Scrapy 0.25 文档

本文档涵盖了所有Scrapy的内容。

获得帮助

遇到问题了？我们来帮您！

- 查看下 *FAQ*，这里有些常见的问题的解决办法。
- 寻找详细的信息？试试 *索引* 或者 *模块索引* 。
- 您可以在 scrapy-users的邮件列表 [http://groups.google.com/group/scrapy-users/] 中寻找内容，或者 提问问题 [http://groups.google.com/group/scrapy-users/]
- 在 #scrapy IRC channel 提问
- 在 issue tracker [https://github.com/scrapy/scrapy/issues] 中提交Scrapy的bug

第一步

初窥Scrapy
: 了解Scrapy如何祝你一臂之力。

安装指南
: 安装Scrapy。

Scrapy入门教程
: 编写您的第一个Scrapy项目。

例子
: 通过把玩已存在的Scrapy项目来学习更多内容。

基本概念

命令行工具(Command line tools)
: 学习用于管理Scrapy项目的命令行工具

Items
: 定义爬取的数据

Spiders
: 编写爬取网站的规则

选择器(Selectors)
: 使用XPath提取网页的数据

Scrapy终端(Scrapy shell)
: 在交互环境中测试提取数据的代码

Item Loaders
: 使用爬取到的数据填充item

Item Pipeline
: 后处理(Post-process)，存储爬取的数据

Feed exports
: 以不同格式输出爬取数据到不同的存储端

Link Extractors
: 方便用于提取后续跟进链接的类。

内置服务

Logging
: 了解Scrapy提供的logging功能。

数据收集(Stats Collection)
: 收集爬虫运行数据

发送email
: 当特定事件发生时发送邮件通知

Telnet终端(Telnet Console)
: 使用内置的Python终端检查运行中的crawler(爬虫)

Web Service
使用web service对您的爬虫进行监控和管理

解决特定问题

常见问题(FAQ)
常见问题的解决办法。

调试(Debugging)Spiders
学习如何对scrapy spider的常见问题进行debug。

Spiders Contracts
学习如何使用contract来测试您的spider。

实践经验(Common Practices)
熟悉Scrapy的一些惯例做法。

通用爬虫(Broad Crawls)
调整Scrapy来适应并发爬取大量网站(a lot of domains)。

借助Firefox来爬取
了解如何使用Firefox及其他有用的插件来爬取数据。

使用Firebug进行爬取
了解如何使用Firebug来爬取数据。

调试内存溢出
了解如何查找并让您的爬虫避免内存泄露。

下载项目图片
下载爬取的item中的图片。

Ubuntu 软件包
在Ubuntu下下载最新的Scrapy。

Scrapyd
在生产环境中部署您的Scrapy项目。

自动限速(AutoThrottle)扩展
根据负载(load)动态调节爬取速度。

Benchmarking
在您的硬件平台上测试Scrapy的性能。

Jobs: 暂停，恢复爬虫
学习如何停止和恢复爬虫

DjangoItem
使用Django模型编写爬取的item

扩展Scrapy

架构概览
了解Scrapy架构。

下载器中间件(Downloader Middleware)
自定义页面被请求及下载操作。

Spider中间件(Middleware)
自定义spider的输入与输出。

扩展(Extensions)
提供您自定义的功能来扩展Scrapy

核心API
在extension(扩展)和middleware(中间件)使用api来扩展Scrapy的功能

参考

命令行工具(Command line tools)
学习命令行工具及所有 *可用的命令* 。

Requests and Responses
了解代表HTTP请求和回复的request,response类

Settings
了解如何配置Scrapy及所有 *可用的设置* 。

信号(Signals)
查看如何使用及所有可用的信号

异常(Exceptions)
查看所有可用的exception以及相应的意义。

Item Exporters
快速将您爬取到的item导出到文件中(XML, CSV等格式)

其他

Release notes
了解最近的Scrapy版本的修改。

Contributing to Scrapy
了解如何为Scrapy项目做出贡献。

Versioning and API Stability
了解Scrapy如何命名版本以及API的稳定性。

试验阶段特性
了解最新的特性

讨论

初窥Scrapy

Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架。 可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。

其最初是为了 页面抓取 [http://en.wikipedia.org/wiki/Screen_scraping] (更确切来说, 网络抓取 [http://en.wikipedia.org/wiki/Web_scraping])所设计的， 也可以应用在获取API所返回的数据(例如 Amazon Associates Web Services [http://aws.amazon.com/associates/]) 或者通用的网络爬虫。

本文档将通过介绍Scrapy背后的概念使您对其工作原理有所了解， 并确定Scrapy是否是您所需要的。

当您准备好开始您的项目后，您可以参考 *入门教程* 。

选择一个网站

当您需要从某个网站中获取信息，但该网站未提供API或能通过程序获取信息的机制时， Scrapy可以助你一臂之力。

以 Mininova [http://www.mininova.org] 网站为例，我们想要获取今日添加的所有种子的URL、 名字、描述以及文件大小信息。

今日添加的种子列表可以通过这个页面找到:

> http://www.mininova.org/today

定义您想抓取的数据

第一步是定义我们需要爬取的数据。在Scrapy中， 这是通过 *Scrapy Items* 来完成的。(在本例子中为种子文件)

我们定义的Item:

```
import scrapy

class TorrentItem(scrapy.Item):
    url = scrapy.Field()
    name = scrapy.Field()
    description = scrapy.Field()
    size = scrapy.Field()
```

编写提取数据的Spider

第二步是编写一个spider。其定义了初始URL(http://www.mininova.org/today)、 针对后续链接的规则以及从页面中提取数据的规则。

通过观察页面的内容可以发现，所有种子的URL都类似 `http://www.mininova.org/tor/NUMBER` 。 其中， `NUMBER` 是一个整数。 根据此规律，我们可以定义需要进行跟进的链接的正则表达式: `/tor/\d+` 。

我们使用 XPath [http://www.w3.org/TR/xpath] 来从页面的HTML源码中选择需要提取的数据。 以其中一个种子文件的页面为例:

> http://www.mininova.org/tor/2676093

观察HTML页面源码并创建我们需要的数据(种子名字，描述和大小)的XPath表达式。

通过观察，我们可以发现文件名是包含在 `<h1>` 标签中的:

```
<h1>Darwin - The Evolution Of An Exhibition</h1>
```

与此对应的XPath表达式:

```
//h1/text()
```

种子的描述是被包含在 `id="description"` 的 `<div>` 标签中:

```
<h2>Description:</h2>

<div id="description">
Short documentary made for Plymouth City Museum and Art Gallery regarding the setup of an exhibit about Charles Darwin in

...
```

对应获取描述的XPath表达式:

```
//div[@id='description']
```

文件大小的信息包含在 `id=specifications` 的 `<div>` 的第二个 `<p>` 标签中:

```
<div id="specifications">
```

```
<p>
<strong>Category:</strong>
<a href="/cat/4">Movies</a> &gt; <a href="/sub/35">Documentary</a>
</p>

<p>
<strong>Total size:</strong>
150.62 megabyte</p>
```

选择文件大小的XPath表达式:

```
//div[@id='specifications']/p[2]/text()[2]
```

关于XPath的详细内容请参考 XPath参考 [http://www.w3.org/TR/xpath] 。

最后，结合以上内容给出spider的代码:

```
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors import LinkExtractor

class MininovaSpider(CrawlSpider):

    name = 'mininova'
    allowed_domains = ['mininova.org']
    start_urls = ['http://www.mininova.org/today']
    rules = [Rule(LinkExtractor(allow=['/tor/\d+']), 'parse_torrent')]

    def parse_torrent(self, response):
        torrent = TorrentItem()
        torrent['url'] = response.url
        torrent['name'] = response.xpath("//h1/text()").extract()
        torrent['description'] = response.xpath("//div[@id='description']").extract()
        torrent['size'] = response.xpath("//div[@id='info-left']/p[2]/text()[2]").extract()
        return torrent
```

`TorrentItem` 的定义在 *上面* 。

执行spider，获取数据

终于，我们可以运行spider来获取网站的数据，并以JSON格式存入到 `scraped_data.json` 文件中:

```
scrapy crawl mininova -o scraped_data.json
```

命令中使用了 *feed导出* 来导出JSON文件。您可以修改导出格式(XML或者CSV)或者存储后端(FTP或者 Amazon S3 [http://aws.amazon.com/s3/])，这并不困难。

同时，您也可以编写 *item管道* 将item存储到数据库中。

查看提取到的数据

执行结束后，当您查看 `scraped_data.json` , 您将看到提取到的item:

```
[{"url": "http://www.mininova.org/tor/2676093", "name": ["Darwin - The Evolution Of An Exhibition"], "description": ["Sho
# ... other items ...
]
```

由于 *selectors* 返回list, 所以值都是以list存储的(除了 `url` 是直接赋值之外)。 如果您想要保存单个数据或者对数据执行额外的处理,那将是 *Item Loaders* 发挥作用的地方。

还有什么？

您已经了解了如何通过Scrapy提取存储网页中的信息，但这仅仅只是冰山一角。Scrapy提供了很多强大的特性来使得爬取更为简单高效，例如:

- HTML, XML源数据 *选择及提取* 的内置支持
- 提供了一系列在spider之间共享的可复用的过滤器(即 *Item Loaders*)，对智能处理爬取数据提供了内置支持。
- 通过 *feed导出* 提供了多格式(JSON、CSV、XML)，多存储后端(FTP、S3、本地文件系统)的内置支持
- 提供了media pipeline，可以 *自动下载* 爬取到的数据中的图片(或者其他资源)。
- 高扩展性。您可以通过使用 *signals* ，设计好的API(中间件, *extensions*, *pipelines*)来定制实现您的功能。
- 内置的中间件及扩展为下列功能提供了支持:
  - cookies and session 处理
  - HTTP 压缩
  - HTTP 认证
  - HTTP 缓存
  - user-agent模拟
  - robots.txt
  - 爬取深度限制

    - 其他
- 针对非英语语系中不标准或者错误的编码声明, 提供了自动检测以及健壮的编码支持。
- 支持根据模板生成爬虫。在加速爬虫创建的同时，保持在大型项目中的代码更为一致。详细内容请参阅 **genspider** 命令。
- 针对多爬虫下性能评估、失败检测，提供了可扩展的 *状态收集工具* 。
- 提供 *交互式shell终端* , 为您测试XPath表达式，编写和调试爬虫提供了极大的方便
- 提供 *System service*, 简化在生产环境的部署及运行
- 内置 *Telnet终端* ，通过在Scrapy进程中钩入Python终端，使您可以查看并且调试爬虫
- *Logging* 为您在爬取过程中捕捉错误提供了方便
- 支持 Sitemaps [http://www.sitemaps.org] 爬取
- 具有缓存的DNS解析器

接下来

下一步当然是 下载Scrapy [http://scrapy.org/download/] 了， 您可以阅读 *入门教程* 并加入 社区 [http://scrapy.org/community/] 。感谢您的支持!

讨论

安装指南

安装Scrapy

注解

请先阅读 *平台安装指南*.

下列的安装步骤假定您已经安装好下列程序:

- Python [http://www.python.org] 2.7
- Python Package: pip [http://www.pip-installer.org/en/latest/installing.html] and setuptools [https://pypi.python.org/pypi/setuptools]. 现在 pip [http://www.pip-installer.org/en/latest/installing.html] 依赖 setuptools [https://pypi.python.org/pypi/setuptools]， 如果未安装，则会自动安装 setuptools [https://pypi.python.org/pypi/setuptools] 。
- lxml [http://lxml.de/]. 大多数Linux发行版自带了lxml。如果缺失，请查看http://lxml.de/installation.html
- OpenSSL [https://pypi.python.org/pypi/pyOpenSSL]. 除了Windows(请查看 *平台安装指南*)之外的系统都已经提供。

您可以使用pip来安装Scrapy(推荐使用pip来安装Python package).

使用pip安装:

```
pip install Scrapy
```

平台安装指南

Windows

- 从 http://python.org/download/ 上安装Python 2.7.

  您需要修改 `PATH` 环境变量，将Python的可执行程序及额外的脚本添加到系统路径中。将以下路径添加到 `PATH` 中:

  ```
  C:\Python27\;C:\Python27\Scripts\;
  ```

  请打开命令行，并且运行以下命令来修改 `PATH`:

  ```
  c:\python27\python.exe c:\python27\tools\scripts\win_add2path.py
  ```

  关闭并重新打开命令行窗口，使之生效。运行接下来的命令来确认其输出所期望的Python版本:

  ```
  python --version
  ```

- 从 http://sourceforge.net/projects/pywin32/ 安装 *pywin32*

  请确认下载符合您系统的版本(win32或者amd64)

- 从 https://pip.pypa.io/en/latest/installing.html 安装 pip [http://www.pip-installer.org/en/latest/installing.html]

  打开命令行窗口，确认 `pip` 被正确安装:

  ```
  pip --version
  ```

- 到目前为止Python 2.7 及 `pip` 已经可以正确运行了。接下来安装Scrapy:

  ```
  pip install Scrapy
  ```

Ubuntu 9.10及以上版本

不要 使用Ubuntu提供的 `python-scrapy`，相较于最新版的Scrapy，该包版本太旧，并且运行速度也较为缓慢。

您可以使用官方提供的 *Ubuntu Packages* 。该包解决了全部依赖问题，并且与最新的bug修复保持持续更新。

Archlinux

您可以依照通用的方式或者从 *AUR Scrapy package* 来安装Scrapy:

```
yaourt -S scrapy
```

讨论

Scrapy入门教程

在本篇教程中，我们假定您已经安装好Scrapy。 如若不然，请参考 *安装指南* 。

接下来以 Open Directory Project(dmoz) (dmoz) [http://www.dmoz.org/] 为例来讲述爬取。

本篇教程中将带您完成下列任务:

1. 创建一个Scrapy项目
2. 定义提取的Item
3. 编写爬取网站的 *spider* 并提取 *Item*
4. 编写 *Item Pipeline* 来存储提取到的Item(即数据)

Scrapy由 Python [http://www.python.org] 编写。如果您刚接触并且好奇这门语言的特性以及Scrapy的详情， 对于已经熟悉其他语言并且想快速学习Python的编程老手， 我们推荐 Learn Python The Hard Way [http://learnpythonthehardway.org/book/] ， 对于想从Python开始学习的编程新手， 非程序员的Python学习资料列表 [http://wiki.python.org/moin/BeginnersGuide/NonProgrammers] 将是您的选择。

创建项目

在开始爬取之前，您必须创建一个新的Scrapy项目。 进入您打算存储代码的目录中，运行下列命令:

```
scrapy startproject tutorial
```

该命令将会创建包含下列内容的 `tutorial` 目录:

```
tutorial/
    scrapy.cfg
    tutorial/
        __init__.py
        items.py
        pipelines.py
        settings.py
        spiders/
            __init__.py
            ...
```

这些文件分别是:

- `scrapy.cfg`: 项目的配置文件
- `tutorial/`: 该项目的python模块。之后您将在此加入代码。
- `tutorial/items.py`: 项目中的item文件.
- `tutorial/pipelines.py`: 项目中的pipelines文件.
- `tutorial/settings.py`: 项目的设置文件.
- `tutorial/spiders/`: 放置spider代码的目录.

定义Item

Item 是保存爬取到的数据的容器；其使用方法和python字典类似， 并且提供了额外保护机制来避免拼写错误导致的未定义字段错误。

类似在ORM中做的一样，您可以通过创建一个 **`scrapy.Item`** 类， 并且定义类型为 **`scrapy.Field`** 的类属性来定义一个Item。 (如果不了解ORM, 不用担心，您会发现这个步骤非常简单)

首先根据需要从dmoz.org获取到的数据对item进行建模。 我们需要从dmoz中获取名字，url，以及网站的描述。 对此，在item中定义相应的字段。编辑 `tutorial` 目录中的 `items.py` 文件:

```
import scrapy

class DmozItem(scrapy.Item):
    title = scrapy.Field()
    link = scrapy.Field()
    desc = scrapy.Field()
```

一开始这看起来可能有点复杂，但是通过定义item， 您可以很方便的使用Scrapy的其他方法。而这些方法需要知道您的item的定义。

编写第一个爬虫(Spider)

Spider是用户编写用于从单个网站(或者一些网站)爬取数据的类。

其包含了一个用于下载的初始URL，如何跟进网页中的链接以及如何分析页面中的内容， 提取生成 *item* 的方法。

为了创建一个Spider，您必须继承 **`scrapy.Spider`** 类， 且定义以下三个属性:

- **`name`**: 用于区别Spider。 该名字必须是唯一的，您不可以为不同的Spider设定相同的名字。

- start_urls: 包含了Spider在启动时进行爬取的url列表。 因此，第一个被获取到的页面将是其中之一。 后续的URL则从初始的URL获取到的数据中提取。
- parse() 是spider的一个方法。 被调用时，每个初始URL完成下载后生成的 Response 对象将会作为唯一的参数传递给该函数。 该方法负责解析返回的数据(response data)，提取数据(生成item)以及生成需要进一步处理的URL的 Request 对象。

以下为我们的第一个Spider代码，保存在 tutorial/spiders 目录下的 dmoz_spider.py 文件中:

```
import scrapy

class DmozSpider(scrapy.spider.Spider):
    name = "dmoz"
    allowed_domains = ["dmoz.org"]
    start_urls = [
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
    ]

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

爬取

进入项目的根目录，执行下列命令启动spider:

```
scrapy crawl dmoz
```

crawl dmoz 启动用于爬取 dmoz.org 的spider，您将得到类似的输出:

```
2014-01-23 18:13:07-0400 [scrapy] INFO: Scrapy started (bot: tutorial)
2014-01-23 18:13:07-0400 [scrapy] INFO: Optional features available: ...
2014-01-23 18:13:07-0400 [scrapy] INFO: Overridden settings: {}
2014-01-23 18:13:07-0400 [scrapy] INFO: Enabled extensions: ...
2014-01-23 18:13:07-0400 [scrapy] INFO: Enabled downloader middlewares: ...
2014-01-23 18:13:07-0400 [scrapy] INFO: Enabled spider middlewares: ...
2014-01-23 18:13:07-0400 [scrapy] INFO: Enabled item pipelines: ...
2014-01-23 18:13:07-0400 [dmoz] INFO: Spider opened
2014-01-23 18:13:08-0400 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Reso
2014-01-23 18:13:09-0400 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Book
2014-01-23 18:13:09-0400 [dmoz] INFO: Closing spider (finished)
```

查看包含 [dmoz] 的输出，可以看到输出的log中包含定义在 start_urls 的初始URL，并且与spider中是一一对应的。在log中可以看到其没有指向其他页面((referer:None))。

除此之外，更有趣的事情发生了。就像我们 parse 方法指定的那样，有两个包含url所对应的内容的文件被创建了: *Book* , *Resources* 。

刚才发生了什么？

Scrapy为Spider的 start_urls 属性中的每个URL创建了 scrapy.Request 对象，并将 parse 方法作为回调函数(callback)赋值给了Request。

Request对象经过调度，执行生成 scrapy.http.Response 对象并送回给spider parse() 方法。

提取Item

Selectors选择器简介

从网页中提取数据有很多方法。Scrapy使用了一种基于 XPath [http://www.w3.org/TR/xpath] 和 CSS [http://www.w3.org/TR/selectors] 表达式机制: *Scrapy Selectors* 。 关于selector和其他提取机制的信息请参考 *Selector文档* 。

这里给出XPath表达式的例子及对应的含义:

- /html/head/title: 选择HTML文档中 <head> 标签内的 <title> 元素
- /html/head/title/text(): 选择上面提到的 <title> 元素的文字
- //td: 选择所有的 <td> 元素
- //div[@class="mine"]: 选择所有具有 class="mine" 属性的 div 元素

上边仅仅是几个简单的XPath例子，XPath实际上要比这远远强大的多。 如果您想了解的更多，我们推荐 这篇XPath教程 [http://www.w3school.com.cn/xpath/index.asp] 。

为了配合XPath，Scrapy除了提供了 Selector 之外，还提供了方法来避免每次从response中提取数据时生成selector的麻烦。

Selector有四个基本的方法(点击相应的方法可以看到详细的API文档):

- xpath(): 传入xpath表达式，返回该表达式所对应的所有节点的selector list列表 。
- css(): 传入CSS表达式，返回该表达式所对应的所有节点的selector list列表.
- extract(): 序列化该节点为unicode字符串并返回list。

- `re()`: 根据传入的正则表达式对数据进行提取，返回unicode字符串list列表。

在Shell中尝试Selector选择器

为了介绍Selector的使用方法，接下来我们将要使用内置的 *Scrapy shell* 。Scrapy Shell需要您预装好IPython(一个扩展的Python终端)。

您需要进入项目的根目录，执行下列命令来启动shell:

```
scrapy shell "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/"
```

注解

当您在终端运行Scrapy时，请一定记得给url地址加上引号，否则包含参数的url(例如 `&` 字符)会导致Scrapy运行失败。

shell的输出类似:

```
[ ... Scrapy log here ... ]

2015-01-07 22:01:53+0800 [domz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Book
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x02CE2530>
[s]   item       {}
[s]   request    <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
[s]   response   <200 http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
[s]   sel        <Selector xpath=None data=u'<html lang="en">\r\n<head>\r\n<meta http-equ'>
[s]   settings   <CrawlerSettings module=<module 'tutorial.settings' from 'tutorial\settings.pyc'>>
[s]   spider     <DomzSpider 'domz' at 0x302e350>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

>>>
```

当shell载入后，您将得到一个包含response数据的本地 `response` 变量。输入 `response.body` 将输出response的包体， 输出 `response.headers` 可以看到response的包头。

更为重要的是，当输入 `response.selector` 时， 您将获取到一个可以用于查询返回数据的selector(选择器)， 以及映射到 `response.selector.xpath()` 、 `response.selector.css()` 的 快捷方法(shortcut): `response.xpath()` 和 `response.css()` 。

同时，shell根据response提前初始化了变量 `sel` 。该selector根据response的类型自动选择最合适的分析规则(XML vs HTML)。

让我们来试试:

```
In [1]: sel.xpath('//title')
Out[1]: [<Selector xpath='//title' data=u'<title>Open Directory - Computers: Progr'>]

In [2]: sel.xpath('//title').extract()
Out[2]: [u'<title>Open Directory - Computers: Programming: Languages: Python: Books</title>']

In [3]: sel.xpath('//title/text()')
Out[3]: [<Selector xpath='//title/text()' data=u'Open Directory - Computers: Programming:'>]

In [4]: sel.xpath('//title/text()').extract()
Out[4]: [u'Open Directory - Computers: Programming: Languages: Python: Books']

In [5]: sel.xpath('//title/text()').re('(\w+):')
Out[5]: [u'Computers', u'Programming', u'Languages', u'Python']
```

提取数据

现在，我们来尝试从这些页面中提取些有用的数据。

您可以在终端中输入 `response.body` 来观察HTML源码并确定合适的XPath表达式。不过，这任务非常无聊且不易。您可以考虑使用Firefox的Firebug扩展来使得工作更为轻松。详情请参考 *使用Firebug进行爬取* 和 *借助Firefox来爬取* 。

在查看了网页的源码后，您会发现网站的信息是被包含在 *第二个* `<ul>` 元素中。

我们可以通过这段代码选择该页面中网站列表里所有 `<li>` 元素:

```
sel.xpath('//ul/li')
```

网站的描述:

```
sel.xpath('//ul/li/text()').extract()
```

网站的标题:

```
sel.xpath('//ul/li/a/text()').extract()
```

以及网站的链接:

```
sel.xpath('//ul/li/a/@href').extract()
```

之前提到过，每个 `.xpath()` 调用返回selector组成的list，因此我们可以拼接更多的 `.xpath()` 来进一步获取某个节点。我们将在下边使用这样的特性:

```
for sel in response.xpath('//ul/li'):
    title = sel.xpath('a/text()').extract()
    link = sel.xpath('a/@href').extract()
    desc = sel.xpath('text()').extract()
    print title, link, desc
```

注解

关于嵌套selctor的更多详细信息，请参考 *嵌套选择器(selectors)* 以及 *选择器(Selectors)* 文档中的 *使用相对XPaths* 部分。

在我们的spider中加入这段代码:

```
import scrapy

class DmozSpider(scrapy.Spider):
    name = "dmoz"
    allowed_domains = ["dmoz.org"]
    start_urls = [
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
    ]

    def parse(self, response):
        for sel in response.xpath('//ul/li'):
            title = sel.xpath('a/text()').extract()
            link = sel.xpath('a/@href').extract()
            desc = sel.xpath('text()').extract()
            print title, link, desc
```

现在尝试再次爬取dmoz.org，您将看到爬取到的网站信息被成功输出:

```
scrapy crawl dmoz
```

使用item

`Item` 对象是自定义的python字典。 您可以使用标准的字典语法来获取到其每个字段的值。(字段即是我们之前用Field赋值的属性):

```
>>> item = DmozItem()
>>> item['title'] = 'Example title'
>>> item['title']
'Example title'
```

一般来说，Spider将会将爬取到的数据以 `Item` 对象返回。所以为了将爬取的数据返回，我们最终的代码将是:

```
import scrapy

from tutorial.items import DmozItem

class DmozSpider(scrapy.Spider):
    name = "dmoz"
    allowed_domains = ["dmoz.org"]
    start_urls = [
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
    ]

    def parse(self, response):
        for sel in response.xpath('//ul/li'):
            item = DmozItem()
            item['title'] = sel.xpath('a/text()').extract()
            item['link'] = sel.xpath('a/@href').extract()
            item['desc'] = sel.xpath('text()').extract()
            yield item
```

注解

您可以在 dirbot [https://github.com/scrapy/dirbot] 项目中找到一个具有完整功能的spider。该项目可以通过 https://github.com/scrapy/dirbot 找到。

现在对dmoz.org进行爬取将会产生 `DmozItem` 对象:

```
[dmoz] DEBUG: Scraped from <200 http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
     {'desc': [u' - By David Mertz; Addison Wesley. Book in progress, full text, ASCII format. Asks for feedback. [author
      'link': [u'http://gnosis.cx/TPiP/'],
```

```
     'title': [u'Text Processing in Python']}
[dmoz] DEBUG: Scraped from <200 http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
    {'desc': [u' - By Sean McGrath; Prentice Hall PTR, 2000, ISBN 0130211192, has CD-ROM. Methods to build XML applicati
     'link': [u'http://www.informit.com/store/product.aspx?isbn=0130211192'],
     'title': [u'XML Processing with Python']}
```

保存爬取到的数据

最简单存储爬取的数据的方式是使用 *Feed exports*:

```
scrapy crawl dmoz -o items.json
```

该命令将采用 JSON [http://en.wikipedia.org/wiki/JSON] 格式对爬取的数据进行序列化，生成 `items.json` 文件。

在类似本篇教程里这样小规模的项目中，这种存储方式已经足够。 如果需要对爬取到的item做更多更为复杂的操作，您可以编写 *Item Pipeline* 。 类似于我们在创建项目时对Item做的，用于您编写自己的 `tutorial/pipelines.py` 也被创建。 不过如果您仅仅想要保存item，您不需要实现任何的pipeline。

下一步

本篇教程仅介绍了Scrapy的基础，还有很多特性没有涉及。请查看 *初窥Scrapy* 章节中的 *还有什么？* 部分,大致浏览大部分重要的特性。

接着，我们推荐您把玩一个例子(查看 *例子*)，而后继续阅读 *基本概念* 。

讨论

例子

学习的最好方法就是参考例子，Scrapy也不例外。Scrapy提供了一个叫做 dirbot [https://github.com/scrapy/dirbot] 的样例项目供您把玩学习。其包含了在教程中介绍的dmoz spider。

您可以通过 https://github.com/scrapy/dirbot 找到 dirbot [https://github.com/scrapy/dirbot] 。其包含了README文件，详细介绍了项目的内容。

如果您熟悉git，您可以checkout代码。或者您可以点击 Downloads [https://github.com/scrapy/dirbot/archives/master] 来下载项目的tarball或者zip的压缩包。

Snipplr上的scrapy标签 [http://snipplr.com/all/tags/scrapy/] 是用来分享spider，middeware，extension或者script代码片段。欢迎(并鼓励)在那分享您的代码。

讨论

命令行工具(Command line tools)

0.10 新版功能.

Scrapy是通过 `scrapy` 命令行工具进行控制的。 这里我们称之为 “Scrapy tool” 以用来和子命令进行区分。 对于子命令，我们称为 “command” 或者 “Scrapy commands”。

Scrapy tool 针对不同的目的提供了多个命令，每个命令支持不同的参数和选项。

默认的Scrapy项目结构

在开始对命令行工具以及子命令的探索前，让我们首先了解一下Scrapy的项目的目录结构。

虽然可以被修改，但所有的Scrapy项目默认有类似于下边的文件结构:

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

`scrapy.cfg` 存放的目录被认为是 *项目的根目录* 。该文件中包含python模块名的字段定义了项目的设置。例如:

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

使用 `scrapy` 工具

您可以以无参数的方式启动Scrapy工具。该命令将会给出一些使用帮助以及可用的命令:

```
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项目中运行，当前激活的项目将会显示在输出的第一行。上面的输出就是响应的例子。如果您在一个项目中运行命令将会得到类似的输出:

```
Scrapy X.Y - project: myproject

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

[...]
```

创建项目

一般来说，使用 `scrapy` 工具的第一件事就是创建您的Scrapy项目:

```
scrapy startproject myproject
```

该命令将会在 `myproject` 目录中创建一个Scrapy项目。

接下来，进入到项目目录中:

```
cd myproject
```

这时候您就可以使用 `scrapy` 命令来管理和控制您的项目了。

控制项目

您可以在您的项目中使用 `scrapy` 工具来对其进行控制和管理。

比如，创建一个新的spider:

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

有些Scrapy命令(比如 `crawl`)要求必须在Scrapy项目中运行。 您可以通过下边的 *commands reference* 来了解哪些命令需要在项目中运行，哪些不用。

另外要注意，有些命令在项目里运行时的效果有些许区别。 以fetch命令为例，如果被爬取的url与某个特定spider相关联， 则该命令将会使用spider的动作(spider-overridden behaviours)。 (比如spider指定的 `user_agent`)。 该表现是有意而为之的。一般来说， `fetch` 命令就是用来测试检查spider是如何下载页面。

可用的工具命令(tool commands)

该章节提供了可用的内置命令的列表。每个命令都提供了描述以及一些使用例子。您总是可以通过运行命令来获取关于每个命令的详细内容:

```
scrapy <command> -h
```

您也可以查看所有可用的命令:

```
scrapy -h
```

Scrapy提供了两种类型的命令。一种必须在Scrapy项目中运行(针对项目(Project-specific)的命令)，另外一种则不需要(全局命令)。全局命令在项目中运行时的表现可能会与在非项目中运行有些许差别(因为可能会使用项目的设定)。

全局命令:

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

项目(Project-only)命令:

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

startproject

- 语法: `scrapy startproject <project_name>`
- 是否需要项目: *no*

在 `project_name` 文件夹下创建一个名为 `project_name` 的Scrapy项目。

例子:

```
$ scrapy startproject myproject
```

genspider

- 语法: `scrapy genspider [-t template] <name> <domain>`
- 是否需要项目: *yes*

在当前项目中创建spider。

这仅仅是创建spider的一种快捷方法。该方法可以使用提前定义好的模板来生成spider。您也可以自己创建spider的源码文件。

例子:

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

$ scrapy genspider -d basic
import scrapy

class $classname(scrapy.Spider):
```

```
    name = "$name"
    allowed_domains = ["$domain"]
    start_urls = (
        'http://www.$domain/',
        )

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

```
$ scrapy genspider -t basic example example.com
Created spider 'example' using template 'basic' in module:
  mybot.spiders.example
```

crawl

- 语法: `scrapy crawl <spider>`
- 是否需要项目: *yes*

使用spider进行爬取。

例子:

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

check

- 语法: `scrapy check [-l] <spider>`
- 是否需要项目: *yes*

运行contract检查。

例子:

```
$ 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
```

list

- 语法: `scrapy list`
- 是否需要项目: *yes*

列出当前项目中所有可用的spider。每行输出一个spider。

使用例子:

```
$ scrapy list
spider1
spider2
```

edit

- 语法: `scrapy edit <spider>`
- 是否需要项目: *yes*

使用 **EDITOR** 中设定的编辑器编辑给定的spider

该命令仅仅是提供一个快捷方式。开发者可以自由选择其他工具或者IDE来编写调试spider。

例子:

```
$ scrapy edit spider1
```

fetch

- 语法: `scrapy fetch <url>`
- 是否需要项目: *no*

使用Scrapy下载器(downloader)下载给定的URL，并将获取到的内容送到标准输出。

该命令以spider下载页面的方式获取页面。例如，如果spider有 `USER_AGENT` 属性修改了 User Agent，该命令将会使用该属性。

因此，您可以使用该命令来查看spider如何获取某个特定页面。

该命令如果非项目中运行则会使用默认Scrapy downloader设定。

例子:

```
$ 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)']}
```

view

- 语法: `scrapy view <url>`
- 是否需要项目: *no*

在浏览器中打开给定的URL，并以Scrapy spider获取到的形式展现。 有些时候spider获取到的页面和普通用户看到的并不相同。 因此该命令可以用来检查spider所获取到的页面，并确认这是您所期望的。

例子:

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

shell

- 语法: `scrapy shell [url]`
- 是否需要项目: *no*

以给定的URL(如果给出)或者空(没有给出URL)启动Scrapy shell。 查看 *Scrapy终端(Scrapy shell)* 获取更多信息。

例子:

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

parse

- 语法: `scrapy parse <url> [options]`
- 是否需要项目: *yes*

获取给定的URL并使用相应的spider分析处理。如果您提供 `--callback` 选项，则使用spider的该方法处理，否则使用 `parse` 。

支持的选项:

- `--spider=SPIDER`: 跳过自动检测spider并强制使用特定的spider
- `--a NAME=VALUE`: 设置spider的参数(可能被重复)
- `--callback` or `-c`: spider中用于解析返回(response)的回调函数
- `--pipelines`: 在pipeline中处理item
- `--rules` or `-r`: 使用 `CrawlSpider` 规则来发现用来解析返回(response)的回调函数
- `--noitems`: 不显示爬取到的item
- `--nolinks`: 不显示提取到的链接
- `--nocolour`: 避免使用pygments对输出着色
- `--depth` or `-d`: 指定跟进链接请求的层次数(默认: 1)
- `--verbose` or `-v`: 显示每个请求的详细信息

例子:

```
$ 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  -----------------------------------------------------------------
[]
```

settings

- 语法: `scrapy settings [options]`
- 是否需要项目: *no*

获取Scrapy的设定

在项目中运行时，该命令将会输出项目的设定值，否则输出Scrapy默认设定。

例子:

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

runspider

- 语法: `scrapy runspider <spider_file.py>`
- 是否需要项目: *no*

在未创建项目的情况下，运行一个编写在Python文件中的spider。

例子:

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

version

- 语法: `scrapy version [-v]`
- 是否需要项目: *no*

输出Scrapy版本。配合 `-v` 运行时，该命令同时输出Python, Twisted以及平台的信息，方便bug提交。

deploy

0.11 新版功能.

- 语法: `scrapy deploy [ <target:project> | -l <target> | -L ]`
- 是否需要项目: *yes*

将项目部署到Scrapyd服务。查看 部署您的项目 [http://scrapyd.readthedocs.org/en/latest/deploy.html] 。

bench

0.17 新版功能.

