

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包可以安装在全局(a . k . a系统广)中，也可以安装在用户空间中。我们不建议广泛安装Scrapy系统。

相反，我们建议您在一个所谓的"虚拟环境" (virtualenv)中安装Scrapy。Virtualenvs允许您不与已经安装的Python系统包(这可能会破坏你的一些系统工具和脚本)发生冲突，仍然正常安装与pip (没有sudo等)的包。

要开始使用虚拟环境，请参阅virtualenv安装说明。要在全球范围内安装(在全球范围内安装它实际上有助于这里)，应该是一个运行的问题：

$ [sudo] pip install virtualenv

查看此用户指南，了解如何创建您的虚拟环境。

注释：如果使用Linux或OS X，virtualenvwrapper是创建virtualenvs的便捷工具。

Once you have created a virtualenv, you can install scrapy inside it with `pip`, just like any other Python package. (See *platform-specific guides* below for non-Python dependencies that you may need to install beforehand).

Python virtualenvs can be created to use Python 2 by default, or Python 3 by default.

- If you want to install scrapy with Python 3, install scrapy within a Python 3 virtualenv.
- And if you want to install scrapy with Python 2, install scrapy within a Python 2 virtualenv.

2.2.2 Platform specific installation notes

窗口

虽然在Windows上使用pip安装Scrapy是可能的，但是我们推荐您安装阿纳康达或者Miniconda并使用conda-forge通道的包，这样可以避免大部分的安装问题。

一旦安装了阿纳康达或者Miniconda，安装Scrapy：

```
conda install -c conda-forge scrapy
```

Ubuntu 14.04 or above

Scrapy is currently tested with recent-enough versions of lxml, twisted and pyOpenSSL, and is compatible with recent Ubuntu distributions. But it should support older versions of Ubuntu too, like Ubuntu 14.04, albeit with potential issues with TLS connections.

Don't use the `python-scrapy` package provided by Ubuntu, they are typically too old and slow to catch up with latest Scrapy.

To install scrapy on Ubuntu (or Ubuntu-based) systems, you need to install these dependencies:

```
sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev zlib1g-dev libffi-
↪dev libssl-dev
```

- `python-dev`, `zlib1g-dev`, `libxml2-dev` and `libxslt1-dev` are required for `lxml`
- `libssl-dev` and `libffi-dev` are required for `cryptography`

If you want to install scrapy on Python 3, you'll also need Python 3 development headers:

```
sudo apt-get install python3 python3-dev
```

Inside a *virtualenv*, you can install Scrapy with `pip` after that:

```
pip install scrapy
```

注解：在Debian Jessie (8.0)及以上版本中可以使用相同的非Python依赖项来安装Scrapy。

Mac OS X

Building Scrapy's dependencies requires the presence of a C compiler and development headers. On OS X this is typically provided by Apple's Xcode development tools. To install the Xcode command line tools open a terminal window and run:

```
xcode-select --install
```

There' s a known issue that prevents pip from updating system packages. This has to be addressed to successfully install Scrapy and its dependencies. Here are some proposed solutions:

- *(Recommended)* **Don' t** use system python, install a new, updated version that doesn' t conflict with the rest of your system. Here' s how to do it using the homebrew package manager:
  - Install homebrew following the instructions in https://brew.sh/
  - Update your PATH variable to state that homebrew packages should be used before system packages (Change .bashrc to .zshrc accordantly if you' re using zsh as default shell):

    ```
    echo "export PATH=/usr/local/bin:/usr/local/sbin:$PATH" >> ~/.bashrc
    ```

  - Reload .bashrc to ensure the changes have taken place:

    ```
    source ~/.bashrc
    ```

  - Install python:

    ```
    brew install python
    ```

  - Latest versions of python have pip bundled with them so you won' t need to install it separately. If this is not the case, upgrade python:

    ```
    brew update; brew upgrade python
    ```

- *(Optional)* Install Scrapy inside an isolated python environment.

  This method is a workaround for the above OS X issue, but it' s an overall good practice for managing dependencies and can complement the first method.

  virtualenv is a tool you can use to create virtual environments in python. We recommended reading a tutorial like http://docs.python-guide.org/en/latest/dev/virtualenvs/ to get started.