- 语法: `scrapy bench`
- 是否需要项目: *no*

运行benchmark测试。 *Benchmarking* 。

自定义项目命令

您也可以通过 **COMMANDS_MODULE** 来添加您自己的项目命令。您可以以 scrapy/commands [https://github.com/scrapy/scrapy/blob/master/scrapy/commands] 中Scrapy commands为例来了解如何实现您的命令。

COMMANDS_MODULE

Default: `''` (empty string)

用于查找添加自定义Scrapy命令的模块。

例子:

```
COMMANDS_MODULE = 'mybot.commands'
```

讨论

Items

爬取的主要目标就是从非结构性的数据源提取结构性数据，例如网页。 Scrapy提供 `Item` 类来满足这样的需求。

`Item` 对象是种简单的容器，保存了爬取到得数据。 其提供了 类似于词典(dictionary-like) [http://docs.python.org/library/stdtypes.html#dict] 的API以及用于声明可用字段的简单语法。

声明Item

Item使用简单的class定义语法以及 `Field` 对象来声明。例如:

```
import scrapy

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    stock = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)
```

注解

熟悉 Django [http://www.djangoproject.com/] 的朋友一定会注意到Scrapy Item定义方式与 Django Models [http://docs.djangoproject.com/en/dev/topics/db/models/] 很类似, 不过没有那么多不同的字段类型(Field type)，更为简单。

Item字段(Item Fields)

`Field` 对象指明了每个字段的元数据(metadata)。例如下面例子中 `last_updated` 中指明了该字段的序列化函数。

您可以为每个字段指明任何类型的元数据。 `Field` 对象对接受的值没有任何限制。也正是因为这个原因，文档也无法提供所有可用的元数据的键(key)参考列表。 `Field` 对象中保存的每个键可以由多个组件使用，并且只有这些组件知道这个键的存在。您可以根据自己的需求，定义使用其他的 `Field` 键。 设置 `Field` 对象的主要目的就是在一个地方定义好所有的元数据。 一般来说，那些依赖某个字段的组件肯定使用了特定的键(key)。您必须查看组件相关的文档，查看其用了哪些元数据键(metadata key)。

需要注意的是，用来声明item的 `Field` 对象并没有被赋值为class的属性。 不过您可以通过 `Item.fields` 属性进行访问。

以上就是所有您需要知道的如何声明item的内容了。

与Item配合

接下来以 *下边声明* 的 `Product` item来演示一些item的操作。您会发现API和 dict API [http://docs.python.org/library/stdtypes.html#dict] 非常相似。

创建item

```
>>> product = Product(name='Desktop PC', price=1000)
>>> print product
Product(name='Desktop PC', price=1000)
```

获取字段的值

```
>>> product['name']
Desktop PC
>>> product.get('name')
Desktop PC

>>> product['price']
1000

>>> product['last_updated']
Traceback (most recent call last):
    ...
KeyError: 'last_updated'

>>> product.get('last_updated', 'not set')
not set

>>> product['lala'] # getting unknown field
Traceback (most recent call last):
    ...
KeyError: 'lala'

>>> product.get('lala', 'unknown field')
'unknown field'
```

```
>>> 'name' in product  # is name field populated?
True

>>> 'last_updated' in product  # is last_updated populated?
False

>>> 'last_updated' in product.fields  # is last_updated a declared field?
True

>>> 'lala' in product.fields  # is lala a declared field?
False
```

设置字段的值

```
>>> product['last_updated'] = 'today'
>>> product['last_updated']
today

>>> product['lala'] = 'test' # setting unknown field
Traceback (most recent call last):
    ...
KeyError: 'Product does not support field: lala'
```

获取所有获取到的值

您可以使用 dict API [http://docs.python.org/library/stdtypes.html#dict] 来获取所有的值:

```
>>> product.keys()
['price', 'name']

>>> product.items()
[('price', 1000), ('name', 'Desktop PC')]
```

其他任务

复制item:

```
>>> product2 = Product(product)
>>> print product2
Product(name='Desktop PC', price=1000)

>>> product3 = product2.copy()
>>> print product3
Product(name='Desktop PC', price=1000)
```

根据item创建字典(dict):

```
>>> dict(product) # create a dict from all populated values
{'price': 1000, 'name': 'Desktop PC'}
```

根据字典(dict)创建item:

```
>>> Product({'name': 'Laptop PC', 'price': 1500})
Product(price=1500, name='Laptop PC')

>>> Product({'name': 'Laptop PC', 'lala': 1500}) # warning: unknown field in dict
Traceback (most recent call last):
    ...
KeyError: 'Product does not support field: lala'
```

扩展Item

您可以通过继承原始的Item来扩展item(添加更多的字段或者修改某些字段的元数据)。

例如:

```
class DiscountedProduct(Product):
    discount_percent = scrapy.Field(serializer=str)
    discount_expiration_date = scrapy.Field()
```

您也可以通过使用原字段的元数据,添加新的值或修改原来的值来扩展字段的元数据:

```
class SpecificProduct(Product):
    name = scrapy.Field(Product.fields['name'], serializer=my_serializer)
```

这段代码在保留所有原来的元数据值的情况下添加(或者覆盖)了 `name` 字段的 `serializer` 。

Item对象

class `scrapy.item.`**`Item`**([*arg*])

返回一个根据给定的参数可选初始化的item。

Item复制了标准的 dict API [http://docs.python.org/library/stdtypes.html#dict] 。包括初始化函数也相同。Item唯一额外添加的属性是:

fields

一个包含了item所有声明的字段的字典，而不仅仅是获取到的字段。该字典的key是字段(field)的名字，值是 *Item声明* 中使用到的 **Field** 对象。

字段(Field)对象

class scrapy.item.**Field**([*arg*])

Field 仅仅是内置的 dict [http://docs.python.org/library/stdtypes.html#dict] 类的一个别名，并没有提供额外的方法或者属性。换句话说， **Field** 对象完完全全就是Python字典(dict)。被用来基于类属性(class attribute)的方法来支持 *item声明语法* 。

讨论

Spiders

Spider类定义了如何爬取某个(或某些)网站。包括了爬取的动作(例如:是否跟进链接)以及如何从网页的内容中提取结构化数据(爬取item)。 换句话说，Spider就是您定义爬取的动作及分析某个网页(或者是有些网页)的地方。

对spider来说，爬取的循环类似下文:

1. 以初始的URL初始化Request，并设置回调函数。 当该request下载完毕并返回时，将生成response，并作为参数传给该回调函数。

   spider中初始的request是通过调用 **start_requests()** 来获取的。 **start_requests()** 读取 **start_urls** 中的URL， 并以 **parse** 为回调函数生成 **Request** 。

2. 在回调函数内分析返回的(网页)内容，返回 **Item** 对象或者 **Request** 或者一个包括二者的可迭代容器。 返回的Request对象之后会经过Scrapy处理，下载相应的内容，并调用设置的callback函数(函数可相同)。

3. 在回调函数内，您可以使用 *选择器(Selectors)* (您也可以使用BeautifulSoup, lxml 或者您想用的任何解析器) 来分析网页内容，并根据分析的数据生成item。

4. 最后，由spider返回的item将被存到数据库(由某些 *Item Pipeline* 处理)或使用 *Feed exports* 存入到文件中。

虽然该循环对任何类型的spider都(多少)适用，但Scrapy仍然为了不同的需求提供了多种默认spider。 之后将讨论这些spider。

Spider参数

Spider可以通过接受参数来修改其功能。 spider参数一般用来定义初始URL或者指定限制爬取网站的部分。 您也可以使用其来配置spider的任何功能。

在运行 **crawl** 时添加 `-a` 可以传递Spider参数:

```
scrapy crawl myspider -a category=electronics
```

Spider在构造器(constructor)中获取参数:

```
import scrapy

class MySpider(Spider):
    name = 'myspider'

    def __init__(self, category=None, *args, **kwargs):
        super(MySpider, self).__init__(*args, **kwargs)
        self.start_urls = ['http://www.example.com/categories/%s' % category]
        # ...
```

Spider参数也可以通过Scrapyd的 `schedule.json` API来传递。 参见 Scrapyd documentation [http://scrapyd.readthedocs.org/].

内置Spider参考手册

Scrapy提供多种方便的通用spider供您继承使用。 这些spider为一些常用的爬取情况提供方便的特性， 例如根据某些规则跟进某个网站的所有链接、根据 Sitemaps [http://www.sitemaps.org] 来进行爬取，或者分析XML/CSV源。

下面spider的示例中，我们假定您有个项目在 `myproject.items` 模块中声明了 `TestItem`:

```
import scrapy

class TestItem(scrapy.Item):
    id = scrapy.Field()
    name = scrapy.Field()
    description = scrapy.Field()
```

Spider

class `scrapy.spider.`**Spider**

Spider是最简单的spider。每个其他的spider必须继承自该类(包括Scrapy自带的其他spider以及您自己编写的spider)。 Spider并没有提供什么特殊的功能。 其仅仅请求给定的 `start_urls`/`start_requests` ，并根据返回的结果(resulting responses)调用spider的 `parse` 方法。

name

定义spider名字的字符串(string)。spider的名字定义了Scrapy如何定位(并初始化)spider，所以其必须是唯一的。 不过您可以生成多个相同的spider实例(instance)，这没有任何限制。 name是spider最重要的属性，而且是必须的。

如果该spider爬取单个网站(single domain)，一个常见的做法是以该网站(domain)(加或不加 后缀 [http://en.wikipedia.org/wiki/Top-level_domain])来命名spider。 例如，如果spider爬取 `mywebsite.com` ，该spider通常会被命名为 `mywebsite` 。

allowed_domains

可选。包含了spider允许爬取的域名(domain)列表(list)。 当 `OffsiteMiddleware` 启用时， 域名不在列表中的URL不会被跟进。

start_urls

URL列表。当没有制定特定的URL时，spider将从该列表中开始进行爬取。 因此，第一个被获取到的页面的URL将是该列表之一。 后续的URL将会从获取到的数据中提取。

custom_settings

A dictionary of settings that will be overridden from the project wide configuration when running this spider. It must be defined as a class attribute since the settings are updated before instantiation.

For a list of available built-in settings see: *内置设定参考手册*.

crawler

This attribute is set by the `from_crawler()` class method after initializating the class, and links to the `Crawler` object to which this spider instance is bound.

Crawlers encapsulate a lot of components in the project for their single entry access (such as extensions, middlewares, signals managers, etc). See *Crawler API* to know more about them.

settings

Configuration on which this spider is been ran. This is a `Settings` instance, see the *Settings* topic for a detailed introduction on this subject.

from_crawler(*crawler*, **args*, ***kwargs*)

This is the class method used by Scrapy to create your spiders.

You probably won't need to override this directly, since the default implementation acts as a proxy to the `__init__()` method, calling it with the given arguments *args* and named arguments *kwargs*.

Nonetheless, this method sets the `crawler` and `settings` attributes in the new instance, so they can be accessed later inside the spider's code.

参数：
- **crawler** (`Crawler` instance) – crawler to which the spider will be bound
- **args** (*list*) – arguments passed to the `__init__()` method
- **kwargs** (*dict*) – keyword arguments passed to the `__init__()` method

start_requests()

该方法必须返回一个可迭代对象(iterable)。该对象包含了spider用于爬取的第一个Request。

当spider启动爬取并且未制定URL时，该方法被调用。 当指定了URL时，`make_requests_from_url()` 将被调用来创建Request对象。 该方法仅仅会被Scrapy调用一次，因此您可以将其实现为生成器。

该方法的默认实现是使用 `start_urls` 的url生成Request。

如果您想要修改最初爬取某个网站的Request对象，您可以重写(override)该方法。 例如，如果您需要在启动时以POST登录某个网站，你可以这么写:

```
def start_requests(self):
    return [scrapy.FormRequest("http://www.example.com/login",
                               formdata={'user': 'john', 'pass': 'secret'},
                               callback=self.logged_in)]

def logged_in(self, response):
    # here you would extract links to follow and return Requests for
    # each of them, with another callback
    pass
```

make_requests_from_url(*url*)

该方法接受一个URL并返回用于爬取的 `Request` 对象。 该方法在初始化request时被 `start_requests()` 调用，也被用于转化url为request。

默认未被复写(overridden)的情况下，该方法返回的Request对象中， `parse()` 作为回调函数，dont_filter参数也被设置为开启。 (详情参见 `Request`).

parse(*response*)

当response没有指定回调函数时，该方法是Scrapy处理下载的response的默认方法。

`parse` 负责处理response并返回处理的数据以及(/或)跟进的URL。 `Spider` 对其他的Request的回调函数也有相同的要求。

该方法及其他的Request回调函数必须返回一个包含 `Request` 及(或) `Item` 的可迭代的对象。

参数： **response** (`Response`) – 用于分析的response

log(*message*[, *level*, *component*])

使用 scrapy.log.msg() 方法记录(log)message。 log中自动带上该spider的 name 属性。 更多数据请参见 *Logging* 。

closed(*reason*)

当spider关闭时，该函数被调用。 该方法提供了一个替代调用signals.connect()来监听 spider_closed 信号的快捷方式。

Spider样例

让我们来看一个例子:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = [
        'http://www.example.com/1.html',
        'http://www.example.com/2.html',
        'http://www.example.com/3.html',
    ]

    def parse(self, response):
        self.log('A response from %s just arrived!' % response.url)
```

另一个在单个回调函数中返回多个Request以及Item的例子:

```
import scrapy
from myproject.items import MyItem

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = [
        'http://www.example.com/1.html',
        'http://www.example.com/2.html',
        'http://www.example.com/3.html',
    ]

    def parse(self, response):
        sel = scrapy.Selector(response)
        for h3 in response.xpath('//h3').extract():
            yield MyItem(title=h3)

        for url in response.xpath('//a/@href').extract():
            yield scrapy.Request(url, callback=self.parse)
```

CrawlSpider

class scrapy.contrib.spiders.**CrawlSpider**

爬取一般网站常用的spider。其定义了一些规则(rule)来提供跟进link的方便的机制。 也许该spider并不是完全适合您的特定网站或项目，但其对很多情况都使用。 因此您可以以其为起点，根据需求修改部分方法。当然您也可以实现自己的spider。

除了从Spider继承过来的(您必须提供的)属性外，其提供了一个新的属性:

rules

一个包含一个(或多个) Rule 对象的集合(list)。 每个 Rule 对爬取网站的动作定义了特定表现。 Rule对象在下边会介绍。 如果多个rule匹配了相同的链接，则根据他们在本属性中被定义的顺序，第一个会被使用。

该spider也提供了一个可复写(overrideable)的方法:

parse_start_url(*response*)

当start_url的请求返回时，该方法被调用。 该方法分析最初的返回值并必须返回一个 Item 对象或者 一个 Request 对象或者 一个可迭代的包含二者对象。

爬取规则(Crawling rules)

class scrapy.contrib.spiders.**Rule**(*link_extractor*, *callback=None*, *cb_kwargs=None*, *follow=None*, *process_links=None*, *process_request=None*)

link_extractor 是一个 *Link Extractor* 对象。 其定义了如何从爬取到的页面提取链接。

callback 是一个callable或string(该spider中同名的函数将会被调用)。 从link_extractor中每获取到链接时将会调用该函数。该回调函数接受一个response作为其第一个参数， 并返回一个包含 Item 以及(或) Request 对象(或者这两者的子类)的列表(list)。

警告

当编写爬虫规则时，请避免使用 parse 作为回调函数。 由于 CrawlSpider 使用 parse 方法来实现其逻辑，如果 您覆盖了 parse 方法，crawl spider 将会运行失败。

`cb_kwargs` 包含传递给回调函数的参数(keyword argument)的字典。

`follow` 是一个布尔(boolean)值，指定了根据该规则从response提取的链接是否需要跟进。 如果 `callback` 为None， `follow` 默认设置为 `True` ， 否则默认为 `False` 。

`process_links` 是一个callable或string(该spider中同名的函数将会被调用)。 从link_extractor中获取到链接列表时将会调用该函数。该方法主要用来过滤。

`process_request` 是一个callable或string(该spider中同名的函数将会被调用)。 该规则提取到每个request时都会调用该函数。该函数必须返回一个request或者None。 (用来过滤request)

CrawlSpider样例

接下来给出配合rule使用CrawlSpider的例子:

```
import scrapy
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors import LinkExtractor

class MySpider(CrawlSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com']

    rules = (
        # 提取匹配 'category.php' (但不匹配 'subsection.php') 的链接并跟进链接(没有callback意味着follow默认为True)
        Rule(LinkExtractor(allow=('category\.php', ), deny=('subsection\.php', ))),

        # 提取匹配 'item.php' 的链接并使用spider的parse_item方法进行分析
        Rule(LinkExtractor(allow=('item\.php', )), callback='parse_item'),
    )

    def parse_item(self, response):
        self.log('Hi, this is an item page! %s' % response.url)

        item = scrapy.Item()
        item['id'] = response.xpath('//td[@id="item_id"]/text()').re(r'ID: (\d+)')
        item['name'] = response.xpath('//td[@id="item_name"]/text()').extract()
        item['description'] = response.xpath('//td[@id="item_description"]/text()').extract()
        return item
```

该spider将从example.com的首页开始爬取，获取category以及item的链接并对后者使用 `parse_item` 方法。 当item获得返回(response)时，将使用XPath处理HTML并生成一些数据填入 **Item** 中。

XMLFeedSpider

class `scrapy.contrib.spiders.`**`XMLFeedSpider`**

XMLFeedSpider被设计用于通过迭代各个节点来分析XML源(XML feed)。 迭代器可以从 `iternodes` ， `xml` ， `html` 选择。 鉴于 `xml` 以及 `html` 迭代器需要先读取所有DOM再分析而引起的性能问题， 一般还是推荐使用 `iternodes` 。 不过使用 `html` 作为迭代器能有效应对错误的XML。

您必须定义下列类属性来设置迭代器以及标签名(tag name):

`iterator`

用于确定使用哪个迭代器的string。可选项有:

- `'iternodes'` - 一个高性能的基于正则表达式的迭代器
- `'html'` - 使用 **Selector** 的迭代器。 需要注意的是该迭代器使用DOM进行分析，其需要将所有的DOM载入内存，当数据量大的时候会产生问题。
- `'xml'` - 使用 **Selector** 的迭代器。 需要注意的是该迭代器使用DOM进行分析，其需要将所有的DOM载入内存，当数据量大的时候会产生问题。

默认值为 `iternodes` 。

`itertag`

一个包含开始迭代的节点名的string。例如:

```
itertag = 'product'
```

`namespaces`

一个由 `(prefix, url)` 元组(tuple)所组成的list。 其定义了在该文档中会被spider处理的可用的namespace。 `prefix` 及 `uri` 会被自动调用 **register_namespace()** 生成namespace。

您可以通过在 **itertag** 属性中制定节点的namespace。

例如:

```
class YourSpider(XMLFeedSpider):

    namespaces = [('n', 'http://www.sitemaps.org/schemas/sitemap/0.9')]
    itertag = 'n:url'
    # ...
```

除了这些新的属性之外，该spider也有以下可以覆盖(overrideable)的方法:

adapt_response(*response*)

该方法在spider分析response前被调用。您可以在response被分析之前使用该函数来修改内容(body)。 该方法接受一个response并返回一个response(可以相同也可以不同)。

parse_node(*response*, *selector*)

当节点符合提供的标签名时(`itertag`)该方法被调用。 接收到的response以及相应的 `Selector` 作为参数传递给该方法。 该方法返回一个 `Item` 对象或者 `Request` 对象 或者一个包含二者的可迭代对象(iterable)。

process_results(*response*, *results*)

当spider返回结果(item或request)时该方法被调用。 设定该方法的目的是在结果返回给框架核心(framework core)之前做最后的处理， 例如设定item的ID。其接受一个结果的列表(list of results)及对应的response。 其结果必须返回一个结果的列表(list of results)(包含Item或者Request对象)。

XMLFeedSpider例子

该spider十分易用。下边是其中一个例子:

```
from scrapy import log
from scrapy.contrib.spiders import XMLFeedSpider
from myproject.items import TestItem

class MySpider(XMLFeedSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = ['http://www.example.com/feed.xml']
    iterator = 'iternodes' # This is actually unnecessary, since it's the default value
    itertag = 'item'

    def parse_node(self, response, node):
        log.msg('Hi, this is a <%s> node!: %s' % (self.itertag, ''.join(node.extract())))

        item = TestItem()
        item['id'] = node.xpath('@id').extract()
        item['name'] = node.xpath('name').extract()
        item['description'] = node.xpath('description').extract()
        return item
```

简单来说，我们在这里创建了一个spider，从给定的 `start_urls` 中下载feed， 并迭代feed中每个 `item` 标签，输出，并在 `Item` 中存储有些随机数据。

CSVFeedSpider

class `scrapy.contrib.spiders.`**CSVFeedSpider**

该spider除了其按行遍历而不是节点之外其他和XMLFeedSpider十分类似。 而其在每次迭代时调用的是 `parse_row()` 。

delimiter

在CSV文件中用于区分字段的分隔符。类型为string。 默认为 `','` (逗号)。

quotechar

A string with the enclosure character for each field in the CSV file Defaults to `'"'` (quotation mark).

headers

在CSV文件中包含的用来提取字段的行的列表。参考下边的例子。

parse_row(*response*, *row*)

该方法接收一个response对象及一个以提供或检测出来的header为键的字典(代表每行)。 该spider中，您也可以覆盖 `adapt_response` 及 `process_results` 方法来进行预处理(pre-processing)及后(post-processing)处理。

CSVFeedSpider例子

下面的例子和之前的例子很像，但使用了 `CSVFeedSpider`:

```
from scrapy import log
from scrapy.contrib.spiders import CSVFeedSpider
from myproject.items import TestItem

class MySpider(CSVFeedSpider):
    name = 'example.com'
    allowed_domains = ['example.com']
```

```
    start_urls = ['http://www.example.com/feed.csv']
    delimiter = ';'
    quotechar = "'"
    headers = ['id', 'name', 'description']

    def parse_row(self, response, row):
        log.msg('Hi, this is a row!: %r' % row)

        item = TestItem()
        item['id'] = row['id']
        item['name'] = row['name']
        item['description'] = row['description']
        return item
```

SitemapSpider

class `scrapy.contrib.spiders.`**`SitemapSpider`**

SitemapSpider使您爬取网站时可以通过 Sitemaps [http://www.sitemaps.org] 来发现爬取的URL。

其支持嵌套的sitemap，并能从 robots.txt [http://www.robotstxt.org/] 中获取sitemap的url。

`sitemap_urls`

包含您要爬取的url的sitemap的url列表(list)。 您也可以指定为一个 robots.txt [http://www.robotstxt.org/] ，spider会从中分析并提取url。

`sitemap_rules`

一个包含 `(regex, callback)` 元组的列表(list):

- `regex` 是一个用于匹配从sitemap提供的url的正则表达式。 `regex` 可以是一个字符串或者编译的正则对象(compiled regex object)。
- callback指定了匹配正则表达式的url的处理函数。 `callback` 可以是一个字符串(spider中方法的名字)或者是callable。

例如:

```
sitemap_rules = [('/product/', 'parse_product')]
```

规则按顺序进行匹配，之后第一个匹配才会被应用。

如果您忽略该属性，sitemap中发现的所有url将会被 `parse` 函数处理。

`sitemap_follow`

一个用于匹配要跟进的sitemap的正则表达式的列表(list)。其仅仅被应用在 使用 *Sitemap index files* 来指向其他sitemap文件的站点。

默认情况下所有的sitemap都会被跟进。

`sitemap_alternate_links`

指定当一个 `url` 有可选的链接时，是否跟进。 有些非英文网站会在一个 `url` 块内提供其他语言的网站链接。

例如:

```
<url>
    <loc>http://example.com/</loc>
    <xhtml:link rel="alternate" hreflang="de" href="http://example.com/de"/>
</url>
```

当 `sitemap_alternate_links` 设置时，两个URL都会被获取。 当 `sitemap_alternate_links` 关闭时，只有 `http://example.com/` 会被获取。

默认 `sitemap_alternate_links` 关闭。

SitemapSpider样例

简单的例子: 使用 `parse` 处理通过sitemap发现的所有url:

```
from scrapy.contrib.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/sitemap.xml']

    def parse(self, response):
        pass # ... scrape item here ...
```

用特定的函数处理某些url，其他的使用另外的callback:

```
from scrapy.contrib.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/sitemap.xml']
    sitemap_rules = [
        ('/product/', 'parse product'),
```

```
        ('/category/', 'parse_category'),
    ]

    def parse_product(self, response):
        pass # ... scrape product ...

    def parse_category(self, response):
        pass # ... scrape category ...
```

跟进 robots.txt [http://www.robotstxt.org/] 文件定义的sitemap并只跟进包含有 ..sitemap_shop 的url:

```
from scrapy.contrib.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/robots.txt']
    sitemap_rules = [
        ('/shop/', 'parse_shop'),
    ]
    sitemap_follow = ['/sitemap_shops']

    def parse_shop(self, response):
        pass # ... scrape shop here ...
```

在SitemapSpider中使用其他url:

```
from scrapy.contrib.spiders import SitemapSpider

class MySpider(SitemapSpider):
    sitemap_urls = ['http://www.example.com/robots.txt']
    sitemap_rules = [
        ('/shop/', 'parse_shop'),
    ]

    other_urls = ['http://www.example.com/about']

    def start_requests(self):
        requests = list(super(MySpider, self).start_requests())
        requests += [scrapy.Request(x, self.parse_other) for x in self.other_urls]
        return requests

    def parse_shop(self, response):
        pass # ... scrape shop here ...

    def parse_other(self, response):
        pass # ... scrape other here ...
```

讨论

选择器(Selectors)

当抓取网页时，你做的最常见的任务是从HTML源码中提取数据。现有的一些库可以达到这个目的：

- BeautifulSoup [http://www.crummy.com/software/BeautifulSoup/] 是在程序员间非常流行的网页分析库，它基于HTML代码的结构来构造一个Python对象， 对不良标记的处理也非常合理，但它有一个缺点：慢。
- lxml [http://lxml.de/] 是一个基于 ElementTree [http://docs.python.org/library/xml.etree.elementtree.html] (不是Python标准库的一部分)的python化的XML解析库(也可以解析HTML)。

Scrapy提取数据有自己的一套机制。它们被称作选择器(seletors)，因为他们通过特定的 XPath [http://www.w3.org/TR/xpath] 或者 CSS [http://www.w3.org/TR/selectors] 表达式来“选择” HTML文件中的某个部分。

XPath [http://www.w3.org/TR/xpath] 是一门用来在XML文件中选择节点的语言，也可以用在HTML上。 CSS [http://www.w3.org/TR/selectors] 是一门将HTML文档样式化的语言。选择器由它定义，并与特定的HTML元素的样式相关连。

Scrapy选择器构建于 lxml [http://lxml.de/] 库之上，这意味着它们在速度和解析准确性上非常相似。

本页面解释了选择器如何工作，并描述了相应的API。不同于 lxml [http://lxml.de/] API的臃肿，该API短小而简洁。这是因为 lxml [http://lxml.de/] 库除了用来选择标记化文档外，还可以用到许多任务上。

选择器API的完全参考详见 *Selector reference*

使用选择器(selectors)

构造选择器(selectors)

Scrapy selector是以 文字(text) 或 `TextResponse` 构造的 `Selector` 实例。 其根据输入的类型自动选择最优的分析方法(XML vs HTML):

```
>>> from scrapy.selector import Selector
>>> from scrapy.http import HtmlResponse
```

以文字构造:

```
>>> body = '<html><body><span>good</span></body></html>'
>>> Selector(text=body).xpath('//span/text()').extract()
[u'good']
```

以response构造:

```
>>> response = HtmlResponse(url='http://example.com', body=body)
>>> Selector(response=response).xpath('//span/text()').extract()
[u'good']
```

为了方便起见，response对象以 *.selector* 属性提供了一个selector， 您可以随时使用该快捷方法:

```
>>> response.selector.xpath('//span/text()').extract()
[u'good']
```

使用选择器(selectors)

我们将使用 *Scrapy shell* (提供交互测试)和位于Scrapy文档服务器的一个样例页面，来解释如何使用选择器：

http://doc.scrapy.org/en/latest/_static/selectors-sample1.html

这里是它的HTML源码:

```
<html>
 <head>
  <base href='http://example.com/' />
  <title>Example website</title>
 </head>
 <body>
  <div id='images'>
   <a href='image1.html'>Name: My image 1 <br /><img src='image1_thumb.jpg' /></a>
   <a href='image2.html'>Name: My image 2 <br /><img src='image2_thumb.jpg' /></a>
   <a href='image3.html'>Name: My image 3 <br /><img src='image3_thumb.jpg' /></a>
   <a href='image4.html'>Name: My image 4 <br /><img src='image4_thumb.jpg' /></a>
   <a href='image5.html'>Name: My image 5 <br /><img src='image5_thumb.jpg' /></a>
  </div>
 </body>
</html>
```

首先, 我们打开shell:

```
scrapy shell http://doc.scrapy.org/en/latest/_static/selectors-sample1.html
```

接着，当shell载入后，您将获得名为 `response` 的shell变量，其为响应的response， 并且在其 `response.selector` 属性上绑定了一个selector。

因为我们处理的是HTML，选择器将自动使用HTML语法分析。

那么，通过查看 *HTML code* 该页面的源码，我们构建一个XPath来选择title标签内的文字:

```
>>> response.selector.xpath('//title/text()')
[<Selector (text) xpath=//title/text()>]
```

由于在response中使用XPath、CSS查询十分普遍，因此，Scrapy提供了两个实用的快捷方式: `response.xpath()` 及 `response.css()`:

```
>>> response.xpath('//title/text()')
[<Selector (text) xpath=//title/text()>]
>>> response.css('title::text')
[<Selector (text) xpath=//title/text()>]
```

如你所见， `.xpath()` 及 `.css()` 方法返回一个类 **SelectorList** 的实例, 它是一个新选择器的列表。这个API可以用来快速的提取嵌套数据。

为了提取真实的原文数据，你需要调用 `.extract()` 方法如下:

```
>>> response.xpath('//title/text()').extract()
[u'Example website']
```

注意CSS选择器可以使用CSS3伪元素(pseudo-elements)来选择文字或者属性节点:

```
>>> response.css('title::text').extract()
[u'Example website']
```

现在我们将得到根URL(base URL)和一些图片链接:

```
>>> response.xpath('//base/@href').extract()
[u'http://example.com/']

>>> response.css('base::attr(href)').extract()
[u'http://example.com/']

>>> response.xpath('//a[contains(@href, "image")]/@href').extract()
[u'image1.html',
 u'image2.html',
 u'image3.html',
 u'image4.html',
 u'image5.html']

>>> response.css('a[href*=image]::attr(href)').extract()
[u'image1.html',
 u'image2.html',
 u'image3.html',
 u'image4.html',
 u'image5.html']

>>> response.xpath('//a[contains(@href, "image")]/img/@src').extract()
[u'image1_thumb.jpg',
 u'image2_thumb.jpg',
 u'image3_thumb.jpg',
 u'image4_thumb.jpg',
 u'image5_thumb.jpg']

>>> response.css('a[href*=image] img::attr(src)').extract()
[u'image1_thumb.jpg',
 u'image2_thumb.jpg',
 u'image3_thumb.jpg',
 u'image4_thumb.jpg',
 u'image5_thumb.jpg']
```

嵌套选择器(selectors)

选择器方法(`.xpath()` or `.css()`)返回相同类型的选择器列表，因此你也可以对这些选择器调用选择器方法。下面是一个例子:

```
>>> links = response.xpath('//a[contains(@href, "image")]')
>>> links.extract()
[u'<a href="image1.html">Name: My image 1 <br><img src="image1_thumb.jpg"></a>',
 u'<a href="image2.html">Name: My image 2 <br><img src="image2_thumb.jpg"></a>',
 u'<a href="image3.html">Name: My image 3 <br><img src="image3_thumb.jpg"></a>',
 u'<a href="image4.html">Name: My image 4 <br><img src="image4_thumb.jpg"></a>',
 u'<a href="image5.html">Name: My image 5 <br><img src="image5_thumb.jpg"></a>']

>>> for index, link in enumerate(links):
        args = (index, link.xpath('@href').extract(), link.xpath('img/@src').extract())
        print 'Link number %d points to url %s and image %s' % args

Link number 0 points to url [u'image1.html'] and image [u'image1_thumb.jpg']
Link number 1 points to url [u'image2.html'] and image [u'image2_thumb.jpg']
Link number 2 points to url [u'image3.html'] and image [u'image3_thumb.jpg']
```

```
Link number 3 points to url [u'image4.html'] and image [u'image4_thumb.jpg']
Link number 4 points to url [u'image5.html'] and image [u'image5_thumb.jpg']
```

结合正则表达式使用选择器(selectors)

`Selector` 也有一个 `.re()` 方法，用来通过正则表达式来提取数据。然而，不同于使用 `.xpath()` 或者 `.css()` 方法, `.re()` 方法返回unicode字符串的列表。所以你无法构造嵌套式的 `.re()` 调用。

下面是一个例子，从上面的 *HTML code* 中提取图像名字:

```
>>> response.xpath('//a[contains(@href, "image")]/text()').re(r'Name:\s*(.*)')
[u'My image 1',
 u'My image 2',
 u'My image 3',
 u'My image 4',
 u'My image 5']
```

使用相对XPaths

记住如果你使用嵌套的选择器，并使用起始为 `/` 的XPath，那么该XPath将对文档使用绝对路径，而且对于你调用的 `Selector` 不是相对路径。

比如，假设你想提取在 `<div>` 元素中的所有 `<p>` 元素。首先，你将先得到所有的 `<div>` 元素:

```
>>> divs = response.xpath('//div')
```

开始时，你可能会尝试使用下面的错误的方法，因为它其实是从整篇文档中，而不仅仅是从那些 `<div>` 元素内部提取所有的 `<p>` 元素:

```
>>> for p in divs.xpath('//p'):  # this is wrong - gets all <p> from the whole document
...     print p.extract()
```

下面是比较合适的处理方法(注意 `.//p` XPath的点前缀):

```
>>> for p in divs.xpath('.//p'):  # extracts all <p> inside
...     print p.extract()
```

另一种常见的情况将是提取所有直系 `<p>` 的结果:

```
>>> for p in divs.xpath('p'):
...     print p.extract()
```

更多关于相对XPaths的细节详见XPath说明中的 Location Paths [http://www.w3.org/TR/xpath#location-paths] 部分。

使用EXSLT扩展

因建于 lxml [http://lxml.de/] 之上, Scrapy选择器也支持一些 EXSLT [http://www.exslt.org/] 扩展，可以在XPath表达式中使用这些预先制定的命名空间：

前缀	命名空间	用途
re	http://exslt.org/regular-expressions	正则表达式
set	http://exslt.org/sets	集合操作

正则表达式

例如在XPath的 `starts-with()` 或 `contains()` 无法满足需求时， `test()` 函数可以非常有用。

例如在列表中选择有”class”元素且结尾为一个数字的链接:

```
>>> from scrapy import Selector
>>> doc = """
... <div>
...     <ul>
...         <li class="item-0"><a href="link1.html">first item</a></li>
...         <li class="item-1"><a href="link2.html">second item</a></li>
...         <li class="item-inactive"><a href="link3.html">third item</a></li>
...         <li class="item-1"><a href="link4.html">fourth item</a></li>
...         <li class="item-0"><a href="link5.html">fifth item</a></li>
...     </ul>
... </div>
... """
>>> sel = Selector(text=doc, type="html")
>>> sel.xpath('//li//@href').extract()
[u'link1.html', u'link2.html', u'link3.html', u'link4.html', u'link5.html']
>>> sel.xpath('//li[re:test(@class, "item-\d$")]//@href').extract()
[u'link1.html', u'link2.html', u'link4.html', u'link5.html']
>>>
```

警告

C语言库 `libxslt` 不原生支持EXSLT正则表达式，因此 lxml [http://lxml.de/] 在实现时使用了Python `re` 模块的钩子。 因此，在XPath表达式

中使用regexp函数可能会牺牲少量的性能。

集合操作

集合操作可以方便地用于在提取文字元素前从文档树中去除一些部分。

例如使用itemscopes组和对应的itemprops来提取微数据(来自http://schema.org/Product的样本内容):

```
>>> doc = """
... <div itemscope itemtype="http://schema.org/Product">
...   <span itemprop="name">Kenmore White 17" Microwave</span>
...   <img src="kenmore-microwave-17in.jpg" alt='Kenmore 17" Microwave' />
...   <div itemprop="aggregateRating"
...     itemscope itemtype="http://schema.org/AggregateRating">
...    Rated <span itemprop="ratingValue">3.5</span>/5
...    based on <span itemprop="reviewCount">11</span> customer reviews
...   </div>
...
...   <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
...     <span itemprop="price">$55.00</span>
...     <link itemprop="availability" href="http://schema.org/InStock" />In stock
...   </div>
...
...   Product description:
...   <span itemprop="description">0.7 cubic feet countertop microwave.
...   Has six preset cooking categories and convenience features like
...   Add-A-Minute and Child Lock.</span>
...
...   Customer reviews:
...
...   <div itemprop="review" itemscope itemtype="http://schema.org/Review">
...     <span itemprop="name">Not a happy camper</span> -
...     by <span itemprop="author">Ellie</span>,
...     <meta itemprop="datePublished" content="2011-04-01">April 1, 2011
...     <div itemprop="reviewRating" itemscope itemtype="http://schema.org/Rating">
...       <meta itemprop="worstRating" content = "1">
...       <span itemprop="ratingValue">1</span>/
...       <span itemprop="bestRating">5</span>stars
...     </div>
...     <span itemprop="description">The lamp burned out and now I have to replace
...     it. </span>
...   </div>
...
...   <div itemprop="review" itemscope itemtype="http://schema.org/Review">
...     <span itemprop="name">Value purchase</span> -
...     by <span itemprop="author">Lucas</span>,
...     <meta itemprop="datePublished" content="2011-03-25">March 25, 2011
...     <div itemprop="reviewRating" itemscope itemtype="http://schema.org/Rating">
...       <meta itemprop="worstRating" content = "1"/>
...       <span itemprop="ratingValue">4</span>/
...       <span itemprop="bestRating">5</span>stars
...     </div>
...     <span itemprop="description">Great microwave for the price. It is small and
...     fits in my apartment.</span>
...   </div>
...   ...
... </div>
... """
>>>
>>> for scope in sel.xpath('//div[@itemscope]'):
...     print "current scope:", scope.xpath('@itemtype').extract()
...     props = scope.xpath('''
...                 set:difference(./descendant::*/@itemprop,
...                                .//*[@itemscope]/*/@itemprop)''')
...     print "    properties:", props.extract()
...     print
...

current scope: [u'http://schema.org/Product']
    properties: [u'name', u'aggregateRating', u'offers', u'description', u'review', u'review']

current scope: [u'http://schema.org/AggregateRating']
    properties: [u'ratingValue', u'reviewCount']

current scope: [u'http://schema.org/Offer']
    properties: [u'price', u'availability']

current scope: [u'http://schema.org/Review']
    properties: [u'name', u'author', u'datePublished', u'reviewRating', u'description']

current scope: [u'http://schema.org/Rating']
    properties: [u'worstRating', u'ratingValue', u'bestRating']

current scope: [u'http://schema.org/Review']
    properties: [u'name', u'author', u'datePublished', u'reviewRating', u'description']
```

```
current scope: [u'http://schema.org/Rating']
    properties: [u'worstRating', u'ratingValue', u'bestRating']

>>>
```

在这里，我们首先在 `itemscope` 元素上迭代，对于其中的每一个元素，我们寻找所有的 `itemprops` 元素，并排除那些本身在另一个 `itemscope` 内的元素。

Some XPath tips

Here are some tips that you may find useful when using XPath with Scrapy selectors, based on this post from ScrapingHub's blog [http://blog.scrapinghub.com/2014/07/17/xpath-tips-from-the-web-scraping-trenches/]. If you are not much familiar with XPath yet, you may want to take a look first at this XPath tutorial [http://www.zvon.org/comp/r/tut-XPath_1.html].

Using text nodes in a condition

When you need to use the text content as argument to a XPath string function [http://www.w3.org/TR/xpath/#section-String-Functions], avoid using `.//text()` and use just `.` instead.

This is because the expression `.//text()` yields a collection of text elements – a *node-set*. And when a node-set is converted to a string, which happens when it is passed as argument to a string function like `contains()` or `starts-with()`, it results in the text for the first element only.

Example:

```
>>> from scrapy import Selector
>>> sel = Selector(text='<a href="#">Click here to go to the <strong>Next Page</strong></a>')
```

Converting a *node-set* to string:

```
>>> sel.xpath('//a//text()').extract() # take a peek at the node-set
[u'Click here to go to the ', u'Next Page']
>>> sel.xpath("string(//a[1]//text())").extract() # convert it to string
[u'Click here to go to the ']
```

A *node* converted to a string, however, puts together the text of itself plus of all its descendants:

```
>>> sel.xpath("//a[1]").extract() # select the first node
[u'<a href="#">Click here to go to the <strong>Next Page</strong></a>']
>>> sel.xpath("string(//a[1])").extract() # convert it to string
[u'Click here to go to the Next Page']
```

So, using the `.//text()` node-set won't select anything in this case::

```
>>> sel.xpath("//a[contains(.//text(), 'Next Page')]").extract()
[]
```

But using the `.` to mean the node, works:

```
>>> sel.xpath("//a[contains(., 'Next Page')]").extract()
[u'<a href="#">Click here to go to the <strong>Next Page</strong></a>']
```

Beware the difference between //node[1] and (//node)[1]

`//node[1]` selects all the nodes occurring first under their respective parents.

`(//node)[1]` selects all the nodes in the document, and then gets only the first of them.

Example:

```
>>> from scrapy import Selector
>>> sel = Selector(text="""
....:     <ul class="list">
....:         <li>1</li>
....:         <li>2</li>
....:         <li>3</li>
....:     </ul>
....:     <ul class="list">
....:         <li>4</li>
....:         <li>5</li>
....:         <li>6</li>
....:     </ul>""")
>>> xp = lambda x: sel.xpath(x).extract()
```

This gets all first `<li>` elements under whatever it is its parent:

```
>>> xp("//li[1]")
[u'<li>1</li>', u'<li>4</li>']
```

And this gets the first `<li>` element in the whole document:

```
>>> xp("(//li)[1]")
[u'<li>1</li>']
```

This gets all first `<li>` elements under an `<ul>` parent:

```
>>> xp("//ul/li[1]")
[u'<li>1</li>', u'<li>4</li>']
```

And this gets the first `<li>` element under an `<ul>` parent in the whole document:

```
>>> xp("(//ul/li)[1]")
[u'<li>1</li>']
```

When querying by class, consider using CSS

Because an element can contain multiple CSS classes, the XPath way to select elements by class is the rather verbose:

```
*[contains(concat(' ', normalize-space(@class), ' '), ' someclass ')]
```

If you use `@class='someclass'` you may end up missing elements that have other classes, and if you just use `contains(@class, 'someclass')` to make up for that you may end up with more elements that you want, if they have a different class name that shares the string `someclass`.

As it turns out, Scrapy selectors allow you to chain selectors, so most of the time you can just select by class using CSS and then switch to XPath when needed:

```
>>> 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']
```

This is cleaner than using the verbose XPath trick shown above. Just remember to use the `.` in the XPath expressions that will follow.

内建选择器的参考

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

`Selector` 的实例是对选择某些内容响应的封装。

`response` 是 `HtmlResponse` 或 `XmlResponse` 的一个对象，将被用来选择和提取数据。

`text` 是在 `response` 不可用时的一个unicode字符串或utf-8编码的文字。将 `text` 和 `response` 一起使用是未定义行为。

`type` 定义了选择器类型，可以是 `"html"`, `"xml"` or `None` (默认).

如果 `type` 是 `None`，选择器会根据 `response` 类型(参见下面)自动选择最佳的类型，或者在和 `text` 一起使用时，默认为 `"html"`。

如果 `type` 是 `None`，并传递了一个 `response`，选择器类型将从response类型中推导如下：

- `"html"` for `HtmlResponse` type
- `"xml"` for `XmlResponse` type
- `"html"` for anything else

其他情况下，如果设定了 `type`，选择器类型将被强制设定，而不进行检测。

`xpath`(*query*)

寻找可以匹配xpath `query` 的节点，并返回 `SelectorList` 的一个实例结果，单一化其所有元素。列表元素也实现了 `Selector` 的接口。

`query` 是包含XPATH查询请求的字符串。

注解

为了方便起见，该方法也可以通过 `response.xpath()` 调用

`css`(*query*)

应用给定的CSS选择器，返回 `SelectorList` 的一个实例。

`query` 是一个包含CSS选择器的字符串。

在后台，通过 *cssselect* 库和运行 `.xpath()` 方法，CSS查询会被转换为XPath查询。

注解

为了方便起见，该方法也可以通过 response.css() 调用

extract()
串行化并将匹配到的节点返回一个unicode字符串列表。 结尾是编码内容的百分比。

re(*regex*)
应用给定的regex，并返回匹配到的unicode字符串列表。、

regex 可以是一个已编译的正则表达式，也可以是一个将被 re.compile(regex) 编译为正则表达式的字符串。

register_namespace(*prefix*, *uri*)
注册给定的命名空间，其将在 Selector 中使用。 不注册命名空间，你将无法从非标准命名空间中选择或提取数据。参见下面的例子。

remove_namespaces()
移除所有的命名空间，允许使用少量的命名空间xpaths遍历文档。参加下面的例子。

__nonzero__()
如果选择了任意的真实文档，将返回 True ，否则返回 False 。 也就是说， Selector 的布尔值是通过它选择的内容确定的。

SelectorList对象

class scrapy.selector.**SelectorList**

SelectorList 类是内建 list 类的子类，提供了一些额外的方法。

xpath(*query*)
对列表中的每个元素调用 .xpath() 方法，返回结果为另一个单一化的 SelectorList 。

query 和 Selector.xpath() 中的参数相同。

css(*query*)
对列表中的各个元素调用 .css() 方法，返回结果为另一个单一化的 SelectorList 。

query 和 Selector.css() 中的参数相同。

extract()
对列表中的各个元素调用 .extract() 方法，返回结果为单一化的unicode字符串列表。

re()
对列表中的各个元素调用 .re() 方法，返回结果为单一化的unicode字符串列表。

__nonzero__()
列表非空则返回True，否则返回False。

在HTML响应上的选择器样例

这里是一些 Selector 的样例，用来说明一些概念。 在所有的例子中，我们假设已经有一个通过 HtmlResponse 对象实例化的 Selector ，如下:

```
sel = Selector(html_response)
```

1. 从HTML响应主体中提取所有的 <h1> 元素，返回:class:*Selector* 对象(即 SelectorList 的一个对象)的列表:

   ```
   sel.xpath("//h1")
   ```

2. 从HTML响应主体上提取所有 <h1> 元素的文字，返回一个unicode字符串的列表:

   ```
   sel.xpath("//h1").extract()         # this includes the h1 tag
   sel.xpath("//h1/text()").extract()  # this excludes the h1 tag
   ```

3. 在所有 <p> 标签上迭代，打印它们的类属性:

   ```
   for node in sel.xpath("//p"):
       print node.xpath("@class").extract()
   ```

在XML响应上的选择器样例

这里是一些样例，用来说明一些概念。在两个例子中，我们假设已经有一个通过 XmlResponse 对象实例化的 Selector ，如下:

```
sel = Selector(xml_response)
```

1. 从XML响应主体中选择所有的 <product> 元素，返回 **Selector** 对象(即 **SelectorList** 对象)的列表:

```
sel.xpath("//product")
```

2. 从 Google Base XML feed [https://support.google.com/merchants/answer/160589?hl=en&ref_topic=2473799] 中提取所有的价钱，这需要注册一个命名空间:

```
sel.register_namespace("g", "http://base.google.com/ns/1.0")
sel.xpath("//g:price").extract()
```

移除命名空间

在处理爬虫项目时，完全去掉命名空间而仅仅处理元素名字，写更多简单/实用的XPath会方便很多。你可以为此使用 **Selector.remove_namespaces()** 方法。

让我们来看一个例子，以Github博客的atom订阅来解释这个情况。

首先，我们使用想爬取的url来打开shell:

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

一旦进入shell，我们可以尝试选择所有的 <link> 对象，可以看到没有结果(因为Atom XML命名空间混淆了这些节点):

```
>>> response.xpath("//link")
[]
```

但一旦我们调用 **Selector.remove_namespaces()** 方法，所有的节点都可以直接通过他们的名字来访问:

```
>>> response.selector.remove_namespaces()
>>> response.xpath("//link")
[<Selector xpath='//link' data=u'<link xmlns="http://www.w3.org/2005/Atom'>,
 <Selector xpath='//link' data=u'<link xmlns="http://www.w3.org/2005/Atom'>,
 ...
```

如果你对为什么命名空间移除操作并不总是被调用，而需要手动调用有疑惑。这是因为存在如下两个原因，按照相关顺序如下：

1. 移除命名空间需要迭代并修改文件的所有节点，而这对于Scrapy爬取的所有文档操作需要一定的性能消耗
2. 会存在这样的情况，确实需要使用命名空间，但有些元素的名字与命名空间冲突。尽管这些情况非常少见。

讨论

Item Loaders

Item Loaders提供了一种便捷的方式填充抓取到的 *Items* 。 虽然Items可以使用自带的类字典形式API填充，但是Items Loaders提供了更便捷的API， 可以分析原始数据并对Item进行赋值。

从另一方面来说， *Items* 提供保存抓取数据的 *容器* ， 而 Item Loaders提供的是 *填充* 容器的机制。

Item Loaders提供的是一种灵活，高效的机制，可以更方便的被spider或source format (HTML, XML, etc)扩展，并override更易于维护的、不同的内容分析规则。

Using Item Loaders to populate items

要使用Item Loader, 你必须先将它实例化. 你可以使用类似字典的对象(例如: Item or dict)来进行实例化, 或者不使用对象也可以, 当不用对象进行实例化的时候,Item会自动使用 `ItemLoader.default_item_class` 属性中指定的Item 类在Item Loader constructor中实例化.

然后,你开始收集数值到Item Loader时,通常使用 *Selectors*. 你可以在同一个item field 里面添加多个数值;Item Loader将知道如何用合适的处理函数来“添加”这些数值.

下面是在 *Spider* 中典型的Item Loader的用法, 使用 *Items chapter* 中声明的 *Product item*:

```
from scrapy.contrib.loader import ItemLoader
from myproject.items import Product

def parse(self, response):
    l = ItemLoader(item=Product(), response=response)
    l.add_xpath('name', '//div[@class="product_name"]')
    l.add_xpath('name', '//div[@class="product_title"]')
    l.add_xpath('price', '//p[@id="price"]')
    l.add_css('stock', 'p#stock]')
    l.add_value('last_updated', 'today') # you can also use literal values
    return l.load_item()
```

快速查看这些代码之后,我们可以看到发现 `name` 字段被从页面中两个不同的XPath位置提取:

1. `//div[@class="product_name"]`
2. `//div[@class="product_title"]`

换言之,数据通过用 `add_xpath()` 的方法,把从两个不同的XPath位置提取的数据收集起来. 这是将在以后分配给 `name` 字段中的数据。

之后,类似的请求被用于 `price` 和 `stock` 字段 (后者使用 CSS selector 和 `add_css()` 方法), 最后使用不同的方法 `add_value()` 对 `last_update` 填充文本值(`today`).

最终, 当所有数据被收集起来之后, 调用 `ItemLoader.load_item()` 方法, 实际上填充并且返回了之前通过调用 `add_xpath()`, `add_css()`, and `add_value()` 所提取和收集到的数据的Item.

Input and Output processors

Item Loader在每个(Item)字段中都包含了一个输入处理器和一个输出处理器。输入处理器收到数据时立刻提取数据 (通过 `add_xpath()`, `add_css()` 或者 `add_value()` 方法) 之后输入处理器的结果被收集起来并且保存在ItemLoader内. 收集到所有的数据后, 调用 `ItemLoader.load_item()` 方法来填充,并得到填充后的 `Item` 对象. 这是当输出处理器被和之前收集到的数据(和用输入处理器处理的)被调用. 输出处理器的结果是被分配到Item的最终值。

让我们看一个例子来说明如何输入和输出处理器被一个特定的字段调用(同样适用于其他field)::

```
l = ItemLoader(Product(), some_selector)
l.add_xpath('name', xpath1) # (1)
l.add_xpath('name', xpath2) # (2)
l.add_css('name', css) # (3)
l.add_value('name', 'test') # (4)
return l.load_item() # (5)
```

发生了这些事情:

1. 从 `xpath1` 提取出的数据,传递给 *输入处理器* 的 `name` 字段.输入处理器的结果被收集和保存在Item Loader中(但尚未分配给该Item)。
2. 从 `xpath2` 提取出来的数据,传递给(1)中使用的相同的 *输入处理器* .输入处理器的结果被附加到在(1)中收集的数据(如果有的话) 。
3. 和之前相似，只不过这里的数据是通过 `css` CSS selector抽取，之后传输到在(1)和(2)使用 的 *input processor* 中。最终输入处理器的结果被附加到在(1)和(2)中收集的数据之后 (如果存在数据的话)。
4. 这里的处理方式也和之前相似，但是此处的值是通过add_value直接赋予的， 而不是利用XPath表达式或CSS selector获取。得到的值仍然是被传送到输入处理器。 在这里例程中，因为得到的值并非可迭代，所以在传输到输入处理器之前需要将其 转化为可迭代的单个元素，这才是它所接受的形式。
5. 在之前步骤中所收集到的数据被传送到 *output processor* 的 `name` field中。 输出处理器的结果就是赋到item中 `name` field的值。

需要注意的是，输入和输出处理器都是可调用对象，调用时传入需要被分析的数据， 处理后返回分析得到的值。因此你可以使用任意函数

作为输入、输出处理器。 唯一需注意的是它们必须接收一个（并且只是一个）迭代器性质的positional参数。

注解

Both input and output processors must receive an iterator as their first argument. The output of those functions can be anything. The result of input processors will be appended to an internal list (in the Loader) containing the collected values (for that field). The result of the output processors is the value that will be finally assigned to the item.

The other thing you need to keep in mind is that the values returned by input processors are collected internally (in lists) and then passed to output processors to populate the fields.

Last, but not least, Scrapy comes with some *commonly used processors* built-in for convenience.

Declaring Item Loaders

Item Loaders are declared like Items, by using a class definition syntax. Here is an example:

```
from scrapy.contrib.loader import ItemLoader
from scrapy.contrib.loader.processor import TakeFirst, MapCompose, Join

class ProductLoader(ItemLoader):

    default_output_processor = TakeFirst()

    name_in = MapCompose(unicode.title)
    name_out = Join()

    price_in = MapCompose(unicode.strip)

    # ...
```

As you can see, input processors are declared using the `_in` suffix while output processors are declared using the `_out` suffix. And you can also declare a default input/output processors using the `ItemLoader.default_input_processor` and `ItemLoader.default_output_processor` attributes.

Declaring Input and Output Processors

As seen in the previous section, input and output processors can be declared in the Item Loader definition, and it's very common to declare input processors this way. However, there is one more place where you can specify the input and output processors to use: in the *Item Field* metadata. Here is an example:

```
import scrapy
from scrapy.contrib.loader.processor import Join, MapCompose, TakeFirst
from w3lib.html import remove_tags

def filter_price(value):
    if value.isdigit():
        return value

class Product(scrapy.Item):
    name = scrapy.Field(
        input_processor=MapCompose(remove_tags),
        output_processor=Join(),
    )
    price = scrapy.Field(
        input_processor=MapCompose(remove_tags, filter_price),
        output_processor=TakeFirst(),
    )
```

```
>>> from scrapy.contrib.loader import ItemLoader
>>> il = ItemLoader(item=Product())
>>> il.add_value('name', [u'Welcome to my', u'<strong>website</strong>'])
>>> il.add_value('price', [u'&euro;', u'<span>1000</span>'])
>>> il.load_item()
{'name': u'Welcome to my website', 'price': u'1000'}
```

The precedence order, for both input and output processors, is as follows:

1. Item Loader field-specific attributes: `field_in` and `field_out` (most precedence)
2. Field metadata (`input_processor` and `output_processor` key)
3. Item Loader defaults: `ItemLoader.default_input_processor()` and `ItemLoader.default_output_processor()` (least precedence)

See also: *Reusing and extending Item Loaders*.

Item Loader Context

The Item Loader Context is a dict of arbitrary key/values which is shared among all input and output processors in the Item Loader. It can be passed when declaring, instantiating or using Item Loader. They are used to modify the behaviour of the input/output processors.

For example, suppose you have a function `parse_length` which receives a text value and extracts a length from it:

```
def parse_length(text, loader_context):
    unit = loader_context.get('unit', 'm')
    # ... length parsing code goes here ...
    return parsed_length
```

By accepting a `loader_context` argument the function is explicitly telling the Item Loader that it's able to receive an Item Loader context, so the Item Loader passes the currently active context when calling it, and the processor function (`parse_length` in this case) can thus use them.

There are several ways to modify Item Loader context values:

1. By modifying the currently active Item Loader context (`context` attribute):

   ```
   loader = ItemLoader(product)
   loader.context['unit'] = 'cm'
   ```

2. On Item Loader instantiation (the keyword arguments of Item Loader constructor are stored in the Item Loader context):

   ```
   loader = ItemLoader(product, unit='cm')
   ```

3. On Item Loader declaration, for those input/output processors that support instantiating them with an Item Loader context. `MapCompose` is one of them:

   ```
   class ProductLoader(ItemLoader):
       length_out = MapCompose(parse_length, unit='cm')
   ```

ItemLoader objects

class `scrapy.contrib.loader.`**`ItemLoader`**([*item*, *selector*, *response*,]***kwargs*)

Return a new Item Loader for populating the given Item. If no item is given, one is instantiated automatically using the class in `default_item_class`.

When instantiated with a *selector* or a *response* parameters the `ItemLoader` class provides convenient mechanisms for extracting data from web pages using *selectors*.

参数:
- **item** (`Item` object) – The item instance to populate using subsequent calls to `add_xpath()`, `add_css()`, or `add_value()`.
- **selector** (`Selector` object) – The selector to extract data from, when using the `add_xpath()` (resp. `add_css()`) or `replace_xpath()` (resp. `replace_css()`) method.
- **response** (`Response` object) – The response used to construct the selector using the `default_selector_class`, unless the selector argument is given, in which case this argument is ignored.

The item, selector, response and the remaining keyword arguments are assigned to the Loader context (accessible through the `context` attribute).

`ItemLoader` instances have the following methods:

`get_value`(*value*, **processors*, ***kwargs*)

Process the given `value` by the given `processors` and keyword arguments.

Available keyword arguments:

参数:
- **re** (*str or compiled regex*) – a regular expression to use for extracting data from the given value using **`extract_regex()`** method, applied before processors

Examples:

```
>>> from scrapy.contrib.loader.processor import TakeFirst
>>> loader.get_value(u'name: foo', TakeFirst(), unicode.upper, re='name: (.+)')
'FOO`
```

`add_value`(*field_name*, *value*, **processors*, ***kwargs*)

Process and then add the given `value` for the given field.

The value is first passed through `get_value()` by giving the `processors` and `kwargs`, and then passed through the *field input processor* and its result appended to the data collected for that field. If the field already contains collected data, the new data is added.

The given `field_name` can be `None`, in which case values for multiple fields may be added. And the processed value should be a dict with field_name mapped to values.

Examples:

```
loader.add_value('name', u'Color TV')
loader.add_value('colours', [u'white', u'blue'])
loader.add_value('length', u'100')
loader.add_value('name', u'name: foo', TakeFirst(), re='name: (.+)')
```

```
loader.add_value(None, {'name': u'foo', 'sex': u'male'})
```

replace_value(*field_name*, *value*, **processors*, ***kwargs*)

Similar to `add_value()` but replaces the collected data with the new value instead of adding it.

get_xpath(*xpath*, **processors*, ***kwargs*)

Similar to `ItemLoader.get_value()` but receives an XPath instead of a value, which is used to extract a list of unicode strings from the selector associated with this `ItemLoader`.

参数：
- **xpath** (*str*) – the XPath to extract data from
- **re** (*str or compiled regex*) – a regular expression to use for extracting data from the selected XPath region

Examples:

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.get_xpath('//p[@class="product-name"]')
# HTML snippet: <p id="price">the price is $1200</p>
loader.get_xpath('//p[@id="price"]', TakeFirst(), re='the price is (.*)')
```

add_xpath(*field_name*, *xpath*, **processors*, ***kwargs*)

Similar to `ItemLoader.add_value()` but receives an XPath instead of a value, which is used to extract a list of unicode strings from the selector associated with this `ItemLoader`.

See `get_xpath()` for `kwargs`.

参数： **xpath** (*str*) – the XPath to extract data from

Examples:

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.add_xpath('name', '//p[@class="product-name"]')
# HTML snippet: <p id="price">the price is $1200</p>
loader.add_xpath('price', '//p[@id="price"]', re='the price is (.*)')
```

replace_xpath(*field_name*, *xpath*, **processors*, ***kwargs*)

Similar to `add_xpath()` but replaces collected data instead of adding it.

get_css(*css*, **processors*, ***kwargs*)

Similar to `ItemLoader.get_value()` but receives a CSS selector instead of a value, which is used to extract a list of unicode strings from the selector associated with this `ItemLoader`.

参数：
- **css** (*str*) – the CSS selector to extract data from
- **re** (*str or compiled regex*) – a regular expression to use for extracting data from the selected CSS region

Examples:

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.get_css('p.product-name')
# HTML snippet: <p id="price">the price is $1200</p>
loader.get_css('p#price', TakeFirst(), re='the price is (.*)')
```

add_css(*field_name*, *css*, **processors*, ***kwargs*)

Similar to `ItemLoader.add_value()` but receives a CSS selector instead of a value, which is used to extract a list of unicode strings from the selector associated with this `ItemLoader`.

See `get_css()` for `kwargs`.

参数： **css** (*str*) – the CSS selector to extract data from

Examples:

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.add_css('name', 'p.product-name')
# HTML snippet: <p id="price">the price is $1200</p>
loader.add_css('price', 'p#price', re='the price is (.*)')
```

replace_css(*field_name*, *css*, **processors*, ***kwargs*)

Similar to `add_css()` but replaces collected data instead of adding it.

load_item()

Populate the item with the data collected so far, and return it. The data collected is first passed through the *output processors* to get the final value to assign to each item field.

get_collected_values(*field_name*)

Return the collected values for the given field.

get_output_value(*field_name*)
Return the collected values parsed using the output processor, for the given field. This method doesn't populate or modify the item at all.

get_input_processor(*field_name*)
Return the input processor for the given field.

get_output_processor(*field_name*)
Return the output processor for the given field.

ItemLoader instances have the following attributes:

item
The Item object being parsed by this Item Loader.

context
The currently active *Context* of this Item Loader.

default_item_class
An Item class (or factory), used to instantiate items when not given in the constructor.

default_input_processor
The default input processor to use for those fields which don't specify one.

default_output_processor
The default output processor to use for those fields which don't specify one.

default_selector_class
The class used to construct the selector of this ItemLoader, if only a response is given in the constructor. If a selector is given in the constructor this attribute is ignored. This attribute is sometimes overridden in subclasses.

selector
The Selector object to extract data from. It's either the selector given in the constructor or one created from the response given in the constructor using the default_selector_class. This attribute is meant to be read-only.

Reusing and extending Item Loaders

As your project grows bigger and acquires more and more spiders, maintenance becomes a fundamental problem, especially when you have to deal with many different parsing rules for each spider, having a lot of exceptions, but also wanting to reuse the common processors.

Item Loaders are designed to ease the maintenance burden of parsing rules, without losing flexibility and, at the same time, providing a convenient mechanism for extending and overriding them. For this reason Item Loaders support traditional Python class inheritance for dealing with differences of specific spiders (or groups of spiders).

Suppose, for example, that some particular site encloses their product names in three dashes (e.g. `---Plasma TV---`) and you don't want to end up scraping those dashes in the final product names.

Here's how you can remove those dashes by reusing and extending the default Product Item Loader (`ProductLoader`):

```
from scrapy.contrib.loader.processor import MapCompose
from myproject.ItemLoaders import ProductLoader

def strip_dashes(x):
    return x.strip('-')

class SiteSpecificLoader(ProductLoader):
    name_in = MapCompose(strip_dashes, ProductLoader.name_in)
```

Another case where extending Item Loaders can be very helpful is when you have multiple source formats, for example XML and HTML. In the XML version you may want to remove `CDATA` occurrences. Here's an example of how to do it:

```
from scrapy.contrib.loader.processor import MapCompose
from myproject.ItemLoaders import ProductLoader
from myproject.utils.xml import remove_cdata

class XmlProductLoader(ProductLoader):
    name_in = MapCompose(remove_cdata, ProductLoader.name_in)
```

And that's how you typically extend input processors.

As for output processors, it is more common to declare them in the field metadata, as they usually depend only on the field and not on each specific site parsing rule (as input processors do). See also: *Declaring Input and Output Processors*.

There are many other possible ways to extend, inherit and override your Item Loaders, and different Item Loaders hierarchies may fit better for different projects. Scrapy only provides the mechanism; it doesn't impose any specific organization of your Loaders collection - that's up

to you and your project's needs.

Available built-in processors

Even though you can use any callable function as input and output processors, Scrapy provides some commonly used processors, which are described below. Some of them, like the `MapCompose` (which is typically used as input processor) compose the output of several functions executed in order, to produce the final parsed value.

Here is a list of all built-in processors:

class `scrapy.contrib.loader.processor.`**`Identity`**

The simplest processor, which doesn't do anything. It returns the original values unchanged. It doesn't receive any constructor arguments nor accepts Loader contexts.

Example:

```
>>> from scrapy.contrib.loader.processor import Identity
>>> proc = Identity()
>>> proc(['one', 'two', 'three'])
['one', 'two', 'three']
```

class `scrapy.contrib.loader.processor.`**`TakeFirst`**

Returns the first non-null/non-empty value from the values received, so it's typically used as an output processor to single-valued fields. It doesn't receive any constructor arguments, nor accept Loader contexts.

Example:

```
>>> from scrapy.contrib.loader.processor import TakeFirst
>>> proc = TakeFirst()
>>> proc(['', 'one', 'two', 'three'])
'one'
```

class `scrapy.contrib.loader.processor.`**`Join`**(*separator=u' '*)

Returns the values joined with the separator given in the constructor, which defaults to `u' '`. It doesn't accept Loader contexts.

When using the default separator, this processor is equivalent to the function: `u' '.join`

Examples:

```
>>> from scrapy.contrib.loader.processor import Join
>>> proc = Join()
>>> proc(['one', 'two', 'three'])
u'one two three'
>>> proc = Join('<br>')
>>> proc(['one', 'two', 'three'])
u'one<br>two<br>three'
```

class `scrapy.contrib.loader.processor.`**`Compose`**(**functions*, ***default_loader_context*)

A processor which is constructed from the composition of the given functions. This means that each input value of this processor is passed to the first function, and the result of that function is passed to the second function, and so on, until the last function returns the output value of this processor.

By default, stop process on `None` value. This behaviour can be changed by passing keyword argument `stop_on_none=False`.

Example:

```
>>> from scrapy.contrib.loader.processor import Compose
>>> proc = Compose(lambda v: v[0], str.upper)
>>> proc(['hello', 'world'])
'HELLO'
```

Each function can optionally receive a `loader_context` parameter. For those which do, this processor will pass the currently active *Loader context* through that parameter.

The keyword arguments passed in the constructor are used as the default Loader context values passed to each function call. However, the final Loader context values passed to functions are overridden with the currently active Loader context accessible through the **`ItemLoader.context()`** attribute.

class `scrapy.contrib.loader.processor.`**`MapCompose`**(**functions*, ***default_loader_context*)

A processor which is constructed from the composition of the given functions, similar to the `Compose` processor. The difference with this processor is the way internal results are passed among functions, which is as follows:

The input value of this processor is *iterated* and the first function is applied to each element. The results of these function calls (one for each element) are concatenated to construct a new iterable, which is then used to apply the second function, and so on, until the last function is applied to each value of the list of values collected so far. The output values of the last function are concatenated together to produce the output of this processor.

Each particular function can return a value or a list of values, which is flattened with the list of values returned by the same function applied to the other input values. The functions can also return `None` in which case the output of that function is ignored for further processing over the chain.

This processor provides a convenient way to compose functions that only work with single values (instead of iterables). For this reason the **MapCompose** processor is typically used as input processor, since data is often extracted using the **extract()** method of *selectors*, which returns a list of unicode strings.

The example below should clarify how it works:

```
>>> def filter_world(x):
...     return None if x == 'world' else x
...
>>> from scrapy.contrib.loader.processor import MapCompose
>>> proc = MapCompose(filter_world, unicode.upper)
>>> proc([u'hello', u'world', u'this', u'is', u'scrapy'])
[u'HELLO, u'THIS', u'IS', u'SCRAPY']
```

As with the Compose processor, functions can receive Loader contexts, and constructor keyword arguments are used as default context values. See **Compose** processor for more info.

讨论

Scrapy终端(Scrapy shell)

Scrapy终端是一个交互终端，供您在未启动spider的情况下尝试及调试您的爬取代码。 其本意是用来测试提取数据的代码，不过您可以将其作为正常的Python终端，在上面测试任何的Python代码。

该终端是用来测试XPath或CSS表达式，查看他们的工作方式及从爬取的网页中提取的数据。 在编写您的spider时，该终端提供了交互性测试您的表达式代码的功能，免去了每次修改后运行spider的麻烦。

一旦熟悉了Scrapy终端后，您会发现其在开发和调试spider时发挥的巨大作用。

如果您安装了 IPython [http://ipython.org/] ，Scrapy终端将使用 IPython [http://ipython.org/] (替代标准Python终端)。 IPython [http://ipython.org/] 终端与其他相比更为强大，提供智能的自动补全，高亮输出，及其他特性。

我们强烈推荐您安装 IPython [http://ipython.org/] ，特别是如果您使用Unix系统(IPython [http://ipython.org/] 在Unix下工作的很好)。 详情请参考 IPython installation guide [http://ipython.org/install.html] 。

启动终端

您可以使用 `shell` 来启动Scrapy终端:

```
scrapy shell <url>
```

`<url>` 是您要爬取的网页的地址。

使用终端

Scrapy终端仅仅是一个普通的Python终端(或 IPython [http://ipython.org/])。其提供了一些额外的快捷方式。

可用的快捷命令(shortcut)

- `shelp()` - 打印可用对象及快捷命令的帮助列表
- `fetch(request_or_url)` - 根据给定的请求(request)或URL获取一个新的response，并更新相关的对象
- `view(response)` - 在本机的浏览器打开给定的response。 其会在response的body中添加一个 <base> tag [https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base] ，使得外部链接(例如图片及css)能正确显示。 注意，该操作会在本地创建一个临时文件，且该文件不会被自动删除。

可用的Scrapy对象

Scrapy终端根据下载的页面会自动创建一些方便使用的对象，例如 `Response` 对象及 `Selector` 对象(对HTML及XML内容)。

这些对象有:

- `crawler` - 当前 `Crawler` 对象.
- `spider` - 处理URL的spider。 对当前URL没有处理的Spider时则为一个 `Spider` 对象。
- `request` - 最近获取到的页面的 `Request` 对象。 您可以使用 `replace()` 修改该request。或者 使用 `fetch` 快捷方式来获取新的request。
- `response` - 包含最近获取到的页面的 `Response` 对象。
- `sel` - 根据最近获取到的response构建的 `Selector` 对象。
- `settings` - 当前的 *Scrapy settings*

终端会话(shell session)样例

下面给出一个典型的终端会话的例子。 在该例子中，我们首先爬取了 http://scarpy.org 的页面，而后接着爬取 http://slashdot.org 的页面。最后，我们修改了(Slashdot)的请求，将请求设置为POST并重新获取， 得到HTTP 405(不允许的方法)错误。 之后通过Ctrl-D(Unix)或Ctrl-Z(Windows)关闭会话。

需要注意的是，由于爬取的页面不是静态页，内容会随着时间而修改， 因此例子中提取到的数据可能与您尝试的结果不同。 该例子的唯一目的是让您熟悉Scrapy终端。

首先，我们启动终端:

```
scrapy shell 'http://scrapy.org' --nolog
```

接着该终端(使用Scrapy下载器(downloader))获取URL内容并打印可用的对象及快捷命令(注意到以 `[s]` 开头的行):

```
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x1e16b50>
[s]   item       {}
[s]   request    <GET http://scrapy.org>
[s]   response   <200 http://scrapy.org>
[s]   sel        <Selector xpath=None data=u'<html>\n  <head>\n    <meta charset="utf-8'>
```

```
[s]   settings   <scrapy.settings.Settings object at 0x2bfd650>
[s]   spider     <Spider 'default' at 0x20c6f50>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

>>>
```

之后，您就可以操作这些对象了:

```
>>> sel.xpath("//h2/text()").extract()[0]
u'Welcome to Scrapy'

>>> fetch("http://slashdot.org")
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x1a13b50>
[s]   item       {}
[s]   request    <GET http://slashdot.org>
[s]   response   <200 http://slashdot.org>
[s]   sel        <Selector xpath=None data=u'<html lang="en">\n<head>\n\n\n\n\n<script id="'>
[s]   settings   <scrapy.settings.Settings object at 0x2bfd650>
[s]   spider     <Spider 'default' at 0x20c6f50>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser

>>> sel.xpath('//title/text()').extract()
[u'Slashdot: News for nerds, stuff that matters']

>>> request = request.replace(method="POST")

>>> fetch(request)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x1e16b50>
...

>>>
```

在spider中启动shell来查看response

有时您想在spider的某个位置中查看被处理的response， 以确认您期望的response到达特定位置。

这可以通过 `scrapy.shell.inspect_response` 函数来实现。

以下是如何在spider中调用该函数的例子:

```
import scrapy

class MySpider(scrapy.Spider):
    name = "myspider"
    start_urls = [
        "http://example.com",
        "http://example.org",
        "http://example.net",
    ]

    def parse(self, response):
        # We want to inspect one specific response.
        if ".org" in response.url:
            from scrapy.shell import inspect_response
            inspect_response(response, self)

        # Rest of parsing code.
```

当运行spider时，您将得到类似下列的输出:

```
2014-01-23 17:48:31-0400 [myspider] DEBUG: Crawled (200) <GET http://example.com> (referer: None)
2014-01-23 17:48:31-0400 [myspider] DEBUG: Crawled (200) <GET http://example.org> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x1e16b50>
...

>>> response.url
'http://example.org'
```

接着测试提取代码:

```
>>> sel.xpath('//h1[@class="fn"]')
[]
```

呃，看来是没有。您可以在浏览器里查看response的结果，判断是否是您期望的结果:

```
>>> view(response)
```

```
True
```

最后您可以点击Ctrl-D(Windows下Ctrl-Z)来退出终端，恢复爬取:

```
>>> ^D
2014-01-23 17:50:03-0400 [myspider] DEBUG: Crawled (200) <GET http://example.net> (referer: None)
...
```

注意: 由于该终端屏蔽了Scrapy引擎，您在这个终端中不能使用 `fetch` 快捷命令(shortcut)。 当您离开终端时，spider会从其停下的地方恢复爬取，正如上面显示的那样。

讨论

Item Pipeline

当Item在Spider中被收集之后，它将会被传递到Item Pipeline，一些组件会按照一定的顺序执行对Item的处理。

每个item pipeline组件(有时称之为“Item Pipeline”)是实现了简单方法的Python类。他们接收到Item并通过它执行一些行为，同时也决定此Item是否继续通过pipeline，或是被丢弃而不再进行处理。

以下是item pipeline的一些典型应用：

- 清理HTML数据
- 验证爬取的数据(检查item包含某些字段)
- 查重(并丢弃)
- 将爬取结果保存到数据库中

编写你自己的item pipeline

编写你自己的item pipeline很简单，每个item pipiline组件是一个独立的Python类，同时必须实现以下方法:

process_item(*self*, *item*, *spider*)

每个item pipeline组件都需要调用该方法，这个方法必须返回一个 `Item` (或任何继承类)对象， 或是抛出 `DropItem` 异常，被丢弃的item将不会被之后的pipeline组件所处理。

参数： **item** (`Item` 对象) – 被爬取的item
spider (`Spider` 对象) – 爬取该item的spider

此外,他们也可以实现以下方法:

open_spider(*self*, *spider*)

当spider被开启时，这个方法被调用。

参数： **spider** (`Spider` 对象) – 被开启的spider

close_spider(*spider*)

当spider被关闭时，这个方法被调用

参数： **spider** (`Spider` 对象) – 被关闭的spider

from_crawler(*cls*, *crawler*)

If present, this classmethod is called to create a pipeline instance from a `Crawler`. It must return a new instance of the pipeline. Crawler object provides access to all Scrapy core components like settings and signals; it is a way for pipeline to access them and hook its functionality into Scrapy.

参数： **crawler** (`Crawler` object) – crawler that uses this pipeline

Item pipeline 样例

验证价格，同时丢弃没有价格的item

让我们来看一下以下这个假设的pipeline，它为那些不含税(`price_excludes_vat` 属性)的item调整了 `price` 属性，同时丢弃了那些没有价格的item:

```
from scrapy.exceptions import DropItem

class PricePipeline(object):

    vat_factor = 1.15

    def process_item(self, item, spider):
        if item['price']:
            if item['price_excludes_vat']:
                item['price'] = item['price'] * self.vat_factor
            return item
        else:
            raise DropItem("Missing price in %s" % item)
```

将item写入JSON文件

以下pipeline将所有(从所有spider中)爬取到的item，存储到一个独立地 `items.jl` 文件，每行包含一个序列化为JSON格式的item:

```
import json
```

```
class JsonWriterPipeline(object):

    def __init__(self):
        self.file = open('items.jl', 'wb')

    def process_item(self, item, spider):
        line = json.dumps(dict(item)) + "\n"
        self.file.write(line)
        return item
```

注解

JsonWriterPipeline的目的只是为了介绍怎样编写item pipeline，如果你想要将所有爬取的item都保存到同一个JSON文件， 你需要使用 *Feed exports* 。

Write items to MongoDB

In this example we'll write items to MongoDB [http://www.mongodb.org/] using pymongo [http://api.mongodb.org/python/current/]. MongoDB address and database name are specified in Scrapy settings; MongoDB collection is named after item class.

The main point of this example is to show how to use `from_crawler()` method and how to clean up the resources properly.

注解

Previous example (JsonWriterPipeline) doesn't clean up resources properly. Fixing it is left as an exercise for the reader.