After any of these workarounds you should be able to install Scrapy:

```
pip install Scrapy
```

PyPy

我们推荐使用最新的PyPy版本。测试的版本为5.9 . 0。对于PyPy3，只进行了Linux安装测试。

现在大多数的废料依赖都有CPython的二元轮，但没有PyPy的二元轮。这意味着这些依赖关系将在安装过程中建立。在OS X上，你很可能会面临一个构建的问题

密码学依赖，解决这个问题的方案在这里进行了描述，即酝酿安装openssl开发包，然后导出这个命令推荐的标志(仅在安装报废时需要)。在Linux上安装，除了安装构建依赖关系外，并没有什么特殊的问题。在Windows上使用PyPy安装Scrapy未进行测试。

通过运行刮刀台可以检查刮刀是否安装正确。如果该命令给出的错误如TypeError：. . .得到了2个未预期的关键字参数，这意味着setuptools无法拾取一个PyPy - specific依赖。为了解决这个问题，运行pip install ' PyPyDispatcher > = 2.1.0 '。

2.2 . 3故障处理

AttributeError：'模块'对象没有属性' OP _ NO _ TLSv1 _ 1 '

安装或升级Scrapy、Twisted或py openssl开发包后，可能会得到一个异常，如下回溯：

```
[…]
  File "[…]/site-packages/twisted/protocols/tls.py", line 63, in <module>
    from twisted.internet._sslverify import _setAcceptableProtocols
  File "[…]/site-packages/twisted/internet/_sslverify.py", line 38, in <module>
    TLSVersion.TLSv1_1: SSL.OP_NO_TLSv1_1,
AttributeError: 'module' object has no attribute 'OP_NO_TLSv1_1'
```

The reason you get this exception is that your system or virtual environment has a version of pyOpenSSL that your version of Twisted does not support.

To install a version of pyOpenSSL that your version of Twisted supports, reinstall Twisted with the `tls` extra option:

```
pip install twisted[tls]
```

详见第2473期。

2 . 3刮教程

在本教程中，我们假设你的系统已经安装了Scrapy，如果没有，点击Scrapy安装程序

我们即将爬取一个名列举了了人名言的网站，引自toscrape . com

本教程将带领您完成这些任务：

1 .创建一个新的Scrapy项目

2 .编写蜘蛛爬取站点并提取数据

12 3 .使用命令行导出报废数据

4 .将蜘蛛改为递归地跟随链接

5 .使用蜘蛛论据

Scrapy是用Python编写的。如果你是新的语言，你可能想从了解语言是什么样的开始，以获得最大的Scrapy。

如果您已经熟悉了其他语言，并且想要快速学习Python，那么Python教程就是一个很好的资源。

如果你刚刚接触编程，想从Python开始，下面的书可能对你有用：

•用python将枯燥的东西自动化

•如何像计算机科学家一样思考

•学习python 3困难的方法

您还可以看一下这个针对非程序员的Python资源列表，以及在learningpython - subreddit中推荐的资源。

(1)创建项目

在开始刮擦之前，你将不得不建立一个新的Scrapy项目。输入一个您想要存储代码并运行的目录：

```
scrapy startproject tutorial
```

This will create a `tutorial` directory with the following contents:

```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

2 . 3 .刮削教程13

Scrapy Documentation

2.3 . 2我们的第一个蜘蛛

Spiders是您定义的、Scrapy用来从网站(或者一组网站)抓取信息的类。它们必须是零碎的亚类。Spider和定义要发出的初始请求，可选的是如何跟踪页面中的链接，以及如何解析下载的页面内容来提取数据。

这是我们第一个蜘蛛的代码。将其保存在项目中的教程/ Spiders目录下的一个名为报价_ Spider . py的文件中：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s' % filename)
```

正如你所看到的，我们的Spider子类是零碎的。Spider并定义了一些属性和方法：

·名称：识别蜘蛛。在一个项目内部必须是唯一的，即不能设置相同的名称

对于不同的蜘蛛。

· start _ request ( )：必须返回一个可迭代的请求(可以返回请求列表或者编写生成器函数)，Spider将从该请求(可以返回请求列表或者编写生成器函数)开始爬行。后续的请求将从这些初始请求中依次产生。