```
import pymongo

class MongoPipeline(object):

    def __init__(self, mongo_uri, mongo_db):
        self.mongo_uri = mongo_uri
        self.mongo_db = mongo_db

    @classmethod
    def from_crawler(cls, crawler):
        return cls(
            mongo_uri=crawler.settings.get('MONGO_URI'),
            mongo_db=crawler.settings.get('MONGO_DATABASE', 'items')
        )

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo_uri)
        self.db = self.client[self.mongo_db]

    def close_spider(self, spider):
        self.client.close()

    def process_item(self, item, spider):
        collection_name = item.__class__.__name__
        self.db[collection_name].insert(dict(item))
        return item
```

去重

一个用于去重的过滤器，丢弃那些已经被处理过的item。让我们假设我们的item有一个唯一的id，但是我们spider返回的多个item中包含有相同的id:

```
from scrapy.exceptions import DropItem

class DuplicatesPipeline(object):

    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        if item['id'] in self.ids_seen:
            raise DropItem("Duplicate item found: %s" % item)
        else:
            self.ids_seen.add(item['id'])
            return item
```

启用一个Item Pipeline组件

为了启用一个Item Pipeline组件，你必须将它的类添加到 `ITEM_PIPELINES` 配置，就像下面这个例子:

```
ITEM_PIPELINES = {
    'myproject.pipelines.PricePipeline': 300,
    'myproject.pipelines.JsonWriterPipeline': 800,
```

```
}
```

分配给每个类的整型值，确定了他们运行的顺序，item按数字从低到高的顺序，通过pipeline，通常将这些数字定义在0-1000范围内。

讨论

Feed exports

0.10 新版功能.

实现爬虫时最经常提到的需求就是能合适的保存爬取到的数据，或者说，生成一个带有爬取数据的”输出文件”(通常叫做”输出feed”)，来供其他系统使用。

Scrapy自带了Feed输出，并且支持多种序列化格式(serialization format)及存储方式(storage backends)。

序列化方式(Serialization formats)

feed输出使用到了 *Item exporters* 。其自带支持的类型有:

- *JSON*
- *JSON lines*
- *CSV*
- *XML*

您也可以通过 **FEED_EXPORTERS** 设置扩展支持的属性。

JSON

- **FEED_FORMAT**: `json`
- 使用的exporter: **JsonItemExporter**
- 大数据量情况下使用JSON请参见 *这个警告*

JSON lines

- **FEED_FORMAT**: `jsonlines`
- 使用的exporter: **JsonLinesItemExporter**

CSV

- **FEED_FORMAT**: `csv`
- 使用的exporter: **CsvItemExporter**

XML

- **FEED_FORMAT**: `xml`
- 使用的exporter: **XmlItemExporter**

Pickle

- **FEED_FORMAT**: `pickle`
- 使用的exporter: **PickleItemExporter**

Marshal

- **FEED_FORMAT**: `marshal`
- 使用的exporter: **MarshalItemExporter**

存储(Storages)

使用feed输出时您可以通过使用 URI [http://en.wikipedia.org/wiki/Uniform_Resource_Identifier] (通过 **FEED_URI** 设置) 来定义存储端。 feed输出支持URI方式支持的多种存储后端类型。

自带支持的存储后端有:

- *本地文件系统*
- *FTP*
- *S3* (需要 boto [http://code.google.com/p/boto/])
- *标准输出*

有些存储后端会因所需的外部库未安装而不可用。例如，S3只有在 boto [http://code.google.com/p/boto/] 库安装的情况下才可使用。

存储URI参数

存储URI也包含参数。当feed被创建时这些参数可以被覆盖:

- `%(time)s` - 当feed被创建时被timestamp覆盖
- `%(name)s` - 被spider的名字覆盖

其他命名的参数会被spider同名的属性所覆盖。例如， 当feed被创建时， `%(site_id)s` 将会被 `spider.site_id` 属性所覆盖。

下面用一些例子来说明:

- 存储在FTP，每个spider一个目录:
  - `ftp://user:password@ftp.example.com/scraping/feeds/%(name)s/%(time)s.json`
- 存储在S3，每一个spider一个目录:
  - `s3://mybucket/scraping/feeds/%(name)s/%(time)s.json`

存储端(Storage backends)

本地文件系统

将feed存储在本地系统。

- URI scheme: `file`
- URI样例: `file:///tmp/export.csv`
- 需要的外部依赖库: none

注意: (只有)存储在本地文件系统时，您可以指定一个绝对路径 `/tmp/export.csv` 并忽略协议(scheme)。不过这仅仅只能在Unix系统中工作。

FTP

将feed存储在FTP服务器。

- URI scheme: `ftp`
- URI样例: `ftp://user:pass@ftp.example.com/path/to/export.csv`
- 需要的外部依赖库: none

S3

将feed存储在 Amazon S3 [http://aws.amazon.com/s3/] 。

- URI scheme: `s3`
- URI样例:
  - `s3://mybucket/path/to/export.csv`
  - `s3://aws_key:aws_secret@mybucket/path/to/export.csv`
- 需要的外部依赖库: boto [http://code.google.com/p/boto/]

您可以通过在URI中传递user/pass来完成AWS认证，或者也可以通过下列的设置来完成:

- **`AWS_ACCESS_KEY_ID`**
- **`AWS_SECRET_ACCESS_KEY`**

标准输出

feed输出到Scrapy进程的标准输出。

- URI scheme: `stdout`
- URI样例: `stdout:`
- 需要的外部依赖库: none

设定(Settings)

这些是配置feed输出的设定:

- **`FEED_URI`** (必须)
- **`FEED_FORMAT`**
- **`FEED_STORAGES`**
- **`FEED_EXPORTERS`**
- **`FEED_STORE_EMPTY`**

FEED_URI

Default: `None`

输出feed的URI。支持的URI协议请参见 *存储端(Storage backends)* 。

为了启用feed输出，该设定是必须的。

FEED_FORMAT

输出feed的序列化格式。可用的值请参见 *序列化方式(Serialization formats)* 。

FEED_STORE_EMPTY

Default: `False`

是否输出空feed(没有item的feed)。

FEED_STORAGES

Default:: `{}`

包含项目支持的额外feed存储端的字典。 字典的键(key)是URI协议(scheme)，值是存储类(storage class)的路径。

FEED_STORAGES_BASE

Default:

```
{
    '': 'scrapy.contrib.feedexport.FileFeedStorage',
    'file': 'scrapy.contrib.feedexport.FileFeedStorage',
    'stdout': 'scrapy.contrib.feedexport.StdoutFeedStorage',
    's3': 'scrapy.contrib.feedexport.S3FeedStorage',
    'ftp': 'scrapy.contrib.feedexport.FTPFeedStorage',
}
```

包含Scrapy内置支持的feed存储端的字典。

FEED_EXPORTERS

Default:: `{}`

包含项目支持的额外输出器(exporter)的字典。 该字典的键(key)是URI协议(scheme)，值是 *Item输出器(exporter)* 类的路径。

FEED_EXPORTERS_BASE

Default:

```
FEED_EXPORTERS_BASE = {
    'json': 'scrapy.contrib.exporter.JsonItemExporter',
    'jsonlines': 'scrapy.contrib.exporter.JsonLinesItemExporter',
    'csv': 'scrapy.contrib.exporter.CsvItemExporter',
    'xml': 'scrapy.contrib.exporter.XmlItemExporter',
    'marshal': 'scrapy.contrib.exporter.MarshalItemExporter',
}
```

包含Scrapy内置支持的feed输出器(exporter)的字典。

讨论

Link Extractors

Link Extractors 是用于从网页(`scrapy.http.Response`)中抽取会被follow的链接的对象。

Scrapy默认提供2种可用的 Link Extractor, 但你通过实现一个简单的接口创建自己定制的Link Extractor来满足需求。Scrapy 提供了 `scrapy.contrib.linkextractors import LinkExtractor`， 不过您也可以通过实现一个简单的接口来创建您自己的Link Extractor，满足需求。

每个LinkExtractor有唯一的公共方法是 `extract_links`， 其接收 一个 `Response` 对象， 并返回 **`scrapy.link.Link`** 对象。Link Extractors只实例化一次，其 `extract_links` 方法会根据不同的response被调用多次来提取链接。

Link Extractors在 `CrawlSpider` 类(在Scrapy可用)中使用, 通过一套规则,但你也可以用它在你的Spider中,即使你不是从 `CrawlSpider` 继承的子类, 因为它的目的很简单: 提取链接。

内置Link Extractor 参考

Scrapy 自带的Link Extractors类在 `scrapy.contrib.linkextractors` 模块提供。

默认的link extractor 是 `LinkExtractor`， 其实就是 `LxmlLinkExtractor`:

```
from scrapy.contrib.linkextractors import LinkExtractor
```

在以前版本的Scrapy版本中提供了其他的link extractor，不过都已经被废弃了。

LxmlLinkExtractor

class `scrapy.contrib.linkextractors.lxmlhtml.`**`LxmlLinkExtractor`**(*allow=()*, *deny=()*, *allow_domains=()*, *deny_domains=()*, *deny_extensions=None*, *restrict_xpaths=()*, *tags=('a'*, *'area')*, *attrs=('href'*, *)*, *canonicalize=True*, *unique=True*, *process_value=None*)

LxmlLinkExtractor is the recommended link extractor with handy filtering options. It is implemented using lxml's robust HTMLParser.

参数:

- **allow** (*a regular expression (or list of)*) – a single regular expression (or list of regular expressions) that the (absolute) urls must match in order to be extracted. If not given (or empty), it will match all links.
- **deny** (*a regular expression (or list of)*) – a single regular expression (or list of regular expressions) that the (absolute) urls must match in order to be excluded (ie. not extracted). It has precedence over the `allow` parameter. If not given (or empty) it won't exclude any links.
- **allow_domains** (*str or list*) – a single value or a list of string containing domains which will be considered for extracting the links
- **deny_domains** (*str or list*) – a single value or a list of strings containing domains which won't be considered for extracting the links
- **deny_extensions** (*list*) – a single value or list of strings containing extensions that should be ignored when extracting links. If not given, it will default to the `IGNORED_EXTENSIONS` list defined in the scrapy.linkextractor module.
- **restrict_xpaths** (*str or list*) – is a XPath (or list of XPath's) which defines regions inside the response where links should be extracted from. If given, only the text selected by those XPath will be scanned for links. See examples below.
- **tags** (*str or list*) – a tag or a list of tags to consider when extracting links. Defaults to `('a', 'area')`.
- **attrs** (*list*) – an attribute or list of attributes which should be considered when looking for links to extract (only for those tags specified in the `tags` parameter). Defaults to `('href',)`
- **canonicalize** (*boolean*) – canonicalize each extracted url (using scrapy.utils.url.canonicalize_url). Defaults to `True`.
- **unique** (*boolean*) – whether duplicate filtering should be applied to extracted links.
- **process_value** (*callable*) –
  它接收来自扫描标签和属性提取每个值, 可以修改该值, 并返回一个新的, 或返回 `None` 完全忽略链接的功能。如果没有给出, `process_value` 默认是 `lambda x: x`。
  例如,从这段代码中提取链接:

  ```
  <a href="javascript:goToPage('../other/page.html'); return false">Link text</a>
  ```

  你可以使用下面的这个 `process_value` 函数:

  ```
  def process_value(value):
      m = re.search("javascript:goToPage\('(.*?)'", value)
      if m:
          return m.group(1)
  ```

讨论

Logging

Scrapy提供了log功能。您可以通过 `scrapy.log` 模块使用。当前底层实现使用了 Twisted logging [http://twistedmatrix.com/projects/core/documentation/howto/logging.html]， 不过可能在之后会有所变化。

log服务必须通过显示调用 `scrapy.log.start()` 来开启，以捕捉顶层的Scrapy日志消息。 在此之上，每个crawler都拥有独立的log观察者(observer)(创建时自动连接(attach)),接收其spider的日志消息。

Log levels

Scrapy提供5层logging级别:

1. `CRITICAL` - 严重错误(critical)
2. `ERROR` - 一般错误(regular errors)
3. `WARNING` - 警告信息(warning messages)
4. `INFO` - 一般信息(informational messages)
5. `DEBUG` - 调试信息(debugging messages)

如何设置log级别

您可以通过终端选项(command line option) *–loglevel/-L* 或 `LOG_LEVEL` 来设置log级别。

如何记录信息(log messages)

下面给出如何使用 `WARNING` 级别来记录信息的例子:

```
from scrapy import log
log.msg("This is a warning", level=log.WARNING)
```

在Spider中添加log(Logging from Spiders)

在spider中添加log的推荐方式是使用Spider的 `log()` 方法。该方法会自动在调用 `scrapy.log.msg()` 时赋值 `spider` 参数。其他的参数则直接传递给 `msg()` 方法。

scrapy.log模块

`scrapy.log.`**`start`**(*logfile=None*, *loglevel=None*, *logstdout=None*)

启动Scrapy顶层logger。该方法必须在记录任何顶层消息前被调用 (使用模块的 `msg()` 而不是 `Spider.log` 的消息)。否则，之前的消息将会丢失。

参数:
- **logfile** (*str*) – 用于保存log输出的文件路径。如果被忽略， `LOG_FILE` 设置会被使用。 如果两个参数都是 `None` ，log将会被输出到标准错误流(standard error)。
- **loglevel** – 记录的最低的log级别. 可用的值有: `CRITICAL`, `ERROR`, `WARNING`, `INFO` and `DEBUG`.
- **logstdout** (*boolean*) – 如果为 `True` ， 所有您的应用的标准输出(包括错误)将会被记录(logged instead)。 例如，如果您调用 “print ‘hello’” ，则’hello’ 会在Scrapy的log中被显示。 如果被忽略，则 `LOG_STDOUT` 设置会被使用。

`scrapy.log.`**`msg`**(*message*, *level=INFO*, *spider=None*)

记录信息(Log a message)

参数:
- **message** (*str*) – log的信息
- **level** – 该信息的log级别. 参考 *Log levels*.
- **spider** (`Spider` 对象) – 记录该信息的spider. 当记录的信息和特定的spider有关联时，该参数必须被使用。

`scrapy.log.`**`CRITICAL`**

严重错误的Log级别

`scrapy.log.`**`ERROR`**

错误的Log级别 Log level for errors

`scrapy.log.`**`WARNING`**

警告的Log级别 Log level for warnings

`scrapy.log.`**`INFO`**

记录信息的Log级别(生产部署时推荐的Log级别)

`scrapy.log.`**`DEBUG`**

调试信息的Log级别(开发时推荐的Log级别)

Logging设置

以下设置可以被用来配置logging:

- LOG_ENABLED
- LOG_ENCODING
- LOG_FILE
- LOG_LEVEL
- LOG_STDOUT

讨论

数据收集(Stats Collection)

Scrapy提供了方便的收集数据的机制。数据以key/value方式存储，值大多是计数值。 该机制叫做数据收集器(Stats Collector)，可以通过 *Crawler API* 的属性 **stats** 来使用。在下面的章节 *常见数据收集器使用方法* 将给出例子来说明。

无论数据收集(stats collection)开启或者关闭，数据收集器永远都是可用的。 因此您可以import进自己的模块并使用其API(增加值或者设置新的状态键(stat keys))。 该做法是为了简化数据收集的方法: 您不应该使用超过一行代码来收集您的spider，Scrpay扩展或任何您使用数据收集器代码里头的状态。

数据收集器的另一个特性是(在启用状态下)很高效，(在关闭情况下)非常高效(几乎察觉不到)。

数据收集器对每个spider保持一个状态表。当spider启动时，该表自动打开，当spider关闭时，自动关闭。

常见数据收集器使用方法

通过 **stats** 属性来使用数据收集器。 下面是在扩展中使用状态的例子:

```
class ExtensionThatAccessStats(object):

    def __init__(self, stats):
        self.stats = stats

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler.stats)
```

设置数据:

```
stats.set_value('hostname', socket.gethostname())
```

增加数据值:

```
stats.inc_value('pages_crawled')
```

当新的值比原来的值大时设置数据:

```
stats.max_value('max_items_scraped', value)
```

当新的值比原来的值小时设置数据:

```
stats.min_value('min_free_memory_percent', value)
```

获取数据:

```
>>> stats.get_value('pages_crawled')
8
```

获取所有数据:

```
>>> stats.get_stats()
{'pages_crawled': 1238, 'start_time': datetime.datetime(2009, 7, 14, 21, 47, 28, 977139)}
```

可用的数据收集器

除了基本的 **StatsCollector** ，Scrapy也提供了基于 **StatsCollector** 的数据收集器。 您可以通过 **STATS_CLASS** 设置来选择。默认使用的是 **MemoryStatsCollector** 。

MemoryStatsCollector

class scrapy.statscol.**MemoryStatsCollector**

一个简单的数据收集器。其在spider运行完毕后将其数据保存在内存中。数据可以通过 **spider_stats** 属性访问。该属性是一个以spider名字为键(key)的字典。

这是Scrapy的默认选择。

spider_stats

保存了每个spider最近一次爬取的状态的字典(dict)。该字典以spider名字为键，值也是字典。

DummyStatsCollector

class scrapy.statscol.**DummyStatsCollector**

该数据收集器并不做任何事情但非常高效(因为什么都不做(写文档的人真调皮o(╯□╰)o))。 您可以通过设置 **STATS_CLASS** 启用这个收集器，来关闭数据收集，提高效率。 不过，数据收集的性能负担相较于Scrapy其他的处理(例如分析页面)来说是非常小的。

讨论

发送email

虽然Python通过 smtplib [http://docs.python.org/library/smtplib.html] 库使得发送email变得很简单，Scrapy仍然提供了自己的实现。 该功能十分易用，同时由于采用了 Twisted非阻塞式(non-blocking)IO [http://twistedmatrix.com/documents/current/core/howto/defer-intro.html] ，其避免了对爬虫的非阻塞式IO的影响。 另外，其也提供了简单的API来发送附件。 通过一些 *settings* 设置，您可以很简单的进行配置。

简单例子

有两种方法可以创建邮件发送器(mail sender)。 您可以通过标准构造器(constructor)创建:

```
from scrapy.mail import MailSender
mailer = MailSender()
```

或者您可以传递一个Scrapy设置对象，其会参考 *settings*:

```
mailer = MailSender.from_settings(settings)
```

这是如何来发送邮件了(不包括附件):

```
mailer.send(to=["someone@example.com"], subject="Some subject", body="Some body", cc=["another@example.com"])
```

MailSender类参考手册

在Scrapy中发送email推荐使用MailSender。其同框架中其他的部分一样，使用了 Twisted非阻塞式(non-blocking)IO [http://twistedmatrix.com/documents/current/core/howto/defer-intro.html] 。

class `scrapy.mail.`**`MailSender`**(*smtphost=None*, *mailfrom=None*, *smtpuser=None*, *smtppass=None*, *smtpport=None*)

参数：
- **smtphost** (*str*) – 发送email的SMTP主机(host)。如果忽略，则使用 **`MAIL_HOST`** 。
- **mailfrom** (*str*) – 用于发送email的地址(address)(填入 `From:`) 。 如果忽略，则使用 **`MAIL_FROM`** 。
- **smtpuser** – SMTP用户。如果忽略,则使用 **`MAIL_USER`** 。 如果未给定，则将不会进行SMTP认证(authentication)。
- **smtppass** (*str*) – SMTP认证的密码
- **smtpport** (*int*) – SMTP连接的短裤
- **smtptls** – 强制使用STARTTLS
- **smtpssl** (*boolean*) – 强制使用SSL连接

classmethod **`from_settings`**(*settings*)

使用Scrapy设置对象来初始化对象。其会参考 *这些Scrapy设置*.

参数： **settings** (`scrapy.settings.Settings` object) – the e-mail recipients

`send`(*to*, *subject*, *body*, *cc=None*, *attachs=()*, *mimetype='text/plain'*)

发送email到给定的接收者。

参数：
- **to** (*list*) – email接收者
- **subject** (*str*) – email内容
- **cc** (*list*) – 抄送的人
- **body** (*str*) – email的内容
- **attachs** (*iterable*) – 可迭代的元组 `(attach_name, mimetype, file_object)`。 `attach_name` 是一个在email的附件中显示的名字的字符串， `mimetype` 是附件的mime类型， `file_object` 是包含附件内容的可读的文件对象。
- **mimetype** (*str*) – email的mime类型

Mail设置

这些设置定义了 **`MailSender`** 构造器的默认值。其使得在您不编写任何一行代码的情况下，为您的项目配置实现email通知的功能。

MAIL_FROM

默认值: `'scrapy@localhost'`

用于发送email的地址(address)(填入 `From:`) 。

MAIL_HOST

默认值: `'localhost'`

发送email的SMTP主机(host)。

MAIL_PORT

默认值: `25`

发用邮件的SMTP端口。

MAIL_USER

默认值: `None`

SMTP用户。如果未给定，则将不会进行SMTP认证(authentication)。

MAIL_PASS

默认值: `None`

用于SMTP认证，与 **`MAIL_USER`** 配套的密码。

MAIL_TLS

默认值: `False`

强制使用STARTTLS。STARTTLS能使得在已经存在的不安全连接上，通过使用SSL/TLS来实现安全连接。

MAIL_SSL

默认值: `False`

强制使用SSL加密连接。

讨论

Telnet终端(Telnet Console)

Scrapy提供了内置的telnet终端，以供检查，控制Scrapy运行的进程。 telnet仅仅是一个运行在Scrapy进程中的普通python终端。因此您可以在其中做任何事。

telnet终端是一个 *自带的Scrapy扩展* 。 该扩展默认为启用，不过您也可以关闭。 关于扩展的更多内容请参考 *Telnet console 扩展* 。

如何访问telnet终端

telnet终端监听设置中定义的 `TELNETCONSOLE_PORT` ， 默认为 `6023` 。 访问telnet请输入:

```
telnet localhost 6023
>>>
```

Windows及大多数Linux发行版都自带了所需的telnet程序。

telnet终端中可用的变量

telnet仅仅是一个运行在Scrapy进程中的普通python终端。因此您可以做任何事情，甚至是导入新终端。

telnet为了方便提供了一些默认定义的变量:

快捷名称	描述
`crawler`	Scrapy Crawler (`scrapy.crawler.Crawler` 对象)
`engine`	Crawler.engine属性
`spider`	当前激活的爬虫(spider)
`slot`	the engine slot
`extensions`	扩展管理器(manager) (Crawler.extensions属性)
`stats`	状态收集器 (Crawler.stats属性)
`settings`	Scrapy设置(setting)对象 (Crawler.settings属性)
`est`	打印引擎状态的报告
`prefs`	针对内存调试 (参考 *调试内存溢出*)
`p`	pprint.pprint 函数的简写
`hpy`	针对内存调试 (参考 *调试内存溢出*)

Telnet console usage examples

下面是使用telnet终端的一些例子:

查看引擎状态

在终端中您可以使用Scrapy引擎的 `est()` 方法来快速查看状态:

```
telnet localhost 6023
>>> est()
Execution engine status

time()-engine.start_time                        : 8.62972998619
engine.has_capacity()                           : False
len(engine.downloader.active)                   : 16
engine.scraper.is_idle()                        : False
engine.spider.name                              : followall
engine.spider_is_idle(engine.spider)            : False
engine.slot.closing                             : False
len(engine.slot.inprogress)                     : 16
len(engine.slot.scheduler.dqs or [])            : 0
len(engine.slot.scheduler.mqs)                  : 92
len(engine.scraper.slot.queue)                  : 0
len(engine.scraper.slot.active)                 : 0
engine.scraper.slot.active_size                 : 0
engine.scraper.slot.itemproc_size               : 0
engine.scraper.slot.needs_backout()             : False
```

暂停，恢复和停止Scrapy引擎

暂停:

```
telnet localhost 6023
>>> engine.pause()
>>>
```

恢复:

```
telnet localhost 6023
>>> engine.unpause()
>>>
```

停止:

```
telnet localhost 6023
>>> engine.stop()
Connection closed by foreign host.
```

Telnet终端信号

scrapy.telnet.**update_telnet_vars**(*telnet_vars*)

在telnet终端开启前发送该信号。您可以挂载(hook up)该信号来添加，移除或更新 telnet本地命名空间可用的变量。 您可以通过在您的处理函数(handler)中更新 `telnet_vars` 字典来实现该修改。

参数： **telnet_vars** (*dict*) – telnet变量的字典

Telnet设定

以下是终端的一些设定:

TELNETCONSOLE_PORT

Default: `[6023, 6073]`

telnet终端使用的端口范围。如果设为 `None` 或 `0` ， 则动态分配端口。

TELNETCONSOLE_HOST

默认: `'127.0.0.1'`

telnet终端监听的接口(interface)。

讨论

Web Service

webserver 被移动到另外一个项目中。

托管在:

https://github.com/scrapy/scrapy-jsonrpc

讨论

常见问题(FAQ)

Scrapy相BeautifulSoup或lxml比较,如何呢？

BeautifulSoup [http://www.crummy.com/software/BeautifulSoup/] 及 lxml [http://lxml.de/] 是HTML和XML的分析库。Scrapy则是 编写爬虫，爬取网页并获取数据的应用框架(application framework)。

Scrapy提供了内置的机制来提取数据(叫做 *选择器(selectors)*)。 但如果您觉得使用更为方便，也可以使用 BeautifulSoup [http://www.crummy.com/software/BeautifulSoup/] (或 lxml [http://lxml.de/])。 总之，它们仅仅是分析库，可以在任何Python代码中被导入及使用。

换句话说，拿Scrapy与 BeautifulSoup [http://www.crummy.com/software/BeautifulSoup/] (或 lxml [http://lxml.de/]) 比较就好像是拿 jinja2 [http://jinja.pocoo.org/2/] 与 Django [http://www.djangoproject.com] 相比。

Scrapy支持那些Python版本？

Scrapy仅仅支持Python 2.7。 Python2.6的支持从Scrapy 0.20开始被废弃了。

Scrapy支持Python 3么？

不。但是Python 3.3+的支持已经在计划中了。 现在，Scrapy支持Python 2.7。

参见

Scrapy支持那些Python版本？.

Scrapy是否从Django中”剽窃”了X呢？

也许吧，不过我们不喜欢这个词。我们认为 Django [http://www.djangoproject.com] 是一个很好的开源项目，同时也是 一个很好的参考对象，所以我们把其作为Scrapy的启发对象。

我们坚信，如果有些事情已经做得很好了，那就没必要再重复制造轮子。这个想法，作为 开源项目及免费软件的基石之一，不仅仅针对软件，也包括文档，过程，政策等等。 所以，与其自行解决每个问题，我们选择从其他已经很好地解决问题的项目中复制想法(copy idea)，并把注意力放在真正需要解决的问题上。

如果Scrapy能启发其他的项目，我们将为此而自豪。欢迎来抄(steal)我们！

Scrapy支持HTTP代理么？

是的。(从Scrapy 0.8开始)通过HTTP代理下载中间件对HTTP代理提供了支持。参考 **`HttpProxyMiddleware`**.

如何爬取属性在不同页面的item呢？

参考 *Passing additional data to callback functions*.

Scrapy退出，ImportError: Nomodule named win32api

这是个Twisted bug [http://twistedmatrix.com/trac/ticket/3707]， 您需要安装 pywin32 [http://sourceforge.net/projects/pywin32/]。

我要如何在spider里模拟用户登录呢?

参考 *使用FormRequest.from_response()方法模拟用户登录*.

Scrapy是以广度优先还是深度优先进行爬取的呢？

默认情况下，Scrapy使用 LIFO [http://en.wikipedia.org/wiki/LIFO] 队列来存储等待的请求。简单的说，就是 深度优先顺序 [http://en.wikipedia.org/wiki/Depth-first_search]。深度优先对大多数情况下是更方便的。如果您想以 广度优先顺序 [http://en.wikipedia.org/wiki/Breadth-first_search] 进行爬取，你可以设置以下的设定:

```
DEPTH_PRIORITY = 1
SCHEDULER_DISK_QUEUE = 'scrapy.squeue.PickleFifoDiskQueue'
SCHEDULER_MEMORY_QUEUE = 'scrapy.squeue.FifoMemoryQueue'
```

我的Scrapy爬虫有内存泄露，怎么办?

参考 *调试内存溢出*.

另外，Python自己也有内存泄露，在 *Leaks without leaks* 有所描述。

如何让Scrapy减少内存消耗？

参考上一个问题

我能在spider中使用基本HTTP认证么？

可以。参考 **HttpAuthMiddleware**.

为什么Scrapy下载了英文的页面，而不是我的本国语言？

尝试通过覆盖 **DEFAULT_REQUEST_HEADERS** 设置来修改默认的 Accept-Language [http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.4] 请求头。

我能在哪里找到Scrapy项目的例子？

参考 *例子*.

我能在不创建Scrapy项目的情况下运行一个爬虫(spider)么？

是的。您可以使用 **runspider** 命令。例如，如果您有个 spider写在 `my_spider.py` 文件中，您可以运行:

```
scrapy runspider my_spider.py
```

详情请参考 **runspider** 命令。

我收到了 “Filtered offsite request” 消息。如何修复？

这些消息(以 `DEBUG` 所记录)并不意味着有问题，所以你可以不修复它们。

这些消息由Offsite Spider中间件(Middleware)所抛出。 该(默认启用的)中间件筛选出了不属于当前spider的站点请求。

更多详情请参见: **OffsiteMiddleware**.

发布Scrapy爬虫到生产环境的推荐方式？

参见 *Scrapyd*.

我能对大数据(large exports)使用JSON么？

这取决于您的输出有多大。参考 **JsonItemExporter** 文档中的 *这个警告*

我能在信号处理器(signal handler)中返回(Twisted)引用么？

有些信号支持从处理器中返回引用，有些不行。参考 *内置信号参考手册(Built-in signals reference)* 来了解详情。

reponse返回的状态值999代表了什么？

999是雅虎用来控制请求量所定义的返回值。 试着减慢爬取速度，将spider的下载延迟改为 `2` 或更高:

```
class MySpider(CrawlSpider):

    name = 'myspider'

    download_delay = 2

    # [ ... rest of the spider code ... ]
```

或在 **DOWNLOAD_DELAY** 中设置项目的全局下载延迟。

我能在spider中调用 `pdb.set_trace()` 来调试么？

可以，但你也可以使用Scrapy终端。这能让你快速分析(甚至修改) spider处理返回的返回(response)。通常来说，比老旧的 `pdb.set_trace()` 有用多了。

更多详情请参考 *在spider中启动shell来查看response*.

将所有爬取到的item转存(dump)到JSON/CSV/XML文件的最简单的方法?

dump到JSON文件:

```
scrapy crawl myspider -o items.json
```

dump到CSV文件:

```
scrapy crawl myspider -o items.csv
```

dump到XML文件:

```
scrapy crawl myspider -o items.xml
```

更多详情请参考 *Feed exports*

在某些表单中巨大神秘的 __VIEWSTATE 参数是什么?

__VIEWSTATE 参数存在于ASP.NET/VB.NET建立的站点中。关于这个参数的作用请参考 这篇文章 [http://search.cpan.org/~ecarroll/HTML-TreeBuilderX-ASP_NET-0.09/lib/HTML/TreeBuilderX/ASP_NET.pm] 。这里有一个爬取这种站点的 样例爬虫 [http://github.com/AmbientLighter/rpn-fas/blob/master/fas/spiders/mp.py] 。

分析大XML/CSV数据源的最好方法是?

使用XPath选择器来分析大数据源可能会有问题。选择器需要在内存中对数据建立完整的 DOM树，这过程速度很慢且消耗大量内存。

为了避免一次性读取整个数据源，您可以使用 scrapy.utils.iterators 中的 xmliter 及 csviter 方法。 实际上，这也是feed spider(参考 *Spiders*)中的处理方法。

Scrapy自动管理cookies么?

是的，Scrapy接收并保持服务器返回来的cookies，在之后的请求会发送回去，就像正常的网页浏览器做的那样。

更多详情请参考 *Requests and Responses* 及 *CookiesMiddleware* 。

如何才能看到Scrapy发出及接收到的Cookies呢?

启用 COOKIES_DEBUG 选项。

要怎么停止爬虫呢?

在回调函数中raise CloseSpider 异常。 更多详情请参见: CloseSpider 。

如何避免我的Scrapy机器人(bot)被禁止(ban)呢?

参考 *避免被禁止(ban)*.

我应该使用spider参数(arguments)还是设置(settings)来配置spider呢?

spider参数 及 *设置(settings)* 都可以用来配置您的spider。 没有什么强制的规则来限定要使用哪个，但设置(settings)更适合那些一旦设置就不怎么会修改的参数， 而spider参数则意味着修改更为频繁，在每次spider运行都有修改，甚至是spider运行所必须的元素 (例如，设置spider的起始url)。

这里以例子来说明这个问题。假设您有一个spider需要登录某个网站来 爬取数据，并且仅仅想爬取特定网站的特定部分(每次都不一定相同)。 在这个情况下，认证的信息将写在设置中，而爬取的特定部分的url将是spider参数。

我爬取了一个XML文档但是XPath选择器不返回任何的item

也许您需要移除命名空间(namespace)。参见 *移除命名空间*.

讨论

调试(Debugging)Spiders

本篇介绍了调试spider的常用技术。 考虑下面的spider:

```
import scrapy
from myproject.items import MyItem

class MySpider(scrapy.Spider):
    name = 'myspider'
    start_urls = (
        'http://example.com/page1',
        'http://example.com/page2',
        )

    def parse(self, response):
        # collect `item_urls`
        for item_url in item_urls:
            yield scrapy.Request(item_url, self.parse_item)

    def parse_item(self, response):
        item = MyItem()
        # populate `item` fields
        # and extract item_details_url
        yield scrapy.Request(item_details_url, self.parse_details, meta={'item': item})

    def parse_details(self, response):
        item = response.meta['item']
        # populate more `item` fields
        return item
```

简单地说，该spider分析了两个包含item的页面(start_urls)。Item有详情页面， 所以我们使用 **Request** 的 `meta` 功能来传递已经部分获取的item。

Parse命令

检查spier输出的最基本方法是使用 **parse** 命令。这能让你在函数层(method level)上检查spider 各个部分的效果。其十分灵活并且易用，不过不能在代码中调试。

查看特定url爬取到的item:

```
$ scrapy parse --spider=myspider -c parse_item -d 2 <item_url>
[ ... scrapy log lines crawling example.com spider ... ]

>>> STATUS DEPTH LEVEL 2 <<<
# Scraped Items  ------------------------------------------------------------
[{'url': <item_url>}]

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

使用 `--verbose` 或 `-v` 选项，查看各个层次的状态:

```
$ scrapy parse --spider=myspider -c parse_item -d 2 -v <item_url>
[ ... scrapy log lines crawling example.com spider ... ]

>>> DEPTH LEVEL: 1 <<<
# Scraped Items  ------------------------------------------------------------
[]

# Requests  -----------------------------------------------------------------
[<GET item_details_url>]


>>> DEPTH LEVEL: 2 <<<
# Scraped Items  ------------------------------------------------------------
[{'url': <item_url>}]

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

检查从单个start_url爬取到的item也是很简单的:

```
$ scrapy parse --spider=myspider -d 3 'http://example.com/page1'
```

Scrapy终端(Shell)

尽管 **parse** 命令对检查spider的效果十分有用，但除了显示收到的response及输出外， 其对检查回调函数内部的过程并没有提供什么便利。 如何调试 `parse_detail` 没有收到item的情况呢？

幸运的是，救世主 **shell** 出现了(参考 *在spider中启动shell来查看response*):

```
from scrapy.shell import inspect_response

def parse_details(self, response):
    item = response.meta.get('item', None)
    if item:
        # populate more `item` fields
        return item
    else:
        inspect_response(response, self)
```

参考 *在spider中启动shell来查看response* 。

在浏览器中打开

有时候您想查看某个response在浏览器中显示的效果，这是可以使用 `open_in_browser` 功能。下面是使用的例子:

```
from scrapy.utils.response import open_in_browser

def parse_details(self, response):
    if "item name" not in response.body:
        open_in_browser(response)
```

`open_in_browser` 将会使用Scrapy获取到的response来打开浏览器，并且调整 base tag [http://www.w3schools.com/tags/tag_base.asp] 使得图片及样式(style)能正常显示。

Logging

记录(logging)是另一个获取到spider运行信息的方法。虽然不是那么方便， 但好处是log的内容在以后的运行中也可以看到:

```
from scrapy import log

def parse_details(self, response):
    item = response.meta.get('item', None)
    if item:
        # populate more `item` fields
        return item
    else:
        self.log('No item received for %s' % response.url,
            level=log.WARNING)
```

更多内容请参见 *Logging* 部分。

讨论

Spiders Contracts

0.15 新版功能.

注解

这是一个新引入(Scrapy 0.15)的特性，在后续的功能/API更新中可能有所改变，查看 *release notes* 来了解更新。

测试spider是一件挺烦人的事情，尤其是只能编写单元测试(unit test)没有其他办法时，就更恼人了。 Scrapy通过合同(contract)的方式来提供了测试spider的集成方法。

您可以硬编码(hardcode)一个样例(sample)url， 设置多个条件来测试回调函数处理repsponse的结果，来测试spider的回调函数。 每个contract包含在文档字符串(docstring)里，以 @ 开头。 查看下面的例子:

```
def parse(self, response):
    """ This function parses a sample response. Some contracts are mingled
    with this docstring.

    @url http://www.amazon.com/s?field-keywords=selfish+gene
    @returns items 1 16
    @returns requests 0 0
    @scrapes Title Author Year Price
    """
```

该回调函数使用了三个内置的contract来测试:

class scrapy.contracts.default.**UrlContract**

该constract(@url)设置了用于检查spider的其他constract状态的样例url。 该contract是必须的，所有缺失该contract的回调函数在测试时将会被忽略:

```
@url url
```

class scrapy.contracts.default.**ReturnsContract**

该contract(@returns)设置spider返回的items和requests的上界和下界。 上界是可选的:

```
@returns item(s)|request(s) [min [max]]
```

class scrapy.contracts.default.**ScrapesContract**

该contract(@scrapes)检查回调函数返回的所有item是否有特定的fields:

```
@scrapes field_1 field_2 ...
```

使用 **check** 命令来运行contract检查。

自定义Contracts

如果您想要比内置scrapy contract更为强大的功能，可以在您的项目里创建并设置您自己的 contract，并使用 **SPIDER_CONTRACTS** 设置来加载:

```
SPIDER_CONTRACTS = {
    'myproject.contracts.ResponseCheck': 10,
    'myproject.contracts.ItemValidate': 10,
}
```

每个contract必须继承 **scrapy.contracts.Contract** 并覆盖下列三个方法:

class scrapy.contracts.**Contract**(*method*, **args*)

参数：
- **method** (*function*) – contract所关联的回调函数
- **args** (*list*) – 传入docstring的(以空格区分的)argument列表(list)

adjust_request_args(*args*)

接收一个 字典(dict) 作为参数。该参数包含了所有 **Request** 对象 参数的默认值。该方法必须返回相同或修改过的字典。

pre_process(*response*)

该函数在sample request接收到response后，传送给回调函数前被调用，运行测试。

post_process(*output*)

该函数处理回调函数的输出。迭代器(Iterators)在传输给该函数前会被列表化(listified)。

该样例contract在response接收时检查了是否有自定义header。 在失败时Raise **scrapy.exceptions.ContractFaild** 来展现错误:

```python
from scrapy.contracts import Contract
from scrapy.exceptions import ContractFail

class HasHeaderContract(Contract):
    """ Demo contract which checks the presence of a custom header
        @has_header X-CustomHeader
    """

    name = 'has_header'

    def pre_process(self, response):
        for header in self.args:
            if header not in response.headers:
                raise ContractFail('X-CustomHeader not present')
```

讨论

实践经验(Common Practices)

本章节记录了使用Scrapy的一些实践经验(common practices)。 这包含了很多使用不会包含在其他特定章节的的内容。

在脚本中运行Scrapy

除了常用的 `scrapy crawl` 来启动Scrapy，您也可以使用 *API* 在脚本中启动Scrapy。

需要注意的是，Scrapy是在Twisted异步网络库上构建的， 因此其必须在Twisted reactor里运行。

另外，在spider运行结束后，您必须自行关闭Twisted reactor。 这可以通过在 `CrawlerRunner.crawl` 所返回的对象中添加回调函数来实现。

下面给出了如何实现的例子，使用 testspiders [https://github.com/scrapinghub/testspiders] 项目作为例子。

```
from twisted.internet import reactor
from scrapy.crawler import CrawlerRunner
from scrapy.utils.project import get_project_settings

runner = CrawlerRunner(get_project_settings())

# 'followall' is the name of one of the spiders of the project.
d = runner.crawl('followall', domain='scrapinghub.com')
d.addBoth(lambda _: reactor.stop())
reactor.run() # the script will block here until the crawling is finished
```

Running spiders outside projects it's not much different. You have to create a generic `Settings` object and populate it as needed (See *内置设定参考手册* for the available settings), instead of using the configuration returned by *get_project_settings*.

Spiders can still be referenced by their name if `SPIDER_MODULES` is set with the modules where Scrapy should look for spiders. Otherwise, passing the spider class as first argument in the `CrawlerRunner.crawl` method is enough.

```
from twisted.internet import reactor
from scrapy.spider import Spider
from scrapy.crawler import CrawlerRunner
from scrapy.settings import Settings

class MySpider(Spider):
    # Your spider definition
    ...

settings = Settings({'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)'})
runner = CrawlerRunner(settings)

d = runner.crawl(MySpider)
d.addBoth(lambda _: reactor.stop())
reactor.run() # the script will block here until the crawling is finished
```

参见

Twisted Reactor Overview [http://twistedmatrix.com/documents/current/core/howto/reactor-basics.html].

同一进程运行多个spider

默认情况下，当您执行 `scrapy crawl` 时，Scrapy每个进程运行一个spider。 当然，Scrapy通过 *内部(internal)API* 也支持单进程多个spider。

下面以 testspiders [https://github.com/scrapinghub/testspiders] 作为例子来说明如何同时运行多个spider:

```
from twisted.internet import reactor, defer
from scrapy.crawler import CrawlerRunner
from scrapy.utils.project import get_project_settings

runner = CrawlerRunner(get_project_settings())
dfs = set()
for domain in ['scrapinghub.com', 'insophia.com']:
    d = runner.crawl('followall', domain=domain)
    dfs.add(d)

defer.DeferredList(dfs).addBoth(lambda _: reactor.stop())
reactor.run() # the script will block here until all crawling jobs are finished
```

相同的例子，不过通过链接(chaining) deferred来线性运行spider:

```
from twisted.internet import reactor, defer
```

```
from scrapy.crawler import CrawlerRunner
from scrapy.utils.project import get_project_settings

runner = CrawlerRunner(get_project_settings())

@defer.inlineCallbacks
def crawl():
    for domain in ['scrapinghub.com', 'insophia.com']:
        yield runner.crawl('followall', domain=domain)
    reactor.stop()

crawl()
reactor.run() # the script will block here until the last crawl call is finished
```

参见

在脚本中运行Scrapy.

分布式爬虫(Distributed crawls)

Scrapy并没有提供内置的机制支持分布式(多服务器)爬取。不过还是有办法进行分布式爬取， 取决于您要怎么分布了。

如果您有很多spider，那分布负载最简单的办法就是启动多个Scrapyd，并分配到不同机器上。

如果想要在多个机器上运行一个单独的spider，那您可以将要爬取的url进行分块，并发送给spider。 例如:

首先，准备要爬取的url列表，并分配到不同的文件url里:

```
http://somedomain.com/urls-to-crawl/spider1/part1.list
http://somedomain.com/urls-to-crawl/spider1/part2.list
http://somedomain.com/urls-to-crawl/spider1/part3.list
```

接着在3个不同的Scrapd服务器中启动spider。spider会接收一个(spider)参数 `part` ， 该参数表示要爬取的分块:

```
curl http://scrapy1.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=1
curl http://scrapy2.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=2
curl http://scrapy3.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=3
```

避免被禁止(ban)

有些网站实现了特定的机制，以一定规则来避免被爬虫爬取。 与这些规则打交道并不容易，需要技巧，有时候也需要些特别的基础。 如果有疑问请考虑联系 商业支持 [http://scrapy.org/support/] 。

下面是些处理这些站点的建议(tips):

- 使用user agent池，轮流选择之一来作为user agent。池中包含常见的浏览器的user agent(google一下一大堆)
- 禁止cookies(参考 **COOKIES_ENABLED**)，有些站点会使用cookies来发现爬虫的轨迹。
- 设置下载延迟(2或更高)。参考 **DOWNLOAD_DELAY** 设置。
- 如果可行，使用 Google cache [http://www.googleguide.com/cached_pages.html] 来爬取数据，而不是直接访问站点。
- 使用IP池。例如免费的 Tor项目 [https://www.torproject.org/] 或付费服务(ProxyMesh [http://proxymesh.com/])。
- 使用高度分布式的下载器(downloader)来绕过禁止(ban)，您就只需要专注分析处理页面。这样的例子有: Crawlera [http://crawlera.com]

如果您仍然无法避免被ban，考虑联系 商业支持 [http://scrapy.org/support/].

动态创建Item类

对于有些应用，item的结构由用户输入或者其他变化的情况所控制。您可以动态创建class。

```
from scrapy.item import DictItem, Field

def create_item_class(class_name, field_list):
fields = {field_name: Field() for field_name in field_list}

return type(class_name, (DictItem,), {'fields': fields})
```

讨论

通用爬虫(Broad Crawls)

Scrapy默认对特定爬取进行优化。这些站点一般被一个单独的Scrapy spider进行处理， 不过这并不是必须或要求的(例如，也有通用的爬虫能处理任何给定的站点)。

除了这种爬取完某个站点或没有更多请求就停止的"专注的爬虫"，还有一种通用的爬取类型，其能爬取大量(甚至是无限)的网站， 仅仅受限于时间或其他的限制。 这种爬虫叫做"通用爬虫(broad crawls)"，一般用于搜索引擎。

通用爬虫一般有以下通用特性:

- 其爬取大量(一般来说是无限)的网站而不是特定的一些网站。
- 其不会将整个网站都爬取完毕，因为这十分不实际(或者说是不可能)完成的。相反，其会限制爬取的时间及数量。
- 其在逻辑上十分简单(相较于具有很多提取规则的复杂的spider)，数据会在另外的阶段进行后处理(post-processed)
- 其并行爬取大量网站以避免被某个网站的限制所限制爬取的速度(为表示尊重，每个站点爬取速度很慢但同时爬取很多站点)。

正如上面所述，Scrapy默认设置是对特定爬虫做了优化，而不是通用爬虫。不过， 鉴于其使用了异步架构，Scrapy对通用爬虫也十分适用。 本篇文章总结了一些将Scrapy作为通用爬虫所需要的技巧， 以及相应针对通用爬虫的Scrapy设定的一些建议。

增加并发

并发是指同时处理的request的数量。其有全局限制和局部(每个网站)的限制。

Scrapy默认的全局并发限制对同时爬取大量网站的情况并不适用，因此您需要增加这个值。 增加多少取决于您的爬虫能占用多少CPU。 一般开始可以设置为 `100` 。不过最好的方式是做一些测试，获得Scrapy进程占取CPU与并发数的关系。 为了优化性能，您应该选择一个能使CPU占用率在80%-90%的并发数。

增加全局并发数:

```
CONCURRENT_REQUESTS = 100
```

降低log级别

当进行通用爬取时，一般您所注意的仅仅是爬取的速率以及遇到的错误。 Scrapy使用 `INFO` log级别来报告这些信息。为了减少CPU使用率(及记录log存储的要求), 在生产环境中进行通用爬取时您不应该使用 `DEBUG` log级别。 不过在开发的时候使用 `DEBUG` 应该还能接受。

设置Log级别:

```
LOG_LEVEL = 'INFO'
```

禁止cookies

除非您 *真的* 需要，否则请禁止cookies。在进行通用爬取时cookies并不需要， (搜索引擎则忽略cookies)。禁止cookies能减少CPU使用率及Scrapy爬虫在内存中记录的踪迹，提高性能。

禁止cookies:

```
COOKIES_ENABLED = False
```

禁止重试

对失败的HTTP请求进行重试会减慢爬取的效率，尤其是当站点响应很慢(甚至失败)时， 访问这样的站点会造成超时并重试多次。这是不必要的，同时也占用了爬虫爬取其他站点的能力。

禁止重试:

```
RETRY_ENABLED = False
```

减小下载超时

如果您对一个非常慢的连接进行爬取(一般对通用爬虫来说并不重要)， 减小下载超时能让卡住的连接能被快速的放弃并解放处理其他站点的能力。

减小下载超时:

```
DOWNLOAD_TIMEOUT = 15
```

禁止重定向

除非您对跟进重定向感兴趣，否则请考虑关闭重定向。 当进行通用爬取时，一般的做法是保存重定向的地址，并在之后的爬取进行解析。这保证了每批爬取的request数目在一定的数量， 否则重定向循环可能会导致爬虫在某个站点耗费过多资源。

关闭重定向:

```
REDIRECT_ENABLED = False
```

启用 “Ajax Crawlable Pages” 爬取

有些站点(基于2013年的经验数据，之多有1%)声明其为 ajax crawlable [https://developers.google.com/webmasters/ajax-crawling/docs/getting-started] 。这意味着该网站提供了原本只有ajax获取到的数据的纯HTML版本。 网站通过两种方法声明:

1. 在url中使用 `#!` - 这是默认的方式;
2. 使用特殊的meta标签 - 这在”main”, “index” 页面中使用。

Scrapy自动解决(1)；解决(2)您需要启用 *AjaxCrawlMiddleware*:

```
AJAXCRAWL_ENABLED = True
```

通用爬取经常抓取大量的 “index” 页面； AjaxCrawlMiddleware能帮助您正确地爬取。 由于有些性能问题，且对于特定爬虫没有什么意义，该中间默认关闭。

讨论

借助Firefox来爬取

这里介绍一些使用Firefox进行爬取的点子及建议，以及一些帮助爬取的Firefox实用插件。

在浏览器中检查DOM的注意事项

Firefox插件操作的是活动的浏览器DOM(live browser DOM)，这意味着当您检查网页源码的时候， 其已经不是原始的HTML，而是经过浏览器清理并执行一些Javascript代码后的结果。 Firefox是个典型的例子，其会在table中添加 `<tbody>` 元素。 而Scrapy相反，其并不修改原始的HTML，因此如果在XPath表达式中使用 `<tbody>` ，您将获取不到任何数据。

所以，当XPath配合Firefox使用时您需要记住以下几点:

- 当检查DOM来查找Scrapy使用的XPath时，禁用Firefox的Javascrpit。
- 永远不要用完整的XPath路径。使用相对及基于属性(例如 `id` ， `class` ， `width` 等)的路径 或者具有区别性的特性例如 `contains(@href, 'image')` 。
- 永远不要在XPath表达式中加入 `<tbody>` 元素，除非您知道您在做什么

对爬取有帮助的实用Firefox插件

Firebug

Firebug [http://getfirebug.com] 是一个在web开发者间很著名的工具，其对抓取也十分有用。 尤其是 检查元素(Inspect Element) [http://www.youtube.com/watch?v=-pT_pDe54aA] 特性对构建抓取数据的XPath十分方便。 当移动鼠标在页面元素时，您能查看相应元素的HTML源码。

查看 *使用Firebug进行爬取* ，了解如何配合Scrapy使用Firebug的详细教程。

XPather

XPather [https://addons.mozilla.org/firefox/addon/1192] 能让你在页面上直接测试XPath表达式。

XPath Checker

XPath Checker [https://addons.mozilla.org/firefox/addon/1095] 是另一个用于测试XPath表达式的Firefox插件。

Tamper Data

Tamper Data [http://addons.mozilla.org/firefox/addon/966] 是一个允许您查看及修改Firefox发送的header的插件。Firebug能查看HTTP header，但无法修改。

Firecookie

Firecookie [https://addons.mozilla.org/firefox/addon/6683] 使得查看及管理cookie变得简单。您可以使用这个插件来创建新的cookie， 删除存在的cookie，查看当前站点的cookie，管理cookie的权限及其他功能。

讨论

使用Firebug进行爬取

注解

本教程所使用的样例站Google Directory已经 被Google关闭 [http://searchenginewatch.com/article/2096661/Google-Directory-Has-Been-Shut-Down] 了。不过教程中的概念任然适用。 如果您打算使用一个新的网站来更新本教程，您的贡献是再欢迎不过了。 详细信息请参考 *Contributing to Scrapy* 。

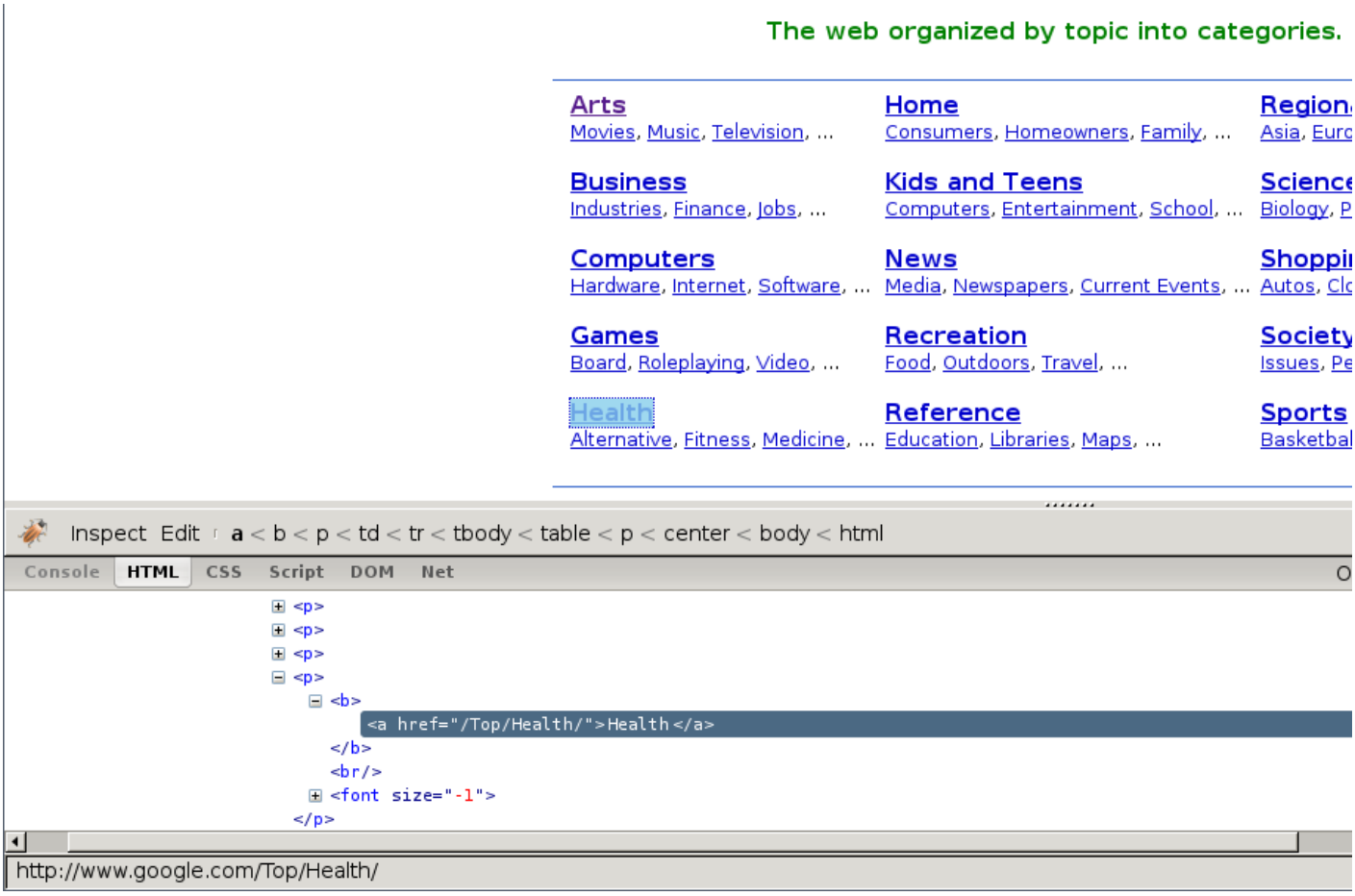
介绍

本文档介绍了如何适用 Firebug [http://getfirebug.com] (一个Firefox的插件)来使得爬取更为简单，有趣。 更多有意思的Firefox插件请参考 *对爬取有帮助的实用Firefox插件* 。 使用Firefox插件检查页面需要有些注意事项: *在浏览器中检查DOM的注意事项* 。

在本样例中将展现如何使用 Firebug [http://getfirebug.com] 从 Google Directory [http://directory.google.com/] 来爬取数据。 Google Directory [http://directory.google.com/] 包含了 *入门教程* 里所使用的 Open Directory Project [http://www.dmoz.org] 中一样的数据，不过有着不同的结构。

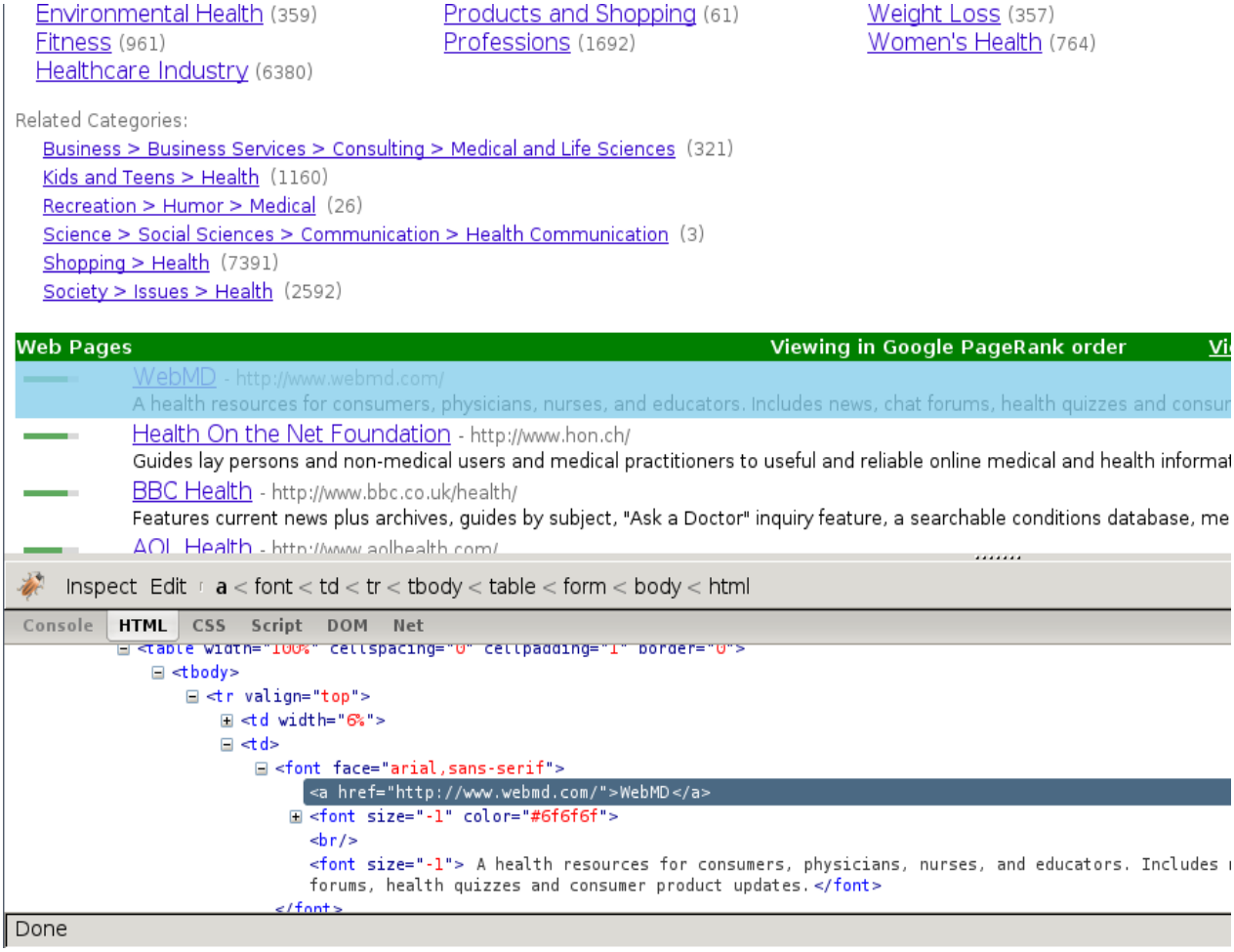
Firebug提供了非常实用的 检查元素 [http://www.youtube.com/watch?v=-pT_pDe54aA] 功能。该功能允许您将鼠标悬浮在不同的页面元素上， 显示相应元素的HTML代码。否则，您只能十分痛苦的在HTML的body中手动搜索标签。

在下列截图中，您将看到 检查元素 [http://www.youtube.com/watch?v=-pT_pDe54aA] 的执行效果。



首先我们能看到目录根据种类进行分类的同时，还划分了子类。

不过，看起来子类还有更多的子类，而不仅仅是页面显示的这些，所以我们接着查找:



正如路径的概念那样，子类包含了其他子类的链接，同时也链接到实际的网站中。

获取到跟进(follow)的链接

查看路径的URL，我们可以看到URL的通用模式(pattern):

http://directory.google.com/Category/Subcategory/Another_Subcategory

了解到这个消息，我们可以构建一个跟进的链接的正则表达式:

```
directory\.google\.com/[A-Z][a-zA-Z_/]+$
```

因此，根据这个表达式，我们创建第一个爬取规则:

```
Rule(LinkExtractor(allow='directory.google.com/[A-Z][a-zA-Z_/]+$', ),
    'parse_category',
    follow=True,
),
```

`Rule` 对象指导基于 `CrawlSpider` 的spider如何跟进目录链接。 `parse_category` 是spider的方法，用于从页面中处理也提取数据。

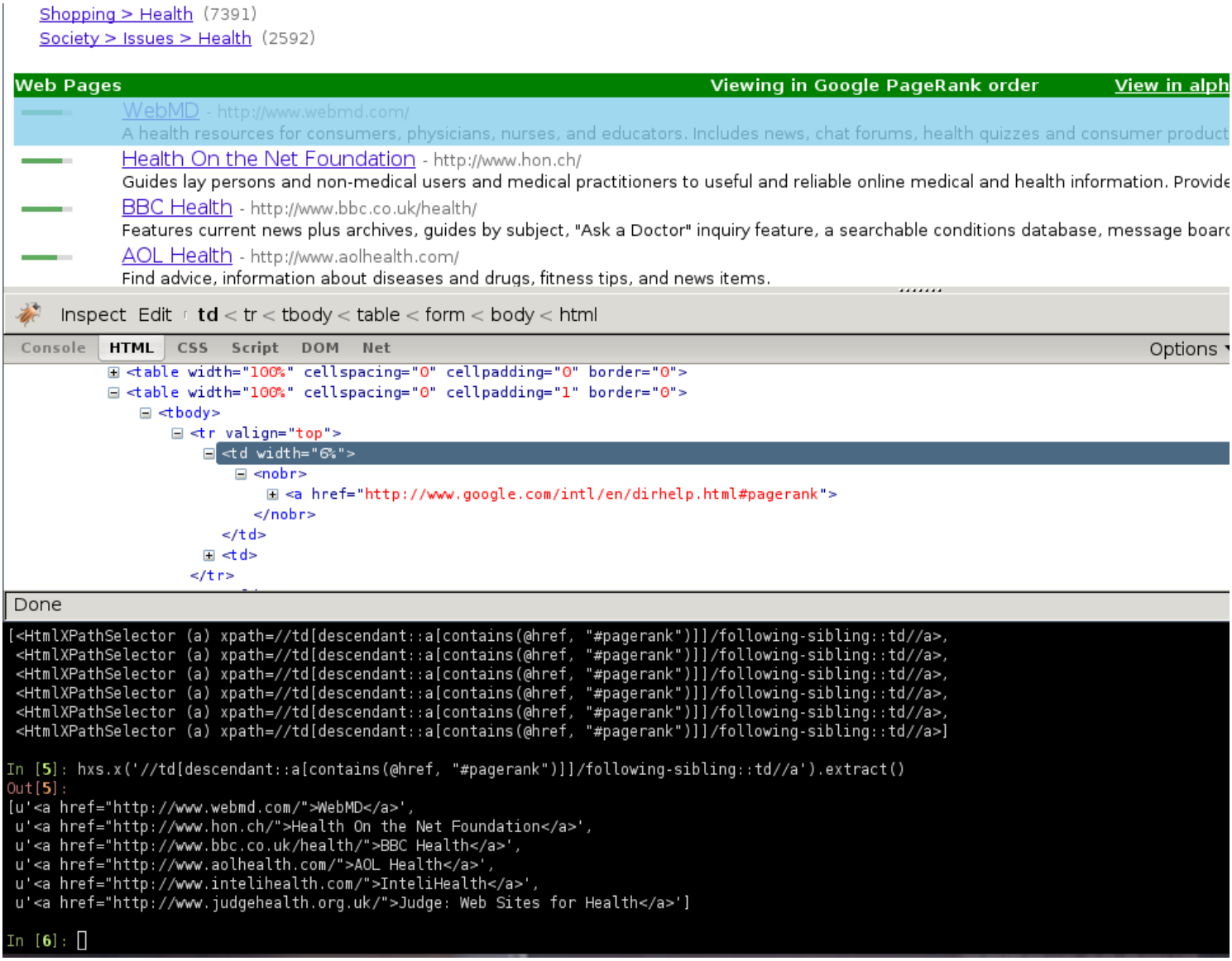
spider的代码如下:

```
from scrapy.contrib.linkextractors import LinkExtractor
from scrapy.contrib.spiders import CrawlSpider, Rule

class GoogleDirectorySpider(CrawlSpider):
    name = 'directory.google.com'
    allowed_domains = ['directory.google.com']
    start_urls = ['http://directory.google.com/']

    rules = (
        Rule(LinkExtractor(allow='directory\.google\.com/[A-Z][a-zA-Z_/]+$'),
            'parse_category', follow=True,
        ),
    )

    def parse_category(self, response):
        # write the category page data extraction code here
        pass
```

提取数据

现在我们来编写提取数据的代码。

在Firebug的帮助下，我们将查看一些包含网站链接的网页(以 http://directory.google.com/Top/Arts/Awards/ 为例)， 找到使用 *Selectors* 提取链接的方法。 我们也将使用 *Scrapy shell* 来测试得到的XPath表达式，确保表达式工作符合预期。



正如您所看到的那样，页面的标记并不是十分明显: 元素并不包含 `id`， `class` 或任何可以区分的属性。所以我们将使用等级槽(rank bar)作为指示点来选择提取的数据，创建XPath。

使用Firebug，我们可以看到每个链接都在 `td` 标签中。该标签存在于同时(在另一个 `td`)包含链接的等级槽(ranking bar)的 `tr` 中。

所以我们选择等级槽(ranking bar)，接着找到其父节点(`tr`)，最后是(包含我们要爬取数据的)链接的 `td` 。

对应的XPath:

```
//td[descendant::a[contains(@href, "#pagerank")]]/following-sibling::td//a
```

使用 *Scrapy终端* 来测试这些复杂的XPath表达式，确保其工作符合预期。

简单来说，该表达式会查找等级槽的 `td` 元素，接着选择所有 `td` 元素，该元素拥有子孙 `a` 元素，且 `a` 元素的属性 `href` 包含字符串 `#pagerank` 。

当然，这不是唯一的XPath，也许也不是选择数据的最简单的那个。 其他的方法也可能是，例如，选择灰色的链接的 `font` 标签。

最终，我们编写 `parse_category()` 方法:

```
def parse_category(self, response):

    # The path to website links in directory page
    links = response.xpath('//td[descendant::a[contains(@href, "#pagerank")]]/following-sibling::td/font')

    for link in links:
        item = DirectoryItem()
        item['name'] = link.xpath('a/text()').extract()
        item['url'] = link.xpath('a/@href').extract()
        item['description'] = link.xpath('font[2]/text()').extract()
        yield item
```

注意，您可能会遇到有些在Firebug找到，但是在原始HTML中找不到的元素， 例如典型的 <tbody> 元素， 或者Firebug检查活动DOM(live DOM)所看到的元素，但元素由javascript动态生成，并不在HTML源码中。 (原文语句乱了,上面为意译- -: or tags which Therefer in page HTML sources may on Firebug inspects the live DOM)

讨论

调试内存溢出

在Scrapy中，类似Requests, Response及Items的对象具有有限的生命周期: 他们被创建，使用，最后被销毁。

这些对象中，Request的生命周期应该是最长的，其会在调度队列(Scheduler queue)中一直等待，直到被处理。 更多内容请参考 *架构概览* 。

由于这些Scrapy对象拥有很长的生命，因此将这些对象存储在内存而没有正确释放的危险总是存在。 而这导致了所谓的"内存泄露"。

为了帮助调试内存泄露，Scrapy提供了跟踪对象引用的机制，叫做 *trackref* ， 或者您也可以使用第三方提供的更先进内存调试库 *Guppy* (更多内容请查看下面)。而这都必须在 *Telnet终端* 中使用。

内存泄露的常见原因

内存泄露经常是由于Scrapy开发者在Requests中(有意或无意)传递对象的引用(例如，使用 `meta` 属性或request回调函数)，使得该对象的生命周期与 Request的生命周期所绑定。这是目前为止最常见的内存泄露的原因， 同时对新手来说也是一个比较难调试的问题。

在大项目中，spider是由不同的人所编写的。而这其中有的spider可能是有"泄露的"， 当所有的爬虫同时运行时，这些影响了其他(写好)的爬虫，最终，影响了整个爬取进程。

与此同时，在不限制框架的功能的同时避免造成这些造成泄露的原因是十分困难的。因此， 我们决定不限制这些功能而是提供调试这些泄露的实用工具。这些工具回答了一个问题: *哪个spider在泄露* 。

内存泄露可能存在与一个您编写的中间件，管道(pipeline) 或扩展，在代码中您没有正确释放 (之前分配的)资源。例如，您在 `spider_opened` 中分配资源但在 `spider_closed` 中没有释放它们。

使用 `trackref` 调试内存泄露

`trackref` 是Scrapy提供用于调试大部分内存泄露情况的模块。 简单来说，其追踪了所有活动(live)的Request, Request, Item及Selector对象的引用。

您可以进入telnet终端并通过 `prefs()` 功能来检查多少(上面所提到的)活跃(alive)对象。 `pref()` 是 `print_live_refs()` 功能的引用:

```
telnet localhost 6023

>>> prefs()
Live References

ExampleSpider                       1   oldest: 15s ago
HtmlResponse                       10   oldest: 1s ago
Selector                            2   oldest: 0s ago
FormRequest                       878   oldest: 7s ago
```

正如所见，报告也展现了每个类中最老的对象的时间(age)。 If you're running multiple spiders per process chances are you can figure out which spider is leaking by looking at the oldest request or response. You can get the oldest object of each class using the `get_oldest()` function (from the telnet console).

如果您有内存泄露，那您能找到哪个spider正在泄露的机会是查看最老的request或response。 您可以使用 `get_oldest()` 方法来获取每个类中最老的对象， 正如此所示(在终端中)(原文档没有样例)。

哪些对象被追踪了?

`trackref` 追踪的对象包括以下类(及其子类)的对象:

- `scrapy.http.Request`
- `scrapy.http.Response`
- `scrapy.item.Item`
- `scrapy.selector.Selector`
- `scrapy.spider.Spider`

真实例子

让我们来看一个假设的具有内存泄露的准确例子。

假如我们有些spider的代码中有一行类似于这样的代码:

```
return Request("http://www.somenastyspider.com/product.php?pid=%d" % product_id,
    callback=self.parse, meta={referer: response}")
```

代码中在request中传递了一个response的引用，使得reponse的生命周期与request所绑定， 进而造成了内存泄露。

让我们来看看如何使用 `trackref` 工具来发现哪一个是有问题的spider(当然是在不知道任何的前提的情况下)。

当crawler运行了一小阵子后，我们发现内存占用增长了很多。 这时候我们进入telnet终端，查看活跃(live)的引用:

```
>>> prefs()
Live References

SomenastySpider                     1   oldest: 15s ago
HtmlResponse                     3890   oldest: 265s ago
Selector                            2   oldest: 0s ago
Request                          3878   oldest: 250s ago
```

上面具有非常多的活跃(且运行时间很长)的response，而其比Request的时间还要长的现象肯定是有问题的。 因此，查看最老的response:

```
>>> from scrapy.utils.trackref import get_oldest
>>> r = get_oldest('HtmlResponse')
>>> r.url
'http://www.somenastyspider.com/product.php?pid=123'
```

就这样，通过查看最老的response的URL，我们发现其属于 somenastyspider.com spider。 现在我们可以查看该spider的代码并发现导致泄露的那行代码(在request中传递response的引用)。

如果您想要遍历所有而不是最老的对象，您可以使用 **iter_all()** 方法:

```
>>> from scrapy.utils.trackref import iter_all
>>> [r.url for r in iter_all('HtmlResponse')]
['http://www.somenastyspider.com/product.php?pid=123',
 'http://www.somenastyspider.com/product.php?pid=584',
...
```

很多spider?

如果您的项目有很多的spider，prefs() 的输出会变得很难阅读。针对于此， 该方法具有 ignore 参数，用于忽略特定的类(及其子类)。例如:

```
>>> from scrapy.spider import Spider
>>> prefs(ignore=Spider)
```

将不会展现任何spider的活跃引用。

scrapy.utils.trackref模块

以下是 **trackref** 模块中可用的方法。

class scrapy.utils.trackref.**object_ref**

如果您想通过 trackref 模块追踪活跃的实例，继承该类(而不是对象)。

scrapy.utils.trackref.**print_live_refs**(*class_name*, *ignore=NoneType*)

打印活跃引用的报告，以类名分类。

参数： **ignore** (*类或者类的元组*) – 如果给定，所有指定类(或者类的元组)的对象将会被忽略。

scrapy.utils.trackref.**get_oldest**(*class_name*)

返回给定类名的最老活跃(alive)对象，如果没有则返回 None 。首先使用 **print_live_refs()** 来获取每个类所跟踪的所有活跃(live)对象的列表。

scrapy.utils.trackref.**iter_all**(*class_name*)

返回一个能给定类名的所有活跃对象的迭代器，如果没有则返回 None 。首先使用 **print_live_refs()** 来获取每个类所跟踪的所有活跃(live)对象的列表。

使用Guppy调试内存泄露

trackref 提供了追踪内存泄露非常方便的机制，其仅仅追踪了比较可能导致内存泄露的对象 (Requests, Response, Items及Selectors)。然而，内存泄露也有可能来自其他(更为隐蔽的)对象。 如果是因为这个原因，通过 trackref 则无法找到泄露点，您仍然有其他工具: Guppy library [http://pypi.python.org/pypi/guppy] 。

如果使用 setuptools , 您可以通过下列命令安装Guppy:

```
easy_install guppy
```

telnet终端也提供了快捷方式(hpy)来访问Guppy堆对象(heap objects)。 下面给出了查看堆中所有可用的Python对象的例子:

```
>>> x = hpy.heap()
>>> x.bytype
Partition of a set of 297033 objects. Total size = 52587824 bytes.
 Index  Count   %     Size   % Cumulative  % Type
     0  22307   8 16423880  31  16423880  31 dict
     1 122285  41 12441544  24  28865424  55 str
     2  68346  23  5966696  11  34832120  66 tuple
     3    227   0  5836528  11  40668648  77 unicode
```

```
     4   2461   1  2222272   4  42890920  82 type
     5  16870   6  2024400   4  44915320  85 function
     6  13949   5  1673880   3  46589200  89 types.CodeType
     7  13422   5  1653104   3  48242304  92 list
     8   3735   1  1173680   2  49415984  94 _sre.SRE_Pattern
     9   1209   0   456936   1  49872920  95 scrapy.http.headers.Headers
<1676 more rows. Type e.g. '_.more' to view.>
```

您可以看到大部分的空间被字典所使用。接着，如果您想要查看哪些属性引用了这些字典， 您可以:

```
>>> x.bytype[0].byvia
Partition of a set of 22307 objects. Total size = 16423880 bytes.
 Index  Count   %     Size   % Cumulative  % Referred Via:
     0  10982  49  9416336  57   9416336  57 '.__dict__'
     1   1820   8  2681504  16  12097840  74 '.__dict__', '.func_globals'
     2   3097  14  1122904   7  13220744  80
     3    990   4   277200   2  13497944  82 "['cookies']"
     4    987   4   276360   2  13774304  84 "['cache']"
     5    985   4   275800   2  14050104  86 "['meta']"
     6    897   4   251160   2  14301264  87 '[2]'
     7      1   0   196888   1  14498152  88 "['moduleDict']", "['modules']"
     8    672   3   188160   1  14686312  89 "['cb_kwargs']"
     9     27   0   155016   1  14841328  90 '[1]'
<333 more rows. Type e.g. '_.more' to view.>
```

如上所示，Guppy模块十分强大，不过也需要一些关于Python内部的知识。关于Guppy的更多内容请参考 Guppy documentation [http://guppy-pe.sourceforge.net/].

Leaks without leaks

有时候，您可能会注意到Scrapy进程的内存占用只在增长，从不下降。不幸的是， 有时候这并不是Scrapy或者您的项目在泄露内存。这是由于一个已知(但不有名)的Python问题。 Python在某些情况下可能不会返回已经释放的内存到操作系统。关于这个问题的更多内容请看:

- Python Memory Management [http://evanjones.ca/python-memory.html]
- Python Memory Management Part 2 [http://evanjones.ca/python-memory-part2.html]
- Python Memory Management Part 3 [http://evanjones.ca/python-memory-part3.html]

改进方案由Evan Jones提出，在 这篇文章 [http://evanjones.ca/memoryallocator/] 中详细介绍，在Python 2.5中合并。 不过这仅仅减小了这个问题，并没有完全修复。引用这片文章:

> *不幸的是，这个patch仅仅会释放没有在其内部分配对象的区域(arena)。这意味着 碎片化是一个大问题。某个应用可以拥有很多空闲内存，分布在所有的区域(arena)中， 但是没法释放任何一个。这个问题存在于所有内存分配器中。解决这个问题的唯一办法是 转化到一个更为紧凑(compact)的垃圾回收器，其能在内存中移动对象。 这需要对Python解析器做一个显著的修改。*

这个问题将会在未来Scrapy发布版本中得到解决。我们打算转化到一个新的进程模型， 并在可回收的子进程池中运行spider。

讨论

下载项目图片

Scrapy提供了一个 *item pipeline* ，来下载属于某个特定项目的图片，比如，当你抓取产品时，也想把它们的图片下载到本地。

这条管道，被称作图片管道，在 **ImagesPipeline** 类中实现，提供了一个方便并具有额外特性的方法，来下载并本地存储图片:

- 将所有下载的图片转换成通用的格式（JPG）和模式（RGB）
- 避免重新下载最近已经下载过的图片
- 缩略图生成
- 检测图像的宽/高，确保它们满足最小限制

这个管道也会为那些当前安排好要下载的图片保留一个内部队列，并将那些到达的包含相同图片的项目连接到那个队列中。 这可以避免多次下载几个项目共享的同一个图片。

Pillow [https://github.com/python-imaging/Pillow] 是用来生成缩略图，并将图片归一化为JPEG/RGB格式，因此为了使用图片管道，你需要安装这个库。 Python Imaging Library [http://www.pythonware.com/products/pil/] (PIL) 在大多数情况下是有效的，但众所周知，在一些设置里会出现问题，因此我们推荐使用 Pillow [https://github.com/python-imaging/Pillow] 而不是 PIL.

使用图片管道

当使用 **ImagesPipeline** ，典型的工作流程如下所示:

1. 在一个爬虫里，你抓取一个项目，把其中图片的URL放入 `image_urls` 组内。
2. 项目从爬虫内返回，进入项目管道。
3. 当项目进入 **ImagesPipeline**，`image_urls` 组内的URLs将被Scrapy的调度器和下载器（这意味着调度器和下载器的中间件可以复用）安排下载，当优先级更高，会在其他页面被抓取前处理。项目会在这个特定的管道阶段保持“locker”的状态，直到完成图片的下载（或者由于某些原因未完成下载）。
4. 当图片下载完，另一个组(`images`)将被更新到结构中。这个组将包含一个字典列表，其中包括下载图片的信息，比如下载路径、源抓取地址（从 `image_urls` 组获得）和图片的校验码。 `images` 列表中的图片顺序将和源 `image_urls` 组保持一致。如果某个图片下载失败，将会记录下错误信息，图片也不会出现在 `images` 组中。

使用样例

为了使用图片管道，你仅需要 *启动它* 并用 `image_urls` 和 `images` 定义一个项目:

```
import scrapy

class MyItem(scrapy.Item):

    # ... other item fields ...
    image_urls = scrapy.Field()
    images = scrapy.Field()
```

如果你需要更加复杂的功能，想重写定制图片管道行为，参见 *实现定制图片管道* 。

开启你的图片管道

为了开启你的图片管道，你首先需要在项目中添加它 **ITEM_PIPELINES** setting:

```
ITEM_PIPELINES = {'scrapy.contrib.pipeline.images.ImagesPipeline': 1}
```

并将 **IMAGES_STORE** 设置为一个有效的文件夹，用来存储下载的图片。否则管道将保持禁用状态，即使你在 **ITEM_PIPELINES** 设置中添加了它。

比如:

```
IMAGES_STORE = '/path/to/valid/dir'
```

图片存储

文件系统是当前官方唯一支持的存储系统，但也支持（非公开的） Amazon S3 [https://s3.amazonaws.com/] 。

文件系统存储

图片存储在文件中（一个图片一个文件），并使用它们URL的 SHA1 hash [http://en.wikipedia.org/wiki/SHA_hash_functions] 作为文件名。

比如，对下面的图片URL:

```
http://www.example.com/image.jpg
```

它的 *SHA1 hash* 值为:

```
3afec3b4765f8f0a07b78f98c07b83f013567a0a
```

将被下载并存为下面的文件:

```
<IMAGES_STORE>/full/3afec3b4765f8f0a07b78f98c07b83f013567a0a.jpg
```

其中:

- `<IMAGES_STORE>` 是定义在 **IMAGES_STORE** 设置里的文件夹
- `full` 是用来区分图片和缩略图（如果使用的话）的一个子文件夹。详情参见 *缩略图生成*.

额外的特性

图片失效

图像管道避免下载最近已经下载的图片。使用 **IMAGES_EXPIRES** 设置可以调整失效期限，可以用天数来指定:

```
# 90天的图片失效期限
IMAGES_EXPIRES = 90
```

缩略图生成

图片管道可以自动创建下载图片的缩略图。

为了使用这个特性，你需要设置 **IMAGES_THUMBS** 字典，其关键字为缩略图名字，值为它们的大小尺寸。

比如:

```
IMAGES_THUMBS = {
    'small': (50, 50),
    'big': (270, 270),
}
```

当你使用这个特性时，图片管道将使用下面的格式来创建各个特定尺寸的缩略图:

```
<IMAGES_STORE>/thumbs/<size_name>/<image_id>.jpg
```

其中:

- `<size_name>` 是 **IMAGES_THUMBS** 字典关键字（`small`， `big`，等）
- `<image_id>` 是图像url的 SHA1 hash [http://en.wikipedia.org/wiki/SHA_hash_functions]

例如使用 `small` 和 `big` 缩略图名字的图片文件:

```
<IMAGES_STORE>/full/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/small/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/big/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
```

第一个是从网站下载的完整图片。

滤出小图片

你可以丢掉那些过小的图片，只需在:setting:*IMAGES_MIN_HEIGHT* 和 **IMAGES_MIN_WIDTH** 设置中指定最小允许的尺寸。

比如:

```
IMAGES_MIN_HEIGHT = 110
IMAGES_MIN_WIDTH = 110
```

注意：这些尺寸一点也不影响缩略图的生成。

默认情况下，没有尺寸限制，因此所有图片都将处理。

实现定制图片管道

下面是你可以在定制的图片管道里重写的方法：

class `scrapy.contrib.pipeline.images.`**`ImagesPipeline`**

`get_media_requests`(*item*, *info*)

在工作流程中可以看到，管道会得到图片的URL并从项目中下载。为了这么做，你需要重写 **get_media_requests()** 方法，并对各个图片URL返回一个Request:

```
def get_media_requests(self, item, info):
    for image_url in item['image_urls']:
        yield scrapy.Request(image_url)
```

这些请求将被管道处理，当它们完成下载后，结果将以2-元素的元组列表形式传送到 item_completed() 方法:

- success 是一个布尔值，当图片成功下载时为 True ，因为某个原因下载失败为``False``
- image_info_or_error 是一个包含下列关键字的字典（如果成功为 True ）或者出问题时为 Twisted Failure [http://twistedmatrix.com/documents/current/api/twisted.python.failure.Failure.html] 。
  - url - 图片下载的url。这是从 get_media_requests() 方法返回请求的url。
  - path - 图片存储的路径（类似 IMAGES_STORE）
  - checksum - 图片内容的 MD5 hash [http://en.wikipedia.org/wiki/MD5]

item_completed() 接收的元组列表需要保证与 get_media_requests() 方法返回请求的顺序相一致。下面是 results 参数的一个典型值:

```
[(True,
  {'checksum': '2b00042f7481c7b056c4b410d28f33cf',
   'path': 'full/7d97e98f8af710c7e7fe703abc8f639e0ee507c4.jpg',
   'url': 'http://www.example.com/images/product1.jpg'}),
 (True,
  {'checksum': 'b9628c4ab9b595f72f280b90c4fd093d',
   'path': 'full/1ca5879492b8fd606df1964ea3c1e2f4520f076f.jpg',
   'url': 'http://www.example.com/images/product2.jpg'}),
 (False,
  Failure(...))]
```

默认 get_media_requests() 方法返回 None ，这意味着项目中没有图片可下载。

item_completed(*results*, *items*, *info*)

当一个单独项目中的所有图片请求完成时（要么完成下载，要么因为某种原因下载失败）， ImagesPipeline.item_completed() 方法将被调用。

> item_completed() 方法需要返回一个输出，其将被送到随后的项目管道阶段，因此你需要返回（或者丢弃）项目，如你在任意管道里所做的一样。

这里是一个 item_completed() 方法的例子，其中我们将下载的图片路径（传入到results中）存储到 image_paths 项目组中，如果其中没有图片，我们将丢弃项目:

```
from scrapy.exceptions import DropItem

def item_completed(self, results, item, info):
    image_paths = [x['path'] for ok, x in results if ok]
    if not image_paths:
        raise DropItem("Item contains no images")
    item['image_paths'] = image_paths
    return item
```

默认情况下， item_completed() 方法返回项目。

定制图片管道的例子

下面是一个图片管道的完整例子，其方法如上所示:

```
import scrapy
from scrapy.contrib.pipeline.images import ImagesPipeline
from scrapy.exceptions import DropItem

class MyImagesPipeline(ImagesPipeline):

    def get_media_requests(self, item, info):
        for image_url in item['image_urls']:
            yield scrapy.Request(image_url)

    def item_completed(self, results, item, info):
        image_paths = [x['path'] for ok, x in results if ok]
        if not image_paths:
            raise DropItem("Item contains no images")
        item['image_paths'] = image_paths
        return item
```

讨论

Ubuntu 软件包

0.10 新版功能.

Scrapinghub [http://scrapinghub.com/] 发布的apt-get可获取版本通常比Ubuntu里更新，并且在比 Github 仓库 [https://github.com/scrapy/scrapy] (master & stable branches) 稳定的同时还包括了最新的漏洞修复。

用法:

1. 把Scrapy签名的GPG密钥添加到APT的钥匙环中:

   ```
   sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 627220E7
   ```

2. 执行如下命令，创建 */etc/apt/sources.list.d/scrapy.list* 文件:

   ```
   echo 'deb http://archive.scrapy.org/ubuntu scrapy main' | sudo tee /etc/apt/sources.list.d/scrapy.list
   ```

3. 更新包列表并安装 scrapy-0.25:

   ```
   sudo apt-get update && sudo apt-get install scrapy-0.25
   ```

注解

如果你要升级Scrapy，请重复步骤3。

警告

debian官方源提供的 *python-scrapy* 是一个非常老的版本且不再获得Scrapy团队支持。

讨论

Scrapyd

Scrapyd被移动成为一个单独的项目。 其文档当前被托管在:

http://scrapyd.readthedocs.org/

讨论

自动限速(AutoThrottle)扩展

该扩展能根据Scrapy服务器及您爬取的网站的负载自动限制爬取速度。

设计目标

1. 更友好的对待网站，而不使用默认的下载延迟0。
2. 自动调整scrapy来优化下载速度，使得用户不用调节下载延迟及并发请求数来找到优化的值。 用户只需指定允许的最大并发请求数，剩下的都交给扩展来完成。

扩展是如何实现的

在Scrapy中，下载延迟是通过计算建立TCP连接到接收到HTTP包头(header)之间的时间来测量的。

注意，由于Scrapy可能在忙着处理spider的回调函数或者无法下载，因此在合作的多任务环境下准确测量这些延迟是十分苦难的。 不过，这些延迟仍然是对Scrapy(甚至是服务器)繁忙程度的合理测量，而这扩展就是以此为前提进行编写的。

限速算法

算法根据以下规则调整下载延迟及并发数:

1. spider永远以1并发请求数及 AUTOTHROTTLE_START_DELAY 中指定的下载延迟启动。
2. 当接收到回复时，下载延迟会调整到该回复的延迟与之前下载延迟之间的平均值。

注解

AutoThrottle扩展尊重标准Scrapy设置中的并发数及延迟。这意味着其永远不会设置一个比 DOWNLOAD_DELAY 更低的下载延迟或者比 CONCURRENT_REQUESTS_PER_DOMAIN 更高的并发数 (或 CONCURRENT_REQUESTS_PER_IP ，取决于您使用哪一个)。

设置

下面是控制AutoThrottle扩展的设置:

- AUTOTHROTTLE_ENABLED
- AUTOTHROTTLE_START_DELAY
- AUTOTHROTTLE_MAX_DELAY
- AUTOTHROTTLE_DEBUG
- CONCURRENT_REQUESTS_PER_DOMAIN
- CONCURRENT_REQUESTS_PER_IP
- DOWNLOAD_DELAY

更多内容请参考 *限速算法* 。

AUTOTHROTTLE_ENABLED

默认: `False`

启用AutoThrottle扩展。

AUTOTHROTTLE_START_DELAY

默认: `5.0`

初始下载延迟(单位:秒)。

AUTOTHROTTLE_MAX_DELAY

默认: `60.0`

在高延迟情况下最大的下载延迟(单位秒)。

AUTOTHROTTLE_DEBUG

默认: `False`

起用AutoThrottle调试(debug)模式，展示每个接收到的response。 您可以通过此来查看限速参数是如何实时被调整的。

讨论

Benchmarking

0.17 新版功能.

Scrapy提供了一个简单的性能测试工具。其创建了一个本地HTTP服务器，并以最大可能的速度进行爬取。 该测试性能工具目的是测试Scrapy在您的硬件上的效率，来获得一个基本的底线用于对比。 其使用了一个简单的spider，仅跟进链接，不做任何处理。

运行:

```
scrapy bench
```

您能看到类似的输出:

```
2013-05-16 13:08:46-0300 [scrapy] INFO: Scrapy 0.17.0 started (bot: scrapybot)
2013-05-16 13:08:47-0300 [follow] INFO: Spider opened
2013-05-16 13:08:47-0300 [follow] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:48-0300 [follow] INFO: Crawled 74 pages (at 4440 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:49-0300 [follow] INFO: Crawled 143 pages (at 4140 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:50-0300 [follow] INFO: Crawled 210 pages (at 4020 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:51-0300 [follow] INFO: Crawled 274 pages (at 3840 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:52-0300 [follow] INFO: Crawled 343 pages (at 4140 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:53-0300 [follow] INFO: Crawled 410 pages (at 4020 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:54-0300 [follow] INFO: Crawled 474 pages (at 3840 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:55-0300 [follow] INFO: Crawled 538 pages (at 3840 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:56-0300 [follow] INFO: Crawled 602 pages (at 3840 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:57-0300 [follow] INFO: Closing spider (closespider_timeout)
2013-05-16 13:08:57-0300 [follow] INFO: Crawled 666 pages (at 3840 pages/min), scraped 0 items (at 0 items/min)
2013-05-16 13:08:57-0300 [follow] INFO: Dumping Scrapy stats:
    {'downloader/request_bytes': 231508,
     'downloader/request_count': 682,
     'downloader/request_method_count/GET': 682,
     'downloader/response_bytes': 1172802,
     'downloader/response_count': 682,
     'downloader/response_status_count/200': 682,
     'finish_reason': 'closespider_timeout',
     'finish_time': datetime.datetime(2013, 5, 16, 16, 8, 57, 985539),
     'log_count/INFO': 14,
     'request_depth_max': 34,
     'response_received_count': 682,
     'scheduler/dequeued': 682,
     'scheduler/dequeued/memory': 682,
     'scheduler/enqueued': 12767,
     'scheduler/enqueued/memory': 12767,
     'start_time': datetime.datetime(2013, 5, 16, 16, 8, 47, 676539)}
2013-05-16 13:08:57-0300 [follow] INFO: Spider closed (closespider_timeout)
```

这说明了您的Scrapy能以3900页面/分钟的速度爬取。注意，这是一个非常简单，仅跟进链接的spider。 任何您所编写的spider会做更多处理，从而减慢爬取的速度。 减慢的程度取决于spider做的处理以及其是如何被编写的。

未来会有更多的用例会被加入到性能测试套装中，以覆盖更多常见的情景。

讨论

Jobs: 暂停，恢复爬虫

有些情况下，例如爬取大的站点，我们希望能暂停爬取，之后再恢复运行。

Scrapy通过如下工具支持这个功能:

- 一个把调度请求保存在磁盘的调度器
- 一个把访问请求保存在磁盘的副本过滤器[duplicates filter]
- 一个能持续保持爬虫状态(键/值对)的扩展

Job 路径

要启用持久化支持，你只需要通过 `JOBDIR` 设置 *job directory* 选项。这个路径将会存储 所有的请求数据来保持一个单独任务的状态(例如：一次spider爬取(a spider run))。必须要注意的是，这个目录不允许被不同的spider 共享，甚至是同一个spider的不同jobs/runs也不行。也就是说，这个目录就是存储一个 *单独* job的状态信息。

怎么使用

要启用一个爬虫的持久化，运行以下命令:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

然后，你就能在任何时候安全地停止爬虫(按Ctrl-C或者发送一个信号)。恢复这个爬虫也是同样的命令:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

保持状态

有的时候，你希望持续保持一些运行长时间的蜘蛛的状态。这时您可以使用 `spider.state` 属性, 该属性的类型必须是dict. scrapy提供了内置扩展负责在spider启动或结束时，从工作路径(job directory)中序列化、存储、加载属性。

下面这个例子展示了使用spider state的回调函数(callback)(简洁起见，省略了其他的代码):

```
def parse_item(self, response):
    # parse item here
    self.state['items_count'] = self.state.get('items_count', 0) + 1
```

持久化的一些坑

如果你想要使用Scrapy的持久化支持,还有一些东西您需要了解:

Cookies的有效期

Cookies是有有效期的(可能过期)。所以如果你没有把你的爬虫及时恢复，那么他可能在被调度回去的时候 就不能工作了。当然如果你的爬虫不依赖cookies就不会有这个问题了。

请求序列化

请求是由 *pickle* 进行序列化的，所以你需要确保你的请求是可被pickle序列化的。 这里最常见的问题是在在request回调函数中使用 `lambda` 方法，导致无法序列化。

例如, 这样就会有问题:

```
def some_callback(self, response):
    somearg = 'test'
    return scrapy.Request('http://www.example.com', callback=lambda r: self.other_callback(r, somearg))

def other_callback(self, response, somearg):
    print "the argument passed is:", somearg
```

这样才对:

```
def some_callback(self, response):
    somearg = 'test'
    return scrapy.Request('http://www.example.com', meta={'somearg': somearg})

#这里的实例代码有错，应该是(译者注)
#    return scrapy.Request('http://www.example.com', meta={'somearg': somearg}, callback=self.other_callback)

def other_callback(self, response):
    somearg = response.meta['somearg']
    print "the argument passed is:", somearg
```

讨论

DjangoItem

DjangoItem 是一个item的类，其从Django模型中获取字段(field)定义。 您可以简单地创建一个 **DjangoItem** 并指定其关联的Django模型。

除了获得您item中定义的字段外， **DjangoItem** 提供了创建并获得一个具有item数据的Django模型实例(Django model instance)的方法。

使用DjangoItem

DjangoItem 使用方法与Django中的ModelForms类似。您创建一个子类， 并定义其 `django_model` 属性。这样，您就可以得到一个字段与Django模型字段(model field)一一对应的item了。

另外，您可以定义模型中没有的字段，甚至是覆盖模型中已经定义的字段。

让我们来看个例子:

创造一个Django模型:

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=255)
    age = models.IntegerField()
```

定义一个基本的 **DjangoItem**:

```
from scrapy.contrib.djangoitem import DjangoItem

class PersonItem(DjangoItem):
    django_model = Person
```

DjangoItem 的使用方法和 Item 类似:

```
>>> p = PersonItem()
>>> p['name'] = 'John'
>>> p['age'] = '22'
```

要从item中获取Django模型，调用 **DjangoItem** 中额外的方法 **save()**:

```
>>> person = p.save()
>>> person.name
'John'
>>> person.age
'22'
>>> person.id
1
```

当我们调用 **save()** 时，模型已经保存了。我们可以在调用时带上 `commit=False` 来避免保存， 并获取到一个未保存的模型:

```
>>> person = p.save(commit=False)
>>> person.name
'John'
>>> person.age
'22'
>>> person.id
None
```

正如之前所说的，我们可以在item中加入字段:

```
import scrapy
from scrapy.contrib.djangoitem import DjangoItem

class PersonItem(DjangoItem):
    django_model = Person
    sex = scrapy.Field()
```

```
>>> p = PersonItem()
>>> p['name'] = 'John'
>>> p['age'] = '22'
>>> p['sex'] = 'M'
```

注解

当执行 **save()** 时添加到item的字段不会有作用(taken into account)。

并且我们可以覆盖模型中的字段:

```
class PersonItem(DjangoItem):
```

```
    django_model = Person
    name = scrapy.Field(default='No Name')
```

这在提供字段属性时十分有用，例如您项目中使用的默认或者其他属性一样。

DjangoItem注意事项

DjangoItem提供了在Scrapy项目中集成DjangoItem的简便方法，不过需要注意的是， 如果在Scrapy中爬取大量(百万级)的item时，Django ORM扩展得并不是很好(not scale well)。 这是因为关系型后端对于一个密集型(intensive)应用(例如web爬虫)并不是一个很好的选择， 尤其是具有大量的索引的数据库。

配置Django的设置

在Django应用之外使用Django模型(model)，您需要设置 `DJANGO_SETTINGS_MODULE` 环境变量以及 –大多数情况下– 修改 `PYTHONPATH` 环境变量来导入设置模块。

完成这个配置有很多方法，具体选择取决您的情况及偏好。 下面详细给出了完成这个配置的最简单方法。

假设您项目的名称为 `mysite` ，位于 `/home/projects/mysite` 且用 `Person` 模型创建了一个应用 `myapp` 。 这意味着您的目录结构类似于:

```
/home/projects/mysite
├── manage.py
├── myapp
│   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

接着您需要将 `/home/projects/mysite` 加入到 `PYTHONPATH` 环境变量中并将 `mysite.settings` 设置为 `DJANGO_SETTINGS_MODULE` 环境变量。这可以在Scrapy设置文件中添加下列代码:

```
import sys
sys.path.append('/home/projects/mysite')

import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'
```

注意，由于我们在python运行环境中，所以我们修改 `sys.path` 变量而不是 `PYTHONPATH` 环境变量。 如果所有设置正确，您应该可以运行 `scrapy shell` 命令并且导入 `Person` 模型(例如 `from myapp.models import Person`)。

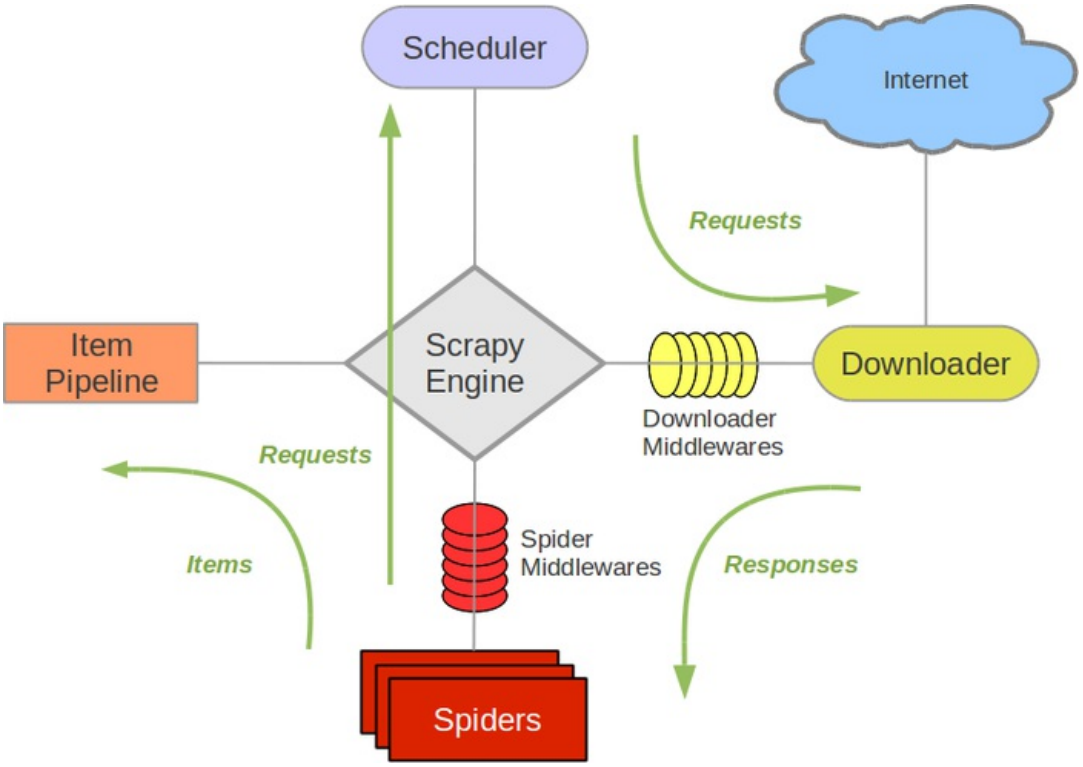
讨论

架构概览

本文档介绍了Scrapy架构及其组件之间的交互。

概述

接下来的图表展现了Scrapy的架构，包括组件及在系统中发生的数据流的概览(绿色箭头所示)。 下面对每个组件都做了简单介绍，并给出了详细内容的链接。数据流如下所描述。



组件

Scrapy Engine

引擎负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。 详细内容查看下面的数据流(Data Flow)部分。

调度器(Scheduler)

调度器从引擎接受request并将他们入队，以便之后引擎请求他们时提供给引擎。

下载器(Downloader)

下载器负责获取页面数据并提供给引擎，而后提供给spider。

Spiders

Spider是Scrapy用户编写用于分析response并提取item(即获取到的item)或额外跟进的URL的类。 每个spider负责处理一个特定(或一些)网站。 更多内容请看 *Spiders* 。

Item Pipeline

Item Pipeline负责处理被spider提取出来的item。典型的处理有清理、 验证及持久化(例如存取到数据库中)。 更多内容查看 *Item Pipeline* 。

下载器中间件(Downloader middlewares)

下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理Downloader传递给引擎的response。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。更多内容请看 *下载器中间件(Downloader Middleware)* 。

Spider中间件(Spider middlewares)

Spider中间件是在引擎及Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items及requests)。 其提供了一个简

便的机制，通过插入自定义代码来扩展Scrapy功能。更多内容请看 *Spider中间件(Middleware)* 。

数据流(Data flow)

Scrapy中的数据流由执行引擎控制，其过程如下:

1. 引擎打开一个网站(open a domain)，找到处理该网站的Spider并向该spider请求第一个要爬取的URL(s)。
2. 引擎从Spider中获取到第一个要爬取的URL并在调度器(Scheduler)以Request调度。
3. 引擎向调度器请求下一个要爬取的URL。
4. 调度器返回下一个要爬取的URL给引擎，引擎将URL通过下载中间件(请求(request)方向)转发给下载器(Downloader)。
5. 一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过下载中间件(返回(response)方向)发送给引擎。
6. 引擎从下载器中接收到Response并通过Spider中间件(输入方向)发送给Spider处理。
7. Spider处理Response并返回爬取到的Item及(跟进的)新的Request给引擎。
8. 引擎将(Spider返回的)爬取到的Item给Item Pipeline，将(Spider返回的)Request给调度器。
9. (从第二步)重复直到调度器中没有更多地request，引擎关闭该网站。

事件驱动网络(Event-driven networking)

Scrapy基于事件驱动网络框架 Twisted [http://twistedmatrix.com/trac/] 编写。因此，Scrapy基于并发性考虑由非阻塞(即异步)的实现。

关于异步编程及Twisted更多的内容请查看下列链接:

- Introduction to Deferreds in Twisted [http://twistedmatrix.com/documents/current/core/howto/defer-intro.html]
- Twisted - hello, asynchronous programming [http://jessenoller.com/2009/02/11/twisted-hello-asynchronous-programming/]

讨论

下载器中间件(Downloader Middleware)

下载器中间件是介于Scrapy的request/response处理的钩子框架。 是用于全局修改Scrapy request和response的一个轻量、底层的系统。

激活下载器中间件

要激活下载器中间件组件，将其加入到 DOWNLOADER_MIDDLEWARES 设置中。 该设置是一个字典(dict)，键为中间件类的路径，值为其中间件的顺序(order)。

这里是一个例子:

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
}
```

DOWNLOADER_MIDDLEWARES 设置会与Scrapy定义的 DOWNLOADER_MIDDLEWARES_BASE 设置合并(但不是覆盖)， 而后根据顺序(order)进行排序，最后得到启用中间件的有序列表: 第一个中间件是最靠近引擎的，最后一个中间件是最靠近下载器的。

关于如何分配中间件的顺序请查看 DOWNLOADER_MIDDLEWARES_BASE 设置，而后根据您想要放置中间件的位置选择一个值。 由于每个中间件执行不同的动作，您的中间件可能会依赖于之前(或者之后)执行的中间件，因此顺序是很重要的。

如果您想禁止内置的(在 DOWNLOADER_MIDDLEWARES_BASE 中设置并默认启用的)中间件， 您必须在项目的 DOWNLOADER_MIDDLEWARES 设置中定义该中间件，并将其值赋为 *None* 。 例如，如果您想要关闭user-agent中间件:

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
}
```

最后，请注意，有些中间件需要通过特定的设置来启用。更多内容请查看相关中间件文档。

编写您自己的下载器中间件

编写下载器中间件十分简单。每个中间件组件是一个定义了以下一个或多个方法的Python类:

class `scrapy.contrib.downloadermiddleware.`**`DownloaderMiddleware`**

`process_request`(*request*, *spider*)

当每个request通过下载中间件时，该方法被调用。

process_request() 必须返回其中之一: 返回 None 、返回一个 Response 对象、返回一个 Request 对象或raise IgnoreRequest 。

如果其返回 None ，Scrapy将继续处理该request，执行其他的中间件的相应方法，直到合适的下载器处理函数(download handler)被调用， 该request被执行(其response被下载)。

如果其返回 Response 对象，Scrapy将不会调用 *任何* 其他的 process_request() 或 process_exception() 方法，或相应地下载函数； 其将返回该response。 已安装的中间件的 process_response() 方法则会在每个response返回时被调用。

如果其返回 Request 对象，Scrapy则停止调用 process_request方法并重新调度返回的request。当新返回的request被执行后， 相应地中间件链将会根据下载的response被调用。

如果其raise一个 IgnoreRequest 异常，则安装的下载中间件的 process_exception() 方法会被调用。如果没有任何一个方法处理该异常， 则request的errback(Request.errback)方法会被调用。如果没有代码处理抛出的异常， 则该异常被忽略且不记录(不同于其他异常那样)。

参数:
- **request** (Request 对象) – 处理的request
- **spider** (Spider 对象) – 该request对应的spider

`process_response`(*request*, *response*, *spider*)

process_request() 必须返回以下之一: 返回一个 Response 对象、 返回一个 Request 对象或raise一个 IgnoreRequest 异常。

如果其返回一个 Response (可以与传入的response相同，也可以是全新的对象)， 该response会被在链中的其他中间件的 process_response() 方法处理。

如果其返回一个 Request 对象，则中间件链停止， 返回的request会被重新调度下载。处理类似于 process_request() 返回request所做的那样。

如果其抛出一个 IgnoreRequest 异常，则调用request的errback(Request.errback)。 如果没有代码处理抛出的异常，则该异常被忽略且不记录(不同于其他异常那样)。

参数:
- **request** (Request 对象) – response所对应的request

- **response** (Response 对象) – 被处理的response
- **spider** (Spider 对象) – response所对应的spider

process_exception(*request*, *exception*, *spider*)

当下载处理器(download handler)或 process_request() (下载中间件)抛出异常(包括 IgnoreRequest 异常)时， Scrapy调用 process_exception() 。

process_exception() 应该返回以下之一: 返回 None 、 一个 Response 对象、或者一个 Request 对象。

如果其返回 None ，Scrapy将会继续处理该异常，接着调用已安装的其他中间件的 process_exception() 方法，直到所有中间件都被调用完毕，则调用默认的异常处理。

如果其返回一个 Response 对象，则已安装的中间件链的 process_response() 方法被调用。Scrapy将不会调用任何其他中间件的 process_exception() 方法。

如果其返回一个 Request 对象， 则返回的request将会被重新调用下载。这将停止中间件的 process_exception() 方法执行，就如返回一个response的那样。

参数:
- **request** (是 Request 对象) – 产生异常的request
- **exception** (Exception 对象) – 抛出的异常
- **spider** (Spider 对象) – request对应的spider

内置下载中间件参考手册

本页面介绍了Scrapy自带的所有下载中间件。关于如何使用及编写您自己的中间件，请参考 *downloader middleware usage guide*.

关于默认启用的中间件列表(及其顺序)请参考 DOWNLOADER_MIDDLEWARES_BASE 设置。

CookiesMiddleware

class scrapy.contrib.downloadermiddleware.cookies.**CookiesMiddleware**

该中间件使得爬取需要cookie(例如使用session)的网站成为了可能。 其追踪了web server发送的cookie，并在之后的request中发送回去， 就如浏览器所做的那样。

以下设置可以用来配置cookie中间件:

- COOKIES_ENABLED
- COOKIES_DEBUG

单spider多cookie session

0.15 新版功能.

Scrapy通过使用 cookiejar Request meta key来支持单spider追踪多cookie session。 默认情况下其使用一个cookie jar(session)，不过您可以传递一个标示符来使用多个。

例如:

```
for i, url in enumerate(urls):
    yield scrapy.Request("http://www.example.com", meta={'cookiejar': i},
        callback=self.parse_page)
```

需要注意的是 cookiejar meta key不是”黏性的(sticky)”。 您需要在之后的request请求中接着传递。例如:

```
def parse_page(self, response):
    # do some processing
    return scrapy.Request("http://www.example.com/otherpage",
        meta={'cookiejar': response.meta['cookiejar']},
        callback=self.parse_other_page)
```

COOKIES_ENABLED

默认: True

是否启用cookies middleware。如果关闭，cookies将不会发送给web server。

COOKIES_DEBUG

默认: False

如果启用，Scrapy将记录所有在request(Cookie 请求头)发送的cookies及response接收到的cookies(Set-Cookie 接收头)。

下边是启用 COOKIES_DEBUG 的记录的样例:

```
2011-04-06 14:35:10-0300 [diningcity] INFO: Spider opened
```

```
2011-04-06 14:35:10-0300 [diningcity] DEBUG: Sending cookies to: <GET http://www.diningcity.com/netherlands/index.html>
        Cookie: clientlanguage_nl=en_EN
2011-04-06 14:35:14-0300 [diningcity] DEBUG: Received cookies from: <200 http://www.diningcity.com/netherlands/index.html
        Set-Cookie: JSESSIONID=B~FA4DC0C496C8762AE4F1A620EAB34F38; Path=/
        Set-Cookie: ip_isocode=US
        Set-Cookie: clientlanguage_nl=en_EN; Expires=Thu, 07-Apr-2011 21:21:34 GMT; Path=/
2011-04-06 14:49:50-0300 [diningcity] DEBUG: Crawled (200) <GET http://www.diningcity.com/netherlands/index.html> (refere
[...]
```

DefaultHeadersMiddleware

class `scrapy.contrib.downloadermiddleware.defaultheaders.`**`DefaultHeadersMiddleware`**

该中间件设置 **`DEFAULT_REQUEST_HEADERS`** 指定的默认request header。

DownloadTimeoutMiddleware

class `scrapy.contrib.downloadermiddleware.downloadtimeout.`**`DownloadTimeoutMiddleware`**

该中间件设置 **`DOWNLOAD_TIMEOUT`** 或 spider的 **`download_timeout`** 属性指定的request下载超时时间.

注解

您也可以使用 **`download_timeout`** Request.meta key 来对每个请求设置下载超时时间. 这种方式在 DownloadTimeoutMiddleware 被关闭时仍然有效.

HttpAuthMiddleware

class `scrapy.contrib.downloadermiddleware.httpauth.`**`HttpAuthMiddleware`**

该中间件完成某些使用 Basic access authentication [http://en.wikipedia.org/wiki/Basic_access_authentication] (或者叫HTTP认证)的spider生成的请求的认证过程。

在spider中启用HTTP认证，请设置spider的 `http_user` 及 `http_pass` 属性。

样例:

```
from scrapy.contrib.spiders import CrawlSpider

class SomeIntranetSiteSpider(CrawlSpider):

    http_user = 'someuser'
    http_pass = 'somepass'
    name = 'intranet.example.com'