•parse ( )：一种方法，它将被调用来处理为每个请求所下载的响应。响应参数是TextResponse的一个实例，它包含页面内容和哈斯弗瑟帮助的方法来处理它。

parse ( )方法通常会对响应进行解析，将报废的数据提取为dicts，并从中发现新的URL，创建新的请求(Request)。

如何运行我们的蜘蛛

要把我们的蜘蛛放到工作中，去项目的顶层目录并运行：

```
scrapy crawl quotes
```

This command runs the spider with name quotes that we've just added, that will send some requests for the quotes.toscrape.com domain. You will get an output similar to this:

.. (为简洁省略)2016-12-16 21：24：05 [Scrapy.core.engine] INFO：Spider open2016-12-16 21：24：05 [Scrapy. extensions.logstats] INFO：抓取0页(0页/分钟)，␣→抓取0项(以0项/ min)

2016-12-16 21：24：05 [Scrapy. extensions. telnet] DEBUG：Telnet控制台收听127.0。

→0.1:6023

2016-12-16 21：24：05 [Scrapy.Core.Engine]调试：抓取(404) <得到http://quotes.

→Toscrape.Com /基本介绍> (引用者：无)

2016-12-16 21：24：05 [Scrapy.Core.Engine]调试：抓取(200) <得到http://quotes.

→Toscrape.Com / Page / 1 / > (引用者：无)

2016-12-16 21：24：05 [Scrapy.Core.Engine]调试：抓取(200) <得到http://quotes.

→toscrape.com / page / 2 / > (引用者：None) 2016-12-16 21：24：05 [引用者] DEBUG：保存的文件引用者- 1 . html2016-12-16 21：24：05 [引用者] DEBUG：保存的文件引用者- 2 . html2016-12-16 21：24：05 [Scrapy.core.engine] INFO：关闭蜘蛛(已完成) . . .

现在，检查当前目录中的文件。需要注意的是，已经创建了两个新的文件：报价- 1 . html和报价- 2 . html，其内容为各自的URL，正如我们的解析方法所指示的那样。

注解：如果你想知道为什么我们还没有解析HTML，等等，我们很快就会覆盖。

What just happened under the hood?

Scrapy schedules the *scrapy.Request* objects returned by the `start_requests` method of the Spider. Upon receiving a response for each one, it instantiates *Response* objects and calls the callback method associated with the request (in this case, the `parse` method) passing the response as argument.

A shortcut to the start_requests method

Instead of implementing a *start_requests()* method that generates *scrapy.Request* objects from URLs, you can just define a *start_urls* class attribute with a list of URLs. This list will then be used by the

---

Scrapy Documentation

默认实现start _ request ( )，为蜘蛛创建初始请求：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
        'http://quotes.toscrape.com/page/2/',
    ]

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
```

The *parse()* method will be called to handle each of the requests for those URLs, even though we haven' t explicitly told Scrapy to do so. This happens because *parse()* is Scrapy' s default callback method, which is called for requests without an explicitly assigned callback.

Extracting data

The best way to learn how to extract data with Scrapy is trying selectors using the *Scrapy shell*. Run:

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

注解: Remember to always enclose urls in quotes when running Scrapy shell from command-line, otherwise urls containing arguments (ie. `&` character) will not work.

On Windows, use double quotes instead:

```
scrapy shell "http://quotes.toscrape.com/page/1/"
```

You will see something like:

```
[ ... Scrapy log here ... ]
2016-09-19 12:09:27 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.
↪toscrape.com/page/1/> (referer: None)
[s] Available Scrapy objects:
```

注解: Remember to always enclose urls in quotes when running Scrapy shell from command-line, otherwise urls containing arguments (ie. `&` character) will not work.

On Windows, use double quotes instead:

```
scrapy shell "http://quotes.toscrape.com/page/1/"
```

You will see something like:

```
[ ... Scrapy log here ... ]
2016-09-19 12:09:27 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.
↪toscrape.com/page/1/> (referer: None)
[s] Available Scrapy objects:
```

(下页继续)

16第2章第一步