    # .. rest of the spider code omitted ...
```

HttpCacheMiddleware

class `scrapy.contrib.downloadermiddleware.httpcache.`**`HttpCacheMiddleware`**

该中间件为所有HTTP request及response提供了底层(low-level)缓存支持。 其由cache存储后端及cache策略组成。

Scrapy提供了两种HTTP缓存存储后端:

- *Filesystem storage backend (默认值)*
- *DBM storage backend*

您可以使用 **`HTTPCACHE_STORAGE`** 设定来修改HTTP缓存存储后端。 您也可以实现您自己的存储后端。

Scrapy提供了两种了缓存策略:

- *RFC2616策略*
- *Dummy策略(默认值)*

您可以使用 **`HTTPCACHE_POLICY`** 设定来修改HTTP缓存存储后端。 您也可以实现您自己的存储策略。

Dummy策略(默认值)

该策略不考虑任何HTTP Cache-Control指令。每个request及其对应的response都被缓存。 当相同的request发生时，其不发送任何数据，直接返回response。

Dummpy策略对于测试spider十分有用。其能使spider运行更快(不需要每次等待下载完成)， 同时在没有网络连接时也能测试。其目的是为了能够回放spider的运行过程， *使之与之前的运行过程一模一样* 。

使用这个策略请设置:

- **`HTTPCACHE_POLICY`** 为 `scrapy.contrib.httpcache.DummyPolicy`

RFC2616策略

该策略提供了符合RFC2616的HTTP缓存，例如符合HTTP Cache-Control， 针对生产环境并且应用在持续性运行环境所设置。该策略能避免下载未修改的数据(来节省带宽，提高爬取速度)。

实现了:

- 当 *no-store* cache-control指令设置时不存储response/request。
- 当 *no-cache* cache-control指定设置时不从cache中提取response，即使response为最新。
- 根据 *max-age* cache-control指令中计算保存时间(freshness lifetime)。
- 根据 *Expires* 指令来计算保存时间(freshness lifetime)。
- 根据response包头的 *Last-Modified* 指令来计算保存时间(freshness lifetime)(Firefox使用的启发式算法)。
- 根据response包头的 *Age* 计算当前年龄(current age)
- 根据 *Date* 计算当前年龄(current age)
- 根据response包头的 *Last-Modified* 验证老旧的response。
- 根据response包头的 *ETag* 验证老旧的response。
- 为接收到的response设置缺失的 *Date* 字段。

目前仍然缺失:

- *Pragma: no-cache* 支持 http://www.mnot.net/cache_docs/#PRAGMA
- *Vary* 字段支持 http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.6
- 当update或delete之后失效相应的response http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.10
- ... 以及其他可能缺失的特性 ..

使用这个策略，设置:

- **HTTPCACHE_POLICY** 为 `scrapy.contrib.httpcache.RFC2616Policy`

Filesystem storage backend (默认值)

文件系统存储后端可以用于HTTP缓存中间件。

使用该存储端，设置:

- **HTTPCACHE_STORAGE** 为 `scrapy.contrib.httpcache.FilesystemCacheStorage`

每个request/response组存储在不同的目录中，包含下列文件:

- `request_body` - the plain request body
- `request_headers` - the request headers (原始HTTP格式)
- `response_body` - the plain response body
- `response_headers` - the request headers (原始HTTP格式)
- `meta` - 以Python `repr()` 格式(grep-friendly格式)存储的该缓存资源的一些元数据。
- `pickled_meta` - 与 `meta` 相同的元数据，不过使用pickle来获得更高效的反序列化性能。

目录的名称与request的指纹(参考 `scrapy.utils.request.fingerprint`)有关，而二级目录是为了避免在同一文件夹下有太多文件 (这在很多文件系统中是十分低效的)。目录的例子:

```
/path/to/cache/dir/example.com/72/72811f648e718090f041317756c03adb0ada46c7
```

DBM storage backend

0.13 新版功能.

同时也有 DBM [http://en.wikipedia.org/wiki/Dbm] 存储后端可以用于HTTP缓存中间件。

默认情况下，其采用 anydbm [http://docs.python.org/library/anydbm.html] 模块，不过您也可以通过 **HTTPCACHE_DBM_MODULE** 设置进行修改。

使用该存储端，设置:

- **HTTPCACHE_STORAGE** 为 `scrapy.contrib.httpcache.DbmCacheStorage`

LevelDB storage backend

0.23 新版功能.

A LevelDB [http://code.google.com/p/leveldb/] storage backend is also available for the HTTP cache middleware.

This backend is not recommended for development because only one process can access LevelDB databases at the same time, so you can't run a crawl and open the scrapy shell in parallel for the same spider.

In order to use this storage backend:

- set `HTTPCACHE_STORAGE` to `scrapy.contrib.httpcache.LeveldbCacheStorage`
- install LevelDB python bindings [http://pypi.python.org/pypi/leveldb] like `pip install leveldb`

HTTPCache中间件设置

`HttpCacheMiddleware` 可以通过以下设置进行配置:

HTTPCACHE_ENABLED

0.11 新版功能.

默认: `False`

HTTP缓存是否开启。

在 0.11 版更改: 在0.11版本前，是使用 `HTTPCACHE_DIR` 来开启缓存。

HTTPCACHE_EXPIRATION_SECS

默认: `0`

缓存的request的超时时间，单位秒。

超过这个时间的缓存request将会被重新下载。如果为0，则缓存的request将永远不会超时。

在 0.11 版更改: 在0.11版本前，0的意义是缓存的request永远超时。

HTTPCACHE_DIR

默认: `'httpcache'`

存储(底层的)HTTP缓存的目录。如果为空，则HTTP缓存将会被关闭。 如果为相对目录，则相对于项目数据目录(project data dir)。更多内容请参考 *默认的Scrapy项目结构* 。

HTTPCACHE_IGNORE_HTTP_CODES

0.10 新版功能.

默认: `[]`

不缓存设置中的HTTP返回值(code)的request。

HTTPCACHE_IGNORE_MISSING

默认: `False`

如果启用，在缓存中没找到的request将会被忽略，不下载。

HTTPCACHE_IGNORE_SCHEMES

0.10 新版功能.

默认: `['file']`

不缓存这些URI标准(scheme)的response。

HTTPCACHE_STORAGE

默认: `'scrapy.contrib.httpcache.FilesystemCacheStorage'`

实现缓存存储后端的类。

HTTPCACHE_DBM_MODULE

0.13 新版功能.

默认: `'anydbm'`

在 *DBM存储后端* 的数据库模块。 该设定针对DBM后端。

HTTPCACHE_POLICY

0.18 新版功能.

默认: `'scrapy.contrib.httpcache.DummyPolicy'`

实现缓存策略的类。

HttpCompressionMiddleware

class `scrapy.contrib.downloadermiddleware.httpcompression.`**`HttpCompressionMiddleware`**

该中间件提供了对压缩(gzip, deflate)数据的支持。

HttpCompressionMiddleware Settings

COMPRESSION_ENABLED

默认: `True`

Compression Middleware(压缩中间件)是否开启。

ChunkedTransferMiddleware

class `scrapy.contrib.downloadermiddleware.chunked.`**`ChunkedTransferMiddleware`**

该中间件添加了对 chunked transfer encoding [http://en.wikipedia.org/wiki/Chunked_transfer_encoding] 的支持。

HttpProxyMiddleware

0.8 新版功能.

class `scrapy.contrib.downloadermiddleware.httpproxy.`**`HttpProxyMiddleware`**

该中间件提供了对request设置HTTP代理的支持。您可以通过在 **`Request`** 对象中设置 `proxy` 元数据来开启代理。

类似于Python标准库模块 urllib [http://docs.python.org/library/urllib.html] 及 urllib2 [http://docs.python.org/library/urllib2.html] ，其使用了下列环境变量:

- `http_proxy`
- `https_proxy`
- `no_proxy`

RedirectMiddleware

class `scrapy.contrib.downloadermiddleware.redirect.`**`RedirectMiddleware`**

该中间件根据response的状态处理重定向的request。

通过该中间件的(被重定向的)request的url可以通过 **`Request.meta`** 的 `redirect_urls` 键找到。

`RedirectMiddleware` 可以通过下列设置进行配置(更多内容请参考设置文档):

- **`REDIRECT_ENABLED`**
- **`REDIRECT_MAX_TIMES`**

如果 **`Request.meta`** 中 `dont_redirect` 设置为True ，则该request将会被此中间件忽略。

RedirectMiddleware settings

REDIRECT_ENABLED

0.13 新版功能.

默认: `True`

是否启用Redirect中间件。

REDIRECT_MAX_TIMES

默认: `20`

单个request被重定向的最大次数。

MetaRefreshMiddleware

class `scrapy.contrib.downloadermiddleware.redirect.`**`MetaRefreshMiddleware`**

该中间件根据meta-refresh html标签处理request重定向。

`MetaRefreshMiddleware` 可以通过以下设定进行配置 (更多内容请参考设置文档)。

- METAREFRESH_ENABLED
- METAREFRESH_MAXDELAY

该中间件遵循 RedirectMiddleware 描述的 REDIRECT_MAX_TIMES 设定，dont_redirect 及 redirect_urls meta key。

MetaRefreshMiddleware settings

METAREFRESH_ENABLED

0.17 新版功能.

默认: `True`

Meta Refresh中间件是否启用。

REDIRECT_MAX_METAREFRESH_DELAY

默认: `100`

跟进重定向的最大 meta-refresh 延迟(单位:秒)。

RetryMiddleware

class `scrapy.contrib.downloadermiddleware.retry.`**`RetryMiddleware`**

该中间件将重试可能由于临时的问题，例如连接超时或者HTTP 500错误导致失败的页面。

爬取进程会收集失败的页面并在最后，spider爬取完所有正常(不失败)的页面后重新调度。 一旦没有更多需要重试的失败页面，该中间件将会发送一个信号(retry_complete)， 其他插件可以监听该信号。

RetryMiddleware 可以通过下列设定进行配置 (更多内容请参考设置文档):

- RETRY_ENABLED
- RETRY_TIMES
- RETRY_HTTP_CODES

关于HTTP错误的考虑:

如果根据HTTP协议，您可能想要在设定 RETRY_HTTP_CODES 中移除400错误。 该错误被默认包括是由于这个代码经常被用来指示服务器过载(overload)了。而在这种情况下，我们想进行重试。

如果 Request.meta 中 `dont_retry` 设为True， 该request将会被本中间件忽略。

RetryMiddleware Settings

RETRY_ENABLED

0.13 新版功能.

默认: `True`

Retry Middleware是否启用。

RETRY_TIMES

默认: `2`

包括第一次下载，最多的重试次数

RETRY_HTTP_CODES

默认: `[500, 502, 503, 504, 400, 408]`

重试的response 返回值(code)。其他错误(DNS查找问题、连接失败及其他)则一定会进行重试。

RobotsTxtMiddleware

class `scrapy.contrib.downloadermiddleware.robotstxt.`**`RobotsTxtMiddleware`**

该中间件过滤所有robots.txt eclusion standard中禁止的request。

确认该中间件及 ROBOTSTXT_OBEY 设置被启用以确保Scrapy尊重robots.txt。

> **警告**
> 记住, 如果您在一个网站中使用了多个并发请求， Scrapy仍然可能下载一些被禁止的页面。这是由于这些页面是在robots.txt被下载前

被请求的。 这是当前robots.txt中间件已知的限制，并将在未来进行修复。

DownloaderStats

class scrapy.contrib.downloadermiddleware.stats.**DownloaderStats**

保存所有通过的request、response及exception的中间件。

您必须启用 DOWNLOADER_STATS 来启用该中间件。

UserAgentMiddleware

class scrapy.contrib.downloadermiddleware.useragent.**UserAgentMiddleware**

用于覆盖spider的默认user agent的中间件。

要使得spider能覆盖默认的user agent，其 *user_agent* 属性必须被设置。

AjaxCrawlMiddleware

class scrapy.contrib.downloadermiddleware.ajaxcrawl.**AjaxCrawlMiddleware**

根据meta-fragment html标签查找 'AJAX可爬取' 页面的中间件。查看 https://developers.google.com/webmasters/ajax-crawling/docs/getting-started 来获得更多内容。

注解

即使没有启用该中间件，Scrapy仍能查找类似于 `'http://example.com/!#foo=bar'` 这样的'AJAX可爬取'页面。AjaxCrawlMiddleware是针对不具有 `'!#'` 的URL，通常发生在'index'或者'main'页面中。

AjaxCrawlMiddleware设置

AJAXCRAWL_ENABLED

0.21 新版功能.

默认: `False`

AjaxCrawlMiddleware是否启用。您可能需要针对 *通用爬虫* 启用该中间件。

讨论

Spider中间件(Middleware)

下载器中间件是介入到Scrapy的spider处理机制的钩子框架，您可以添加代码来处理发送给 *Spiders* 的response及spider产生的item和request。

激活spider中间件

要启用spider中间件，您可以将其加入到 SPIDER_MIDDLEWARES 设置中。 该设置是一个字典，键位中间件的路径，值为中间件的顺序(order)。

样例:

```
SPIDER_MIDDLEWARES = {
    'myproject.middlewares.CustomSpiderMiddleware': 543,
}
```

SPIDER_MIDDLEWARES 设置会与Scrapy定义的 SPIDER_MIDDLEWARES_BASE 设置合并(但不是覆盖)， 而后根据顺序(order)进行排序，最后得到启用中间件的有序列表: 第一个中间件是最靠近引擎的，最后一个中间件是最靠近spider的。

关于如何分配中间件的顺序请查看 SPIDER_MIDDLEWARES_BASE 设置，而后根据您想要放置中间件的位置选择一个值。 由于每个中间件执行不同的动作，您的中间件可能会依赖于之前(或者之后)执行的中间件，因此顺序是很重要的。

如果您想禁止内置的(在 SPIDER_MIDDLEWARES_BASE 中设置并默认启用的)中间件， 您必须在项目的 SPIDER_MIDDLEWARES 设置中定义该中间件，并将其值赋为 *None* 。 例如，如果您想要关闭off-site中间件:

```
SPIDER_MIDDLEWARES = {
    'myproject.middlewares.CustomSpiderMiddleware': 543,
    'scrapy.contrib.spidermiddleware.offsite.OffsiteMiddleware': None,
}
```

最后，请注意，有些中间件需要通过特定的设置来启用。更多内容请查看相关中间件文档。

编写您自己的spider中间件

编写spider中间件十分简单。每个中间件组件是一个定义了以下一个或多个方法的Python类:

class scrapy.contrib.spidermiddleware.**SpiderMiddleware**

process_spider_input(*response*, *spider*)

当response通过spider中间件时，该方法被调用，处理该response。

process_spider_input() 应该返回 None 或者抛出一个异常。

如果其返回 None ，Scrapy将会继续处理该response，调用所有其他的中间件直到spider处理该response。

如果其跑出一个异常(exception)，Scrapy将不会调用任何其他中间件的 process_spider_input() 方法，并调用request的errback。 errback的输出将会以另一个方向被重新输入到中间件链中，使用 process_spider_output() 方法来处理，当其抛出异常时则带调用 process_spider_exception() 。

参数:
- **response** (Response 对象) – 被处理的response
- **spider** (Spider 对象) – 该response对应的spider

process_spider_output(*response*, *result*, *spider*)

当Spider处理response返回result时，该方法被调用。

process_spider_output() 必须返回包含 Request 或 Item 对象的可迭代对象(iterable)。

参数:
- **response** (Response 对象) – 生成该输出的response
- **result** (包含 Request 或 Item 对象的可迭代对象(iterable)) – spider返回的result
- **spider** (Spider 对象) – 其结果被处理的spider

process_spider_exception(*response*, *exception*, *spider*)

当spider或(其他spider中间件的) process_spider_input() 跑出异常时， 该方法被调用。

process_spider_exception() 必须要么返回 None ， 要么返回一个包含 Response 或 Item 对象的可迭代对象(iterable)。

如果其返回 None ，Scrapy将继续处理该异常，调用中间件链中的其他中间件的 process_spider_exception() 方法，直到所有中间件都被调用，该异常到达引擎(异常将被记录并被忽略)。

如果其返回一个可迭代对象，则中间件链的 process_spider_output() 方法被调用， 其他的 process_spider_exception() 将不会被调用。

参数：
- **response** (Response 对象) – 异常被抛出时被处理的response
- **exception** (Exception 对象) – 被跑出的异常
- **spider** (Spider 对象) – 抛出该异常的spider

process_start_requests(*start_requests*, *spider*)

0.15 新版功能.

该方法以spider 启动的request为参数被调用，执行的过程类似于 process_spider_output()，只不过其没有相关联的response并且必须返回request(不是item)。

其接受一个可迭代的对象(start_requests 参数)且必须返回另一个包含 Request 对象的可迭代对象。

注解

当在您的spider中间件实现该方法时， 您必须返回一个可迭代对象(类似于参数start_requests)且不要遍历所有的 start_requests。 该迭代器会很大(甚至是无限)，进而导致内存溢出。 Scrapy引擎在其具有能力处理start request时将会拉起request， 因此start request迭代器会变得无限，而由其他参数来停止spider(例如时间限制或者item/page记数)。

参数：
- **start_requests** (包含 Request 的可迭代对象) – start requests
- **spider** (Spider 对象) – start requests所属的spider

内置spider中间件参考手册

本页面介绍了Scrapy自带的所有spider中间件。关于如何使用及编写您自己的中间件，请参考 *spider middleware usage guide*.

关于默认启用的中间件列表(及其顺序)请参考 SPIDER_MIDDLEWARES_BASE 设置。

DepthMiddleware

class scrapy.contrib.spidermiddleware.depth.**DepthMiddleware**

DepthMiddleware是一个用于追踪每个Request在被爬取的网站的深度的中间件。 其可以用来限制爬取深度的最大深度或类似的事情。

DepthMiddleware 可以通过下列设置进行配置(更多内容请参考设置文档):

- DEPTH_LIMIT - 爬取所允许的最大深度，如果为0，则没有限制。
- DEPTH_STATS - 是否收集爬取状态。
- DEPTH_PRIORITY - 是否根据其深度对requet安排优先级

HttpErrorMiddleware

class scrapy.contrib.spidermiddleware.httperror.**HttpErrorMiddleware**

过滤出所有失败(错误)的HTTP response，因此spider不需要处理这些request。 处理这些request意味着消耗更多资源，并且使得spider逻辑更为复杂。

根据 HTTP标准 [http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html] ，返回值为200-300之间的值为成功的resonse。

如果您想处理在这个范围之外的response，您可以通过 spider的 handle_httpstatus_list 属性或 HTTPERROR_ALLOWED_CODES 设置来指定spider能处理的response返回值。

例如，如果您想要处理返回值为404的response您可以这么做:

```
class MySpider(CrawlSpider):
    handle_httpstatus_list = [404]
```

Request.meta 中的 handle_httpstatus_list 键也可以用来指定每个request所允许的response code。

不过请记住，除非您知道您在做什么，否则处理非200返回一般来说是个糟糕的决定。

更多内容请参考: HTTP Status Code定义 [http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html].

HttpErrorMiddleware settings

HTTPERROR_ALLOWED_CODES

默认: []

忽略该列表中所有非200状态码的response。

HTTPERROR_ALLOW_ALL

默认: False

忽略所有response，不管其状态值。

OffsiteMiddleware

class scrapy.contrib.spidermiddleware.offsite.**OffsiteMiddleware**

过滤出所有URL不由该spider负责的Request。

该中间件过滤出所有主机名不在spider属性 **allowed_domains** 的request。

当spide返回一个主机名不属于该spider的request时， 该中间件将会做一个类似于下面的记录:

```
DEBUG: Filtered offsite request to 'www.othersite.com': <GET http://www.othersite.com/some/page.html>
```

为了避免记录太多无用信息，其将对每个新发现的网站记录一次。因此，例如， 如果过滤出另一个 www.othersite.com 请求，将不会有新的记录。 但如果过滤出 someothersite.com 请求，仍然会有记录信息(仅针对第一次)。

如果spider没有定义 **allowed_domains** 属性，或该属性为空， 则offsite 中间件将会允许所有request。

如果request设置了 **dont_filter** 属性， 即使该request的网站不在允许列表里，则offsite中间件将会允许该request。

RefererMiddleware

class scrapy.contrib.spidermiddleware.referer.**RefererMiddleware**

根据生成Request的Response的URL来设置Request Referer 字段。

RefererMiddleware settings

REFERER_ENABLED

0.15 新版功能.

默认: True

是否启用referer中间件。

UrlLengthMiddleware

class scrapy.contrib.spidermiddleware.urllength.**UrlLengthMiddleware**

过滤出URL长度比URLLENGTH_LIMIT的request。

UrlLengthMiddleware 可以通过下列设置进行配置(更多内容请参考设置文档):

- **URLLENGTH_LIMIT** - 允许爬取URL最长的长度.

讨论

扩展(Extensions)

扩展框架提供一个机制，使得你能将自定义功能绑定到Scrapy。

扩展只是正常的类，它们在Scrapy启动时被实例化、初始化。

扩展设置(Extension settings)

扩展使用 *Scrapy settings* 管理它们的设置，这跟其他Scrapy代码一样。

通常扩展需要给它们的设置加上前缀，以避免跟已有(或将来)的扩展冲突。 比如，一个扩展处理 Google Sitemaps [http://en.wikipedia.org/wiki/Sitemaps]， 则可以使用类似 *GOOGLESITEMAP_ENABLED*、*GOOGLESITEMAP_DEPTH* 等设置。

加载和激活扩展

扩展在扩展类被实例化时加载和激活。 因此，所有扩展的实例化代码必须在类的构造函数(`__init__`)中执行。

要使得扩展可用，需要把它添加到Scrapy的 **EXTENSIONS** 配置中。 在 **EXTENSIONS** 中，每个扩展都使用一个字符串表示，即扩展类的全Python路径。 比如:

```
EXTENSIONS = {
    'scrapy.contrib.corestats.CoreStats': 500,
    'scrapy.telnet.TelnetConsole': 500,
}
```

如你所见，**EXTENSIONS** 配置是一个dict，key是扩展类的路径，value是顺序, 它定义扩展加载的顺序。扩展顺序不像中间件的顺序那么重要，而且扩展之间一般没有关联。 扩展加载的顺序并不重要，因为它们并不相互依赖。

如果你需要添加扩展而且它依赖别的扩展，你就可以使用该特性了。

[1] 这也是为什么Scrapy的配置项 **EXTENSIONS_BASE** (它包括了所有内置且开启的扩展)定义所有扩展的顺序都相同 (`500`)。

可用的(Available)、开启的(enabled)和禁用的(disabled)的扩展

并不是所有可用的扩展都会被开启。一些扩展经常依赖一些特别的配置。 比如，HTTP Cache扩展是可用的但默认是禁用的，除非 **HTTPCACHE_ENABLED** 配置项设置了。

禁用扩展(Disabling an extension)

为了禁用一个默认开启的扩展(比如，包含在 **EXTENSIONS_BASE** 中的扩展)， 需要将其顺序(order)设置为 `None` 。比如:

```
EXTENSIONS = {
    'scrapy.contrib.corestats.CoreStats': None,
}
```

实现你的扩展

实现你的扩展很简单。每个扩展是一个单一的Python class，它不需要实现任何特殊的方法。

Scrapy扩展(包括middlewares和pipelines)的主要入口是 `from_crawler` 类方法， 它接收一个 `Crawler` 类的实例，该实例是控制Scrapy crawler的主要对象。 如果扩展需要，你可以通过这个对象访问settings，signals，stats，控制爬虫的行为。

通常来说，扩展关联到 *signals* 并执行它们触发的任务。

最后，如果 `from_crawler` 方法抛出 **NotConfigured** 异常， 扩展会被禁用。否则，扩展会被开启。

扩展例子(Sample extension)

这里我们将实现一个简单的扩展来演示上面描述到的概念。 该扩展会在以下事件时记录一条日志：

- spider被打开
- spider被关闭
- 爬取了特定数量的条目(items)

该扩展通过 `MYEXT_ENABLED` 配置项开启， items的数量通过 `MYEXT_ITEMCOUNT` 配置项设置。

以下是扩展的代码:

```
from scrapy import signals
from scrapy.exceptions import NotConfigured
```

```python
class SpiderOpenCloseLogging(object):

    def __init__(self, item_count):
        self.item_count = item_count

        self.items_scraped = 0

    @classmethod
    def from_crawler(cls, crawler):
        # first check if the extension should be enabled and raise

        # NotConfigured otherwise

        if not crawler.settings.getbool('MYEXT_ENABLED'):

            raise NotConfigured

        # get the number of items from settings

        item_count = crawler.settings.getint('MYEXT_ITEMCOUNT', 1000)

        # instantiate the extension object

        ext = cls(item_count)

        # connect the extension object to signals

        crawler.signals.connect(ext.spider_opened, signal=signals.spider_opened)

        crawler.signals.connect(ext.spider_closed, signal=signals.spider_closed)

        crawler.signals.connect(ext.item_scraped, signal=signals.item_scraped)

        # return the extension object

        return ext

    def spider_opened(self, spider):
        spider.log("opened spider %s" % spider.name)

    def spider_closed(self, spider):
        spider.log("closed spider %s" % spider.name)

    def item_scraped(self, item, spider):
        self.items_scraped += 1

        if self.items_scraped % self.item_count == 0:
            spider.log("scraped %d items" % self.items_scraped)
```

内置扩展介绍

通用扩展

记录统计扩展(Log Stats extension)

class scrapy.contrib.logstats.**LogStats**

记录基本的统计信息，比如爬取的页面和条目(items)。

核心统计扩展(Core Stats extension)

class scrapy.contrib.corestats.**CoreStats**

如果统计收集器(stats collection)启用了，该扩展开启核心统计收集(参考 *数据收集(Stats Collection)*)。

Telnet console 扩展

class scrapy.telnet.**TelnetConsole**

提供一个telnet控制台，用于进入当前执行的Scrapy进程的Python解析器， 这对代码调试非常有帮助。

telnet控制台通过 **TELNETCONSOLE_ENABLED** 配置项开启， 服务器会监听 **TELNETCONSOLE_PORT** 指定的端口。

内存使用扩展(Memory usage extension)

class scrapy.contrib.memusage.**MemoryUsage**

注解

This extension does not work in Windows.

监控Scrapy进程内存使用量，并且：

1. 如果使用内存量超过某个指定值，发送提醒邮件
2. 如果超过某个指定值，关闭spider

当内存用量达到 MEMUSAGE_WARNING_MB 指定的值，发送提醒邮件。 当内存用量达到 MEMUSAGE_LIMIT_MB 指定的值，发送提醒邮件，同时关闭spider， Scrapy进程退出。

该扩展通过 MEMUSAGE_ENABLED 配置项开启，可以使用以下选项：

- MEMUSAGE_LIMIT_MB
- MEMUSAGE_WARNING_MB
- MEMUSAGE_NOTIFY_MAIL
- MEMUSAGE_REPORT

内存调试扩展(Memory debugger extension)

class scrapy.contrib.memdebug.**MemoryDebugger**

该扩展用于调试内存使用量，它收集以下信息：

- 没有被Python垃圾回收器收集的对象
- 应该被销毁却仍然存活的对象。更多信息请参考 *使用 trackref 调试内存泄露*

开启该扩展，需打开 MEMDEBUG_ENABLED 配置项。 信息将会存储在统计信息(stats)中。

关闭spider扩展

class scrapy.contrib.closespider.**CloseSpider**

当某些状况发生，spider会自动关闭。每种情况使用指定的关闭原因。

关闭spider的情况可以通过下面的设置项配置：

- CLOSESPIDER_TIMEOUT
- CLOSESPIDER_ITEMCOUNT
- CLOSESPIDER_PAGECOUNT
- CLOSESPIDER_ERRORCOUNT

CLOSESPIDER_TIMEOUT

默认值: 0

一个整数值，单位为秒。如果一个spider在指定的秒数后仍在运行， 它将以 closespider_timeout 的原因被自动关闭。 如果值设置为0（或者没有设置），spiders不会因为超时而关闭。

CLOSESPIDER_ITEMCOUNT

缺省值: 0

一个整数值，指定条目的个数。如果spider爬取条目数超过了指定的数， 并且这些条目通过item pipeline传递，spider将会以 closespider_itemcount 的原因被自动关闭。

CLOSESPIDER_PAGECOUNT

0.11 新版功能.

缺省值: 0

一个整数值，指定最大的抓取响应(reponses)数。 如果spider抓取数超过指定的值，则会以 closespider_pagecount 的原因自动关闭。 如果设置为0（或者未设置），spiders不会因为抓取的响应数而关闭。

CLOSESPIDER_ERRORCOUNT

0.11 新版功能.

缺省值: 0

一个整数值，指定spider可以接受的最大错误数。 如果spider生成多于该数目的错误，它将以 closespider_errorcount 的原因关闭。 如果设置为0（或者未设置），spiders不会因为发生错误过多而关闭。

StatsMailer extension

class `scrapy.contrib.statsmailer.`**`StatsMailer`**

这个简单的扩展可用来在一个域名爬取完毕时发送提醒邮件， 包含Scrapy收集的统计信息。 邮件会发送个通过 **`STATSMAILER_RCPTS`** 指定的所有接收人。

Debugging extensions

Stack trace dump extension

class `scrapy.contrib.debug.`**`StackTraceDump`**

当收到 *SIGQUIT* 或 *SIGUSR2* 信号，spider进程的信息将会被存储下来。 存储的信息包括：

1. engine状态(使用 `scrapy.utils.engin.get_engine_status()`)
2. 所有存活的引用(live references)(参考 *使用 trackref 调试内存泄露*)
3. 所有线程的堆栈信息

当堆栈信息和engine状态存储后，Scrapy进程继续正常运行。

该扩展只在POSIX兼容的平台上可运行（比如不能在Windows运行）， 因为 *SIGQUIT* 和 *SIGUSR2* 信号在Windows上不可用。

至少有两种方式可以向Scrapy发送 SIGQUIT [http://en.wikipedia.org/wiki/SIGQUIT] 信号:

1. 在Scrapy进程运行时通过按Ctrl-(仅Linux可行?)
2. 运行该命令(`<pid>` 是Scrapy运行的进程):

```
kill -QUIT <pid>
```

调试扩展(Debugger extension)

class `scrapy.contrib.debug.`**`Debugger`**

当收到 *SIGUSR2* 信号，将会在Scrapy进程中调用 Python debugger [http://docs.python.org/library/pdb.html] 。 debugger退出后，Scrapy进程继续正常运行。

更多信息参考 *Debugging in Python* 。

该扩展只在POSIX兼容平台上工作(比如不能再Windows上运行)。

讨论

核心API

0.15 新版功能.

该节文档讲述Scrapy核心API，目标用户是开发Scrapy扩展(extensions)和中间件(middlewares)的开发人员。

Crawler API

Scrapy API的主要入口是 `Crawler` 的实例对象， 通过类方法 `from_crawler` 将它传递给扩展(extensions)。 该对象提供对所有Scrapy核心组件的访问， 也是扩展访问Scrapy核心组件和挂载功能到Scrapy的唯一途径。

Extension Manager负责加载和跟踪已经安装的扩展， 它通过 `EXTENSIONS` 配置，包含一个所有可用扩展的字典， 字典的顺序跟你在 *configure the downloader middlewares* 配置的顺序一致。

class `scrapy.crawler.`**`Crawler`**(*spidercls*, *settings*)

Crawler必须使用 `scrapy.spider.Spider` 子类及 `scrapy.settings.Settings` 的对象进行实例化

`settings`

crawler的配置管理器。

扩展(extensions)和中间件(middlewares)使用它用来访问Scrapy的配置。

关于Scrapy配置的介绍参考这里 *Settings*。

API参考 `Settings`。

`signals`

crawler的信号管理器。

扩展和中间件使用它将自己的功能挂载到Scrapy。

关于信号的介绍参考 *信号(Signals)*。

API参考 `SignalManager`。

`stats`

crawler的统计信息收集器。

扩展和中间件使用它记录操作的统计信息，或者访问由其他扩展收集的统计信息。

关于统计信息收集器的介绍参考 *数据收集(Stats Collection)*。

API参考类 `StatsCollector` class。

`extensions`

扩展管理器，跟踪所有开启的扩展。

大多数扩展不需要访问该属性。

关于扩展和可用扩展列表器的介绍参考 *扩展(Extensions)*。

`engine`

执行引擎，协调crawler的核心逻辑，包括调度，下载和spider。

某些扩展可能需要访问Scrapy的引擎属性，以修改检查(modify inspect)或修改下载器和调度器的行为， 这是该API的高级使用，但还不稳定。

`spider`

正在爬取的`spider`。该`spider`类的实例由创建`crawler`时所提供，
在调用 :meth:`crawl` 方法是所创建。

`crawl`(**args*, ***kwargs*)

根据给定的 *args* , *kwargs* 的参数来初始化spider类，启动执行引擎，启动crawler。

返回一个延迟deferred对象，当爬取结束时触发它。

class `scrapy.crawler.`**`CrawlerRunner`**(*settings*)

This is a convenient helper class that creates, configures and runs crawlers inside an already setup Twisted reactor [http://twistedmatrix.com/documents/current/core/howto/reactor-basics.html].

The CrawlerRunner object must be instantiated with a `Settings` object.

This class shouldn't be needed (since Scrapy is responsible of using it accordingly) unless writing scripts that manually handle the crawling process. See *在脚本中运行Scrapy* for an example.

crawlers
Set of `crawlers` created by the `crawl()` method.

crawl_deferreds
Set of the deferreds [http://twistedmatrix.com/documents/current/core/howto/defer.html] return by the `crawl()` method. This collection it's useful for keeping track of current crawling state.

crawl(*spidercls*, **args*, ***kwargs*)
This method sets up the crawling of the given *spidercls* with the provided arguments.

It takes care of loading the spider class while configuring and starting a crawler for it.

Returns a deferred that is fired when the crawl is finished.

参数:
- **spidercls** (`Spider` subclass or str) – spider class or spider's name inside the project
- **args** (*list*) – arguments to initializate the spider
- **kwargs** (*dict*) – keyword arguments to initializate the spider

stop()
Stops simultaneously all the crawling jobs taking place.

Returns a deferred that is fired when they all have ended.

设置(Settings) API

scrapy.settings.**SETTINGS_PRIORITIES**
Dictionary that sets the key name and priority level of the default settings priorities used in Scrapy.

Each item defines a settings entry point, giving it a code name for identification and an integer priority. Greater priorities take more precedence over lesser ones when setting and retrieving values in the `Settings` class.

```
SETTINGS_PRIORITIES = {
    'default': 0,
    'command': 10,
    'project': 20,
    'spider': 30,
    'cmdline': 40,
}
```

For a detailed explanation on each settings sources, see: *Settings*.

class scrapy.settings.**Settings**(*values={}*, *priority='project'*)
This object stores Scrapy settings for the configuration of internal components, and can be used for any further customization.

After instantiation of this class, the new object will have the global default settings described on *内置设定参考手册* already populated.

Additional values can be passed on initialization with the `values` argument, and they would take the `priority` level. If the latter argument is a string, the priority name will be looked up in `SETTINGS_PRIORITIES`. Otherwise, a expecific integer should be provided.

Once the object is created, new settings can be loaded or updated with the `set()` method, and can be accessed with the square bracket notation of dictionaries, or with the `get()` method of the instance and its value conversion variants. When requesting a stored key, the value with the highest priority will be retrieved.

set(*name*, *value*, *priority='project'*)
Store a key/value attribute with a given priority. Settings should be populated *before* configuring the Crawler object (through the **configure()** method), otherwise they won't have any effect.

参数:
- **name** (*string*) – the setting name
- **value** (*any*) – the value to associate with the setting
- **priority** (*string or int*) – the priority of the setting. Should be a key of `SETTINGS_PRIORITIES` or an integer

setdict(*values*, *priority='project'*)
Store key/value pairs with a given priority.

This is a helper function that calls `set()` for every item of `values` with the provided `priority`.

参数:
- **values** (*dict*) – the settings names and values
- **priority** (*string or int*) – the priority of the settings. Should be a key of `SETTINGS_PRIORITIES` or an integer

setmodule(*module*, *priority='project'*)

Store settings from a module with a given priority.

This is a helper function that calls **set()** for every globally declared uppercase variable of `module` with the provided `priority`.

参数：
- **module** (*module object or string*) – the module or the path of the module
- **priority** (*string or int*) – the priority of the settings. Should be a key of **SETTINGS_PRIORITIES** or an integer

get(*name*, *default=None*)

获取某项配置的值，且不修改其原有的值。

参数：
- **name** (*字符串*) – 配置名
- **default** (*任何*) – 如果没有该项配置时返回的缺省值

getbool(*name*, *default=False*)

return `False` 将某项配置的值以布尔值形式返回。比如，`1` 和 `'1'`，`True` 都返回``True``， 而 `0`，`'0'`，`False` 和 `None` 返回 `False`。

比如，通过环境变量计算将某项配置设置为 `'0'`，通过该方法获取得到 `False`。

参数：
- **name** (*字符串*) – 配置名
- **default** (*任何*) – 如果该配置项未设置，返回的缺省值

getint(*name*, *default=0*)

将某项配置的值以整数形式返回

参数：
- **name** (*字符串*) – 配置名
- **default** (*任何*) – 如果该配置项未设置，返回的缺省值

getfloat(*name*, *default=0.0*)

将某项配置的值以浮点数形式返回

参数：
- **name** (*字符串*) – 配置名
- **default** (*任何*) – 如果该配置项未设置，返回的缺省值

getlist(*name*, *default=None*)

将某项配置的值以列表形式返回。如果配置值本来就是list则将返回其拷贝。 如果是字符串，则返回被 ”,” 分割后的列表。

比如，某项值通过环境变量的计算被设置为 `'one,two'` ，该方法返回[‘one’, ‘two’]。

参数：
- **name** (*字符串*) – 配置名
- **default** (*任何类型*) – 如果该配置项未设置，返回的缺省值

getdict(*name*, *default=None*)

Get a setting value as a dictionary. If the setting original type is a dictionary, a copy of it will be returned. If it’s a string it will evaluated as a json dictionary.

参数：
- **name** (*string*) – the setting name
- **default** (*any*) – the value to return if no setting is found

copy()

Make a deep copy of current settings.

This method returns a new instance of the **Settings** class, populated with the same values and their priorities.

Modifications to the new object won’t be reflected on the original settings.

freeze()

Disable further changes to the current settings.

After calling this method, the present state of the settings will become immutable. Trying to change values through the **set()** method and its variants won’t be possible and will be alerted.

frozencopy()

Return an immutable copy of the current settings.

Alias for a **freeze()** call in the object returned by **copy()**

SpiderManager API

class `scrapy.spidermanager.`**SpiderManager**

This class is in charge of retrieving and handling the spider classes defined across the project.

Custom spider managers can be employed by specifying their path in the SPIDER_MANAGER_CLASS project setting. They must fully implement the **scrapy.interfaces.ISpiderManager** interface to guarantee an errorless execution.

from_settings(*settings*)

This class method is used by Scrapy to create an instance of the class. It's called with the current project settings, and it loads the spiders found in the modules of the SPIDER_MODULES setting.

参数： **settings** (Settings instance) – project settings

load(*spider_name*)

Get the Spider class with the given name. It'll look into the previously loaded spiders for a spider class with name *spider_name* and will raise a KeyError if not found.

参数： **spider_name** (*str*) – spider class name

list()

Get the names of the available spiders in the project.

find_by_request(*request*)

List the spiders' names that can handle the given request. Will try to match the request's url against the domains of the spiders.

参数： **request** (Request instance) – queried request

信号(Signals) API

class scrapy.signalmanager.**SignalManager**

connect(*receiver*, *signal*)

链接一个接收器函数(receiver function) 到一个信号(signal)。

signal可以是任何对象，虽然Scrapy提供了一些预先定义好的信号， 参考文档 *信号(Signals)*。

参数：
- **receiver** (*可调用对象*) – 被链接到的函数
- **signal** (*对象*) – 链接的信号

send_catch_log(*signal*, ***kwargs*)

发送一个信号，捕获异常并记录日志。

关键字参数会传递给信号处理者(signal handlers)(通过方法 connect() 关联)。

send_catch_log_deferred(*signal*, ***kwargs*)

跟 send_catch_log() 相似但支持返回 deferreds [http://twistedmatrix.com/documents/current/core/howto/defer.html] 形式的信号处理器。

返回一个 deferred [http://twistedmatrix.com/documents/current/core/howto/defer.html] ， 当所有的信号处理器的延迟被触发时调用。 发送一个信号，处理异常并记录日志。

关键字参数会传递给信号处理者(signal handlers)(通过方法 connect() 关联)。

disconnect(*receiver*, *signal*)

解除一个接收器函数和一个信号的关联。这跟方法 connect() 有相反的作用， 参数也相同。

disconnect_all(*signal*)

取消给定信号绑定的所有接收器。

参数： **signal** (*object*) – 要取消绑定的信号

状态收集器(Stats Collector) API

模块 *scrapy.statscol* 下有好几种状态收集器， 它们都实现了状态收集器API对应的类 **Statscollector** (即它们都继承至该类)。

class scrapy.statscol.**StatsCollector**

get_value(*key*, *default=None*)

返回指定key的统计值，如果key不存在则返回缺省值。

get_stats()

以dict形式返回当前spider的所有统计值。

set_value(*key*, *value*)
设置key所指定的统计值为value。

set_stats(*stats*)
使用dict形式的 `stats` 参数覆盖当前的统计值。

inc_value(*key*, *count=1*, *start=0*)
增加key所对应的统计值，增长值由count指定。 如果key未设置，则使用start的值设置为初始值。

max_value(*key*, *value*)
如果key所对应的当前value小于参数所指定的value，则设置value。 如果没有key所对应的value，设置value。

min_value(*key*, *value*)
如果key所对应的当前value大于参数所指定的value，则设置value。 如果没有key所对应的value，设置value。

clear_stats()
清除所有统计信息。

以下方法不是统计收集api的一部分，但实现自定义的统计收集器时会使用到：

open_spider(*spider*)
打开指定spider进行统计信息收集。

close_spider(*spider*)
关闭指定spider。调用后，不能访问和收集统计信息。

讨论

Requests and Responses

Scrapy使用 **Request** 和 **Response** 对象爬取web站点。

一般来说，**Request** 对象在spiders中被生成并且最终传递到 下载器(Downloader)，下载器对其进行处理并返回一个 **Response** 对象，**Response** 对象还会返回到生成request的spider中。

所有 **Request** and **Response** 的子类都会实现一些在基类中非必要的 功能。它们会在 *Request subclasses* 和 *Response subclasses* 两部分进行详细的说明。

Request对象

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

一个 **Request** 对象代表一个HTTP请求，一般来讲， HTTP请求是由Spider产生并被Downloader处理进而生成一个 **Response**。

参数：
- **url** (*string*) – 请求的URL
- **callback** (*callable*) – the function that will be called with the response of this request (once its downloaded) as its first parameter. For more information see *Passing additional data to callback functions* below. If a Request doesn't specify a callback, the spider's **parse()** method will be used. Note that if exceptions are raised during processing, errback is called instead.
- **method** (*string*) – 此请求的HTTP方法。默认是 'GET'。
- **meta** (*dict*) – **Request.meta** 属性的初始值。 一旦此参数被设置， 通过参数传递的字典将会被浅拷贝。
- **body** (*str or unicode*) – request体。如果传进的参数是 unicode 类型，将会被编码为 str 类型。如果 body 参数没有给定，那么将会存储一个空的string类型，不管 这个参数是什么类型的，最终存储的都会是 str 类型(永远不会是 unicode 或是 None)。
- **headers** (*dict*) – 请求头。字典值的类型可以是strings (for single valued headers) 或是 lists (for multi-valued headers)。如果传进的值是 None ，那么HTTP头将不会被发送。
- **cookies** (*dict or list*) –
  请求的cookies。可以被设置成如下两种形式。
  1. Using a dict:

     ```
     request_with_cookies = Request(url="http://www.example.com",
                                    cookies={'currency': 'USD', 'country': 'UY'})
     ```

  2. Using a list of dicts:

     ```
     request_with_cookies = Request(url="http://www.example.com",
                                    cookies=[{'name': 'currency',
                                             'value': 'USD',
                                             'domain': 'example.com',
                                             'path': '/currency'}])
     ```

  The latter form allows for customizing the domain and path attributes of the cookie. This is only useful if the cookies are saved for later requests.
  When some site returns cookies (in a response) those are stored in the cookies for that domain and will be sent again in future requests. That's the typical behaviour of any regular web browser. However, if, for some reason, you want to avoid merging with existing cookies you can instruct Scrapy to do so by setting the dont_merge_cookies key to True in the **Request.meta**.
  Example of request without merging cookies:

  ```
  request_with_cookies = Request(url="http://www.example.com",
                                 cookies={'currency': 'USD', 'country': 'UY'},
                                 meta={'dont_merge_cookies': True})
  ```

  For more info see *CookiesMiddleware*.
- **encoding** (*string*) – the encoding of this request (defaults to 'utf-8'). This encoding will be used to percent-encode the URL and to convert the body to str (if given as unicode).
- **priority** (*int*) – the priority of this request (defaults to 0). The priority is used by the scheduler to define the order used to process requests. Requests with a higher priority value will execute earlier. Negative values are allowed in order to indicate relatively low-priority.
- **dont_filter** (*boolean*) – indicates that this request should not be filtered by the scheduler. This is used when you want to perform an identical request multiple times, to ignore the duplicates filter. Use it with care, or you will get into crawling loops. Default to False.
- **errback** (*callable*) – a function that will be called if any exception was raised while processing the request. This includes pages that failed with 404 HTTP errors and such. It receives a Twisted Failure instance as first parameter.

url

A string containing the URL of this request. Keep in mind that this attribute contains the escaped URL, so it can differ from the URL passed in the constructor.

This attribute is read-only. To change the URL of a Request use **replace()**.

method
A string representing the HTTP method in the request. This is guaranteed to be uppercase. Example: `"GET"`, `"POST"`, `"PUT"`, etc

headers
A dictionary-like object which contains the request headers.

body
A str that contains the request body.

This attribute is read-only. To change the body of a Request use `replace()`.

meta
A dict that contains arbitrary metadata for this request. This dict is empty for new Requests, and is usually populated by different Scrapy components (extensions, middlewares, etc). So the data contained in this dict depends on the extensions you have enabled.

See *Request.meta special keys* for a list of special meta keys recognized by Scrapy.

This dict is shallow copied [http://docs.python.org/library/copy.html] when the request is cloned using the `copy()` or `replace()` methods, and can also be accessed, in your spider, from the `response.meta` attribute.

copy()
Return a new Request which is a copy of this Request. See also: *Passing additional data to callback functions*.

replace([*url*, *method*, *headers*, *body*, *cookies*, *meta*, *encoding*, *dont_filter*, *callback*, *errback*])
Return a Request object with the same members, except for those members given new values by whichever keyword arguments are specified. The attribute `Request.meta` is copied by default (unless a new value is given in the `meta` argument). See also *Passing additional data to callback functions*.

Passing additional data to callback functions

The callback of a request is a function that will be called when the response of that request is downloaded. The callback function will be called with the downloaded `Response` object as its first argument.

Example:

```
def parse_page1(self, response):
    return scrapy.Request("http://www.example.com/some_page.html",
                          callback=self.parse_page2)

def parse_page2(self, response):
    # this would log http://www.example.com/some_page.html
    self.log("Visited %s" % response.url)
```

In some cases you may be interested in passing arguments to those callback functions so you can receive the arguments later, in the second callback. You can use the `Request.meta` attribute for that.

Here's an example of how to pass an item using this mechanism, to populate different fields from different pages:

```
def parse_page1(self, response):
    item = MyItem()
    item['main_url'] = response.url
    request = scrapy.Request("http://www.example.com/some_page.html",
                             callback=self.parse_page2)
    request.meta['item'] = item
    return request

def parse_page2(self, response):
    item = response.meta['item']
    item['other_url'] = response.url
    return item
```

Request.meta special keys

The `Request.meta` attribute can contain any arbitrary data, but there are some special keys recognized by Scrapy and its built-in extensions.

Those are:

- `dont_redirect`
- `dont_retry`
- `handle_httpstatus_list`
- `dont_merge_cookies` (see `cookies` parameter of `Request` constructor)
- `cookiejar`
- `redirect_urls`
- `bindaddress`
- **dont_obey_robotstxt**
- `download_timeout`

bindaddress

The IP of the outgoing IP address to use for the performing the request.

download_timeout

The amount of time (in secs) that the downloader will wait before timing out. See also: DOWNLOAD_TIMEOUT.

Request subclasses

Here is the list of built-in Request subclasses. You can also subclass it to implement your own custom functionality.

FormRequest objects

The FormRequest class extends the base Request with functionality for dealing with HTML forms. It uses lxml.html forms [http://lxml.de/lxmlhtml.html#forms] to pre-populate form fields with form data from Response objects.

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

The FormRequest class adds a new argument to the constructor. The remaining arguments are the same as for the Request class and are not documented here.

参数: **formdata** (*dict or iterable of tuples*) – is a dictionary (or iterable of (key, value) tuples) containing HTML Form data which will be url-encoded and assigned to the body of the request.

The FormRequest objects support the following class method in addition to the standard Request methods:

classmethod **from_response**(*response*[, *formname=None*, *formnumber=0*, *formdata=None*, *formxpath=None*, *clickdata=None*, *dont_click=False*, ...])

Returns a new FormRequest object with its form field values pre-populated with those found in the HTML <form> element contained in the given response. For an example see *使用FormRequest.from_response()方法模拟用户登录*.

The policy is to automatically simulate a click, by default, on any form control that looks clickable, like a <input type="submit">. Even though this is quite convenient, and often the desired behaviour, sometimes it can cause problems which could be hard to debug. For example, when working with forms that are filled and/or submitted using javascript, the default from_response() behaviour may not be the most appropriate. To disable this behaviour you can set the dont_click argument to True. Also, if you want to change the control clicked (instead of disabling it) you can also use the clickdata argument.

参数:
- **response** (Response object) – the response containing a HTML form which will be used to pre-populate the form fields
- **formname** (*string*) – if given, the form with name attribute set to this value will be used.
- **formxpath** (*string*) – if given, the first form that matches the xpath will be used.
- **formnumber** (*integer*) – the number of form to use, when the response contains multiple forms. The first one (and also the default) is 0.
- **formdata** (*dict*) – fields to override in the form data. If a field was already present in the response <form> element, its value is overridden by the one passed in this parameter.
- **clickdata** (*dict*) – attributes to lookup the control clicked. If it's not given, the form data will be submitted simulating a click on the first clickable element. In addition to html attributes, the control can be identified by its zero-based index relative to other submittable inputs inside the form, via the nr attribute.
- **dont_click** (*boolean*) – If True, the form data will be submitted without clicking in any element.

The other parameters of this class method are passed directly to the FormRequest constructor.

0.10.3 新版功能: The formname parameter.

0.17 新版功能: The formxpath parameter.

Request usage examples

Using FormRequest to send data via HTTP POST

If you want to simulate a HTML Form POST in your spider and send a couple of key-value fields, you can return a FormRequest object (from your spider) like this:

```
return [FormRequest(url="http://www.example.com/post/action",
                    formdata={'name': 'John Doe', 'age': '27'},
                    callback=self.after_post)]
```

使用FormRequest.from_response()方法模拟用户登录

通常网站通过 <input type="hidden"> 实现对某些表单字段（如数据或是登录界面中的认证令牌等）的预填充。 使用Scrapy抓取网页时，如果想要预填充或重写像用户名、用户密码这些表单字段， 可以使用 FormRequest.from_response() 方法实现。下面是使用这种方法的爬虫例子:

```python
import scrapy

class LoginSpider(scrapy.Spider):
    name = 'example.com'
    start_urls = ['http://www.example.com/users/login.php']

    def parse(self, response):
        return scrapy.FormRequest.from_response(
            response,
            formdata={'username': 'john', 'password': 'secret'},
            callback=self.after_login
        )

    def after_login(self, response):
        # check login succeed before going on
        if "authentication failed" in response.body:
            self.log("Login failed", level=log.ERROR)
            return

        # continue scraping with authenticated session...
```

Response objects

class scrapy.http.**Response**(*url*[, *status=200*, *headers*, *body*, *flags*])

A Response object represents an HTTP response, which is usually downloaded (by the Downloader) and fed to the Spiders for processing.

参数:
- **url** (*string*) – the URL of this response
- **headers** (*dict*) – the headers of this response. The dict values can be strings (for single valued headers) or lists (for multi-valued headers).
- **status** (*integer*) – the HTTP status of the response. Defaults to 200.
- **body** (*str*) – the response body. It must be str, not unicode, unless you're using a encoding-aware *Response subclass*, such as TextResponse.
- **meta** (*dict*) – the initial values for the Response.meta attribute. If given, the dict will be shallow copied.
- **flags** (*list*) – is a list containing the initial values for the Response.flags attribute. If given, the list will be shallow copied.

url

A string containing the URL of the response.

This attribute is read-only. To change the URL of a Response use replace().

status

An integer representing the HTTP status of the response. Example: 200, 404.

headers

A dictionary-like object which contains the response headers.

body

A str containing the body of this Response. Keep in mind that Response.body is always a str. If you want the unicode version use TextResponse.body_as_unicode() (only available in TextResponse and subclasses).

This attribute is read-only. To change the body of a Response use replace().

request

The Request object that generated this response. This attribute is assigned in the Scrapy engine, after the response and the request have passed through all *Downloader Middlewares*. In particular, this means that:

- HTTP redirections will cause the original request (to the URL before redirection) to be assigned to the redirected response (with the final URL after redirection).
- Response.request.url doesn't always equal Response.url
- This attribute is only available in the spider code, and in the *Spider Middlewares*, but not in Downloader Middlewares (although you have the Request available there by other means) and handlers of the response_downloaded signal.

meta

A shortcut to the Request.meta attribute of the Response.request object (ie. self.request.meta).

Unlike the Response.request attribute, the Response.meta attribute is propagated along redirects and retries, so you will get the original Request.meta sent from your spider.

参见

Request.meta attribute

flags

A list that contains flags for this response. Flags are labels used for tagging Responses. For example: *'cached'*, *'redirected'*, etc. And they're shown on the string representation of the Response (__*str*__ method) which is used by the engine for logging.

copy()

Returns a new Response which is a copy of this Response.

replace([*url*, *status*, *headers*, *body*, *request*, *flags*, *cls*])

Returns a Response object with the same members, except for those members given new values by whichever keyword arguments are specified. The attribute `Response.meta` is copied by default.

Response subclasses

Here is the list of available built-in Response subclasses. You can also subclass the Response class to implement your own functionality.

TextResponse objects

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

`TextResponse` objects adds encoding capabilities to the base `Response` class, which is meant to be used only for binary data, such as images, sounds or any media file.

`TextResponse` objects support a new constructor argument, in addition to the base `Response` objects. The remaining functionality is the same as for the `Response` class and is not documented here.

参数: **encoding** (*string*) – is a string which contains the encoding to use for this response. If you create a `TextResponse` object with a unicode body, it will be encoded using this encoding (remember the body attribute is always a string). If `encoding` is `None` (default value), the encoding will be looked up in the response headers and body instead.

`TextResponse` objects support the following attributes in addition to the standard `Response` ones:

encoding

A string with the encoding of this response. The encoding is resolved by trying the following mechanisms, in order:

1. the encoding passed in the constructor *encoding* argument
2. the encoding declared in the Content-Type HTTP header. If this encoding is not valid (ie. unknown), it is ignored and the next resolution mechanism is tried.
3. the encoding declared in the response body. The TextResponse class doesn't provide any special functionality for this. However, the `HtmlResponse` and `XmlResponse` classes do.
4. the encoding inferred by looking at the response body. This is the more fragile method but also the last one tried.

selector

A `Selector` instance using the response as target. The selector is lazily instantiated on first access.

`TextResponse` objects support the following methods in addition to the standard `Response` ones:

body_as_unicode()

Returns the body of the response as unicode. This is equivalent to:

```
response.body.decode(response.encoding)
```

But **not** equivalent to:

```
unicode(response.body)
```

Since, in the latter case, you would be using you system default encoding (typically *ascii*) to convert the body to unicode, instead of the response encoding.

xpath(*query*)

A shortcut to `TextResponse.selector.xpath(query)`:

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

css(*query*)

A shortcut to `TextResponse.selector.css(query)`:

```
response.css('p')
```

HtmlResponse objects

class `scrapy.http.`**HtmlResponse**(*url*[, ...])

The `HtmlResponse` class is a subclass of `TextResponse` which adds encoding auto-discovering support by looking into the HTML meta http-equiv [http://www.w3schools.com/TAGS/att_meta_http_equiv.asp] attribute. See `TextResponse.encoding`.

XmlResponse objects

class scrapy.http.**XmlResponse**(*url*[, ...])

The **XmlResponse** class is a subclass of **TextResponse** which adds encoding auto-discovering support by looking into the XML declaration line. See **TextResponse.encoding**.

讨论

Settings

Scrapy设定(settings)提供了定制Scrapy组件的方法。您可以控制包括核心(core)，插件(extension)，pipeline及spider组件。

设定为代码提供了提取以key-value映射的配置值的的全局命名空间(namespace)。 设定可以通过下面介绍的多种机制进行设置。

设定(settings)同时也是选择当前激活的Scrapy项目的方法(如果您有多个的话)。

内置设定列表请参考 *内置设定参考手册* 。

指定设定(Designating the settings)

当您使用Scrapy时，您需要声明您所使用的设定。这可以通过使用环境变量: `SCRAPY_SETTINGS_MODULE` 来完成。

`SCRAPY_SETTINGS_MODULE` 必须以Python路径语法编写, 如 `myproject.settings` 。 注意，设定模块应该在 Python import search path [http://docs.python.org/2/tutorial/modules.html#the-module-search-path] 中。

获取设定值(Populating the settings)

设定可以通过多种方式设置，每个方式具有不同的优先级。 下面以优先级降序的方式给出方式列表:

1. 命令行选项(Command line Options)(最高优先级)
2. 每个spider的设定
3. 项目设定模块(Project settings module)
4. 命令默认设定模块(Default settings per-command)
5. 全局默认设定(Default global settings) (最低优先级)

这些设定(settings)由scrapy内部很好的进行了处理，不过您仍可以使用API调用来手动处理。 详情请参考 *设置(Settings) API*.

这些机制将在下面详细介绍。

1. 命令行选项(Command line options)

命令行传入的参数具有最高的优先级。 您可以使用command line 选项 `-s` (或 `--set`) 来覆盖一个(或更多)选项。

样例:

```
scrapy crawl myspider -s LOG_FILE=scrapy.log
```

2. 项目设定模块(Project settings module)

项目设定模块是您Scrapy项目的标准配置文件。 其是获取大多数设定的方法。例如:: `myproject.settings` 。

3. 命令默认设定(Default settings per-command)

每个 *Scrapy tool* 命令拥有其默认设定，并覆盖了全局默认的设定。 这些设定在命令的类的 `default_settings` 属性中指定。

4. 默认全局设定(Default global settings)

全局默认设定存储在 `scrapy.settings.default_settings` 模块， 并在 *内置设定参考手册* 部分有所记录。

如何访问设定(How to access settings)

设定可以通过Crawler的 **`scrapy.crawler.Crawler.settings`** 属性进行访问。其由插件及中间件的 `from_crawler` 方法所传入:

```
class MyExtension(object):

    @classmethod
    def from_crawler(cls, crawler):
        settings = crawler.settings
        if settings['LOG_ENABLED']:
            print "log is enabled!"
```

另外，设定可以以字典方式进行访问。不过为了避免类型错误， 通常更希望返回需要的格式。 这可以通过 **`Settings`** API 提供的方法来实现。

设定名字的命名规则

设定的名字以要配置的组件作为前缀。 例如，一个robots.txt插件的合适设定应该为 `ROBOTSTXT_ENABLED`, `ROBOTSTXT_OBEY`, `ROBOTSTXT_CACHEDIR` 等等。

内置设定参考手册

这里以字母序给出了所有可用的Scrapy设定及其默认值和应用范围。

如果给出可用范围，并绑定了特定的组件，则说明了该设定使用的地方。 这种情况下将给出该组件的模块，通常来说是插件、中间件或pipeline。 同时也意味着为了使设定生效，该组件必须被启用。

AWS_ACCESS_KEY_ID

默认: `None`

连接 Amazon Web services [http://aws.amazon.com/] 的AWS access key。 *S3 feed storage backend* 中使用.

AWS_SECRET_ACCESS_KEY

默认: `None`

连接 Amazon Web services [http://aws.amazon.com/] 的AWS secret key。 *S3 feed storage backend* 中使用。

BOT_NAME

默认: `'scrapybot'`

Scrapy项目实现的bot的名字(也为项目名称)。 这将用来构造默认 User-Agent，同时也用来log。

当您使用 **`startproject`** 命令创建项目时其也被自动赋值。

CONCURRENT_ITEMS

默认: `100`

Item Processor(即 *Item Pipeline*) 同时处理(每个response的)item的最大值。

CONCURRENT_REQUESTS

默认: `16`

Scrapy downloader 并发请求(concurrent requests)的最大值。

CONCURRENT_REQUESTS_PER_DOMAIN

默认: `8`

对单个网站进行并发请求的最大值。

CONCURRENT_REQUESTS_PER_IP

默认: `0`

对单个IP进行并发请求的最大值。如果非0，则忽略 **`CONCURRENT_REQUESTS_PER_DOMAIN`** 设定， 使用该设定。 也就是说，并发限制将针对IP，而不是网站。

该设定也影响 **`DOWNLOAD_DELAY`**: 如果 **`CONCURRENT_REQUESTS_PER_IP`** 非0，下载延迟应用在IP而不是网站上。

DEFAULT_ITEM_CLASS

默认: `'scrapy.item.Item'`

the Scrapy shell 中实例化item使用的默认类。

DEFAULT_REQUEST_HEADERS

默认:

```
{
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language': 'en',
}
```

Scrapy HTTP Request使用的默认header。由 **`DefaultHeadersMiddleware`** 产生。

DEPTH_LIMIT

默认: 0

爬取网站最大允许的深度(depth)值。如果为0，则没有限制。

DEPTH_PRIORITY

默认: 0

整数值。用于根据深度调整request优先级。

如果为0，则不根据深度进行优先级调整。

DEPTH_STATS

默认: True

是否收集最大深度数据。

DEPTH_STATS_VERBOSE

默认: False

是否收集详细的深度数据。如果启用，每个深度的请求数将会被收集在数据中。

DNSCACHE_ENABLED

默认: True

是否启用DNS内存缓存(DNS in-memory cache)。

DOWNLOADER

默认: 'scrapy.core.downloader.Downloader'

用于crawl的downloader.

DOWNLOADER_MIDDLEWARES

默认:: {}

保存项目中启用的下载中间件及其顺序的字典。 更多内容请查看 *激活下载器中间件* 。

DOWNLOADER_MIDDLEWARES_BASE

默认:

```
{
    'scrapy.contrib.downloadermiddleware.robotstxt.RobotsTxtMiddleware': 100,
    'scrapy.contrib.downloadermiddleware.httpauth.HttpAuthMiddleware': 300,
    'scrapy.contrib.downloadermiddleware.downloadtimeout.DownloadTimeoutMiddleware': 350,
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': 400,
    'scrapy.contrib.downloadermiddleware.retry.RetryMiddleware': 500,
    'scrapy.contrib.downloadermiddleware.defaultheaders.DefaultHeadersMiddleware': 550,
    'scrapy.contrib.downloadermiddleware.redirect.MetaRefreshMiddleware': 580,
    'scrapy.contrib.downloadermiddleware.httpcompression.HttpCompressionMiddleware': 590,
    'scrapy.contrib.downloadermiddleware.redirect.RedirectMiddleware': 600,
    'scrapy.contrib.downloadermiddleware.cookies.CookiesMiddleware': 700,
    'scrapy.contrib.downloadermiddleware.httpproxy.HttpProxyMiddleware': 750,
    'scrapy.contrib.downloadermiddleware.chunked.ChunkedTransferMiddleware': 830,
    'scrapy.contrib.downloadermiddleware.stats.DownloaderStats': 850,
    'scrapy.contrib.downloadermiddleware.httpcache.HttpCacheMiddleware': 900,
}
```

包含Scrapy默认启用的下载中间件的字典。 永远不要在项目中修改该设定，而是修改 **DOWNLOADER_MIDDLEWARES** 。更多内容请参考 *激活下载器中间件*.

DOWNLOADER_STATS

默认: True

是否收集下载器数据。

DOWNLOAD_DELAY

默认: 0

下载器在下载同一个网站下一个页面前需要等待的时间。该选项可以用来限制爬取速度， 减轻服务器压力。同时也支持小数:

```
DOWNLOAD_DELAY = 0.25    # 250 ms of delay
```

该设定影响(默认启用的) **RANDOMIZE_DOWNLOAD_DELAY** 设定。 默认情况下，Scrapy在两个请求间不等待一个固定的值， 而是使用0.5到1.5之间的一个随机值 * **DOWNLOAD_DELAY** 的结果作为等待间隔。

当 **CONCURRENT_REQUESTS_PER_IP** 非0时，延迟针对的是每个ip而不是网站。

另外您可以通过spider的 `download_delay` 属性为每个spider设置该设定。

DOWNLOAD_HANDLERS

默认: `{}`

保存项目中启用的下载处理器(request downloader handler)的字典。 例子请查看 *DOWNLOAD_HANDLERS_BASE* 。

DOWNLOAD_HANDLERS_BASE

默认:

```
{
    'file': 'scrapy.core.downloader.handlers.file.FileDownloadHandler',
    'http': 'scrapy.core.downloader.handlers.http.HttpDownloadHandler',
    'https': 'scrapy.core.downloader.handlers.http.HttpDownloadHandler',
    's3': 'scrapy.core.downloader.handlers.s3.S3DownloadHandler',
}
```

保存项目中默认启用的下载处理器(request downloader handler)的字典。 永远不要在项目中修改该设定，而是修改 **DOWNLOADER_HANDLERS** 。

如果需要关闭上面的下载处理器，您必须在项目中的 **DOWNLOAD_HANDLERS** 设定中设置该处理器，并为其赋值为 *None* 。 例如，关闭文件下载处理器:

```
DOWNLOAD_HANDLERS = {
    'file': None,
}
```

DOWNLOAD_TIMEOUT

默认: `180`

下载器超时时间(单位: 秒)。

注解

该超时值可以使用 **download_timeout** 来对每个spider进行设置, 也可以使用 **download_timeout** Request.meta key 来对每个请求进行设置.

This feature needs Twisted >= 11.1.

DOWNLOAD_MAXSIZE

Default: *1073741824* (1024MB)

The maximum response size (in bytes) that downloader will download.

If you want to disable it set to 0.

注解

This size can be set per spider using **download_maxsize** spider attribute and per-request using **download_maxsize** Request.meta key.

DOWNLOAD_WARNSIZE

Default: *33554432* (32Mb)

The response size (in bytes) that downloader will start to warn.

If you want to disable it set to 0.

注解

This size can be set per spider using **download_warnsize** spider attribute and per-request using **download_warnsize** Request.meta key.

This feature needs Twisted >= 11.1.

DUPEFILTER_CLASS

默认: `'scrapy.dupefilter.RFPDupeFilter'`

用于检测过滤重复请求的类。

默认的 (`RFPDupeFilter`) 过滤器基于 `scrapy.utils.request.request_fingerprint` 函数生成的请求fingerprint(指纹)。 如果您需要修改检测的方式，您可以继承 `RFPDupeFilter` 并覆盖其 `request_fingerprint` 方法。 该方法接收 **`Request`** 对象并返回其fingerprint(一个字符串)。

DUPEFILTER_DEBUG

默认: `False`

默认情况下， `RFPDupeFilter` 只记录第一次重复的请求。 设置 **`DUPEFILTER_DEBUG`** 为 `True` 将会使其记录所有重复的requests。

EDITOR

默认: *depends on the environment*

执行 **`edit`** 命令编辑spider时使用的编辑器。 其默认为 `EDITOR` 环境变量。如果该变量未设置，其默认为 `vi` (Unix系统) 或者 IDLE编辑器 (Windows)。

EXTENSIONS

默认:: `{}`

保存项目中启用的插件及其顺序的字典。

EXTENSIONS_BASE

默认:

```
{
    'scrapy.contrib.corestats.CoreStats': 0,
    'scrapy.telnet.TelnetConsole': 0,
    'scrapy.contrib.memusage.MemoryUsage': 0,
    'scrapy.contrib.memdebug.MemoryDebugger': 0,
    'scrapy.contrib.closespider.CloseSpider': 0,
    'scrapy.contrib.feedexport.FeedExporter': 0,
    'scrapy.contrib.logstats.LogStats': 0,
    'scrapy.contrib.spiderstate.SpiderState': 0,
    'scrapy.contrib.throttle.AutoThrottle': 0,
}
```

可用的插件列表。需要注意，有些插件需要通过设定来启用。默认情况下， 该设定包含所有稳定(stable)的内置插件。

更多内容请参考 *extensions用户手册* 及 *所有可用的插件* 。

ITEM_PIPELINES

默认: `{}`

保存项目中启用的pipeline及其顺序的字典。该字典默认为空，值(value)任意。 不过值(value)习惯设定在0-1000范围内。

为了兼容性，**`ITEM_PIPELINES`** 支持列表，不过已经被废弃了。

样例:

```
ITEM_PIPELINES = {
    'mybot.pipelines.validate.ValidateMyItem': 300,
    'mybot.pipelines.validate.StoreMyItem': 800,
}
```

ITEM_PIPELINES_BASE

默认: `{}`

保存项目中默认启用的pipeline的字典。 永远不要在项目中修改该设定，而是修改 **`ITEM_PIPELINES`** 。

LOG_ENABLED

默认: `True`

是否启用logging。

LOG_ENCODING

默认: `'utf-8'`

logging使用的编码。

LOG_FILE

默认: `None`

logging输出的文件名。如果为None，则使用标准错误输出(standard error)。

LOG_LEVEL

默认: `'DEBUG'`

log的最低级别。可选的级别有: CRITICAL、 ERROR、WARNING、INFO、DEBUG。更多内容请查看 *Logging* 。

LOG_STDOUT

默认: `False`

如果为 `True` ，进程所有的标准输出(及错误)将会被重定向到log中。例如， 执行 `print 'hello'` ，其将会在Scrapy log中显示。

MEMDEBUG_ENABLED

默认: `False`

是否启用内存调试(memory debugging)。

MEMDEBUG_NOTIFY

默认: `[]`

如果该设置不为空，当启用内存调试时将会发送一份内存报告到指定的地址；否则该报告将写到log中。

样例:

```
MEMDEBUG_NOTIFY = ['user@example.com']
```

MEMUSAGE_ENABLED

默认: `False`

Scope: `scrapy.contrib.memusage`

是否启用内存使用插件。当Scrapy进程占用的内存超出限制时，该插件将会关闭Scrapy进程， 同时发送email进行通知。

See *内存使用扩展(Memory usage extension)*.

MEMUSAGE_LIMIT_MB

默认: `0`

Scope: `scrapy.contrib.memusage`

在关闭Scrapy之前所允许的最大内存数(单位: MB)(如果 MEMUSAGE_ENABLED为True)。 如果为0，将不做限制。

See *内存使用扩展(Memory usage extension)*.

MEMUSAGE_NOTIFY_MAIL

默认: `False`

Scope: `scrapy.contrib.memusage`

达到内存限制时通知的email列表。

Example:

```
MEMUSAGE_NOTIFY_MAIL = ['user@example.com']
```

See *内存使用扩展(Memory usage extension)*.

MEMUSAGE_REPORT

默认: `False`

Scope: `scrapy.contrib.memusage`

每个spider被关闭时是否发送内存使用报告。

查看 *内存使用扩展(Memory usage extension)*.

MEMUSAGE_WARNING_MB

默认: `0`

Scope: `scrapy.contrib.memusage`

在发送警告email前所允许的最大内存数(单位: MB)(如果 MEMUSAGE_ENABLED为True)。 如果为0，将不发送警告。

NEWSPIDER_MODULE

默认: `''`

使用 **`genspider`** 命令创建新spider的模块。

样例:

```
NEWSPIDER_MODULE = 'mybot.spiders_dev'
```

RANDOMIZE_DOWNLOAD_DELAY

默认: `True`

如果启用，当从相同的网站获取数据时，Scrapy将会等待一个随机的值 (0.5到1.5之间的一个随机值 * **`DOWNLOAD_DELAY`**)。

该随机值降低了crawler被检测到(接着被block)的机会。某些网站会分析请求， 查找请求之间时间的相似性。

随机的策略与 wget [http://www.gnu.org/software/wget/manual/wget.html] `--random-wait` 选项的策略相同。

若 **`DOWNLOAD_DELAY`** 为0(默认值)，该选项将不起作用。

REDIRECT_MAX_TIMES

默认: `20`

定义request允许重定向的最大次数。超过该限制后该request直接返回获取到的结果。 对某些任务我们使用Firefox默认值。

REDIRECT_MAX_METAREFRESH_DELAY

默认: `100`

有些网站使用 meta-refresh 重定向到session超时页面， 因此我们限制自动重定向到最大延迟(秒)。 =>有点不肯定:

REDIRECT_PRIORITY_ADJUST

默认: `+2`

修改重定向请求相对于原始请求的优先级。 负数意味着更多优先级。

ROBOTSTXT_OBEY

默认: `False`

Scope: `scrapy.contrib.downloadermiddleware.robotstxt`

如果启用，Scrapy将会尊重 robots.txt策略。更多内容请查看 *RobotsTxtMiddleware* 。

SCHEDULER

默认: `'scrapy.core.scheduler.Scheduler'`

用于爬取的调度器。

SPIDER_CONTRACTS

默认:: `{}`

保存项目中启用用于测试spider的scrapy contract及其顺序的字典。 更多内容请参考 *Spiders Contracts* 。

SPIDER_CONTRACTS_BASE

默认:

```
{
    'scrapy.contracts.default.UrlContract' : 1,
    'scrapy.contracts.default.ReturnsContract': 2,
    'scrapy.contracts.default.ScrapesContract': 3,
}
```

保存项目中默认启用的scrapy contract的字典。 永远不要在项目中修改该设定，而是修改 **SPIDER_CONTRACTS** 。更多内容请参考 *Spiders Contracts* 。

SPIDER_MANAGER_CLASS

默认: `'scrapy.spidermanager.SpiderManager'`

用于管理spider的类。该类必须实现 *SpiderManager API*

SPIDER_MIDDLEWARES

默认:: `{}`

保存项目中启用的下载中间件及其顺序的字典。 更多内容请参考 *激活spider中间件* 。

SPIDER_MIDDLEWARES_BASE

默认:

```
{
    'scrapy.contrib.spidermiddleware.httperror.HttpErrorMiddleware': 50,
    'scrapy.contrib.spidermiddleware.offsite.OffsiteMiddleware': 500,
    'scrapy.contrib.spidermiddleware.referer.RefererMiddleware': 700,
    'scrapy.contrib.spidermiddleware.urllength.UrlLengthMiddleware': 800,
    'scrapy.contrib.spidermiddleware.depth.DepthMiddleware': 900,
}
```

保存项目中默认启用的spider中间件的字典。 永远不要在项目中修改该设定，而是修改 **SPIDER_MIDDLEWARES** 。更多内容请参考 *激活spider中间件*.

SPIDER_MODULES

默认: `[]`

Scrapy搜索spider的模块列表。

样例:

```
SPIDER_MODULES = ['mybot.spiders_prod', 'mybot.spiders_dev']
```

STATS_CLASS

默认: `'scrapy.statscol.MemoryStatsCollector'`

收集数据的类。该类必须实现 *状态收集器(Stats Collector) API*.

STATS_DUMP

默认: `True`

当spider结束时dump *Scrapy状态数据* (到Scrapy log中)。

更多内容请查看 *数据收集(Stats Collection)* 。

STATSMAILER_RCPTS

默认: `[]` (空list)

spider完成爬取后发送Scrapy数据。更多内容请查看 **StatsMailer** 。

TELNETCONSOLE_ENABLED

默认: `True`

表明 *telnet 终端* (及其插件)是否启用的布尔值。

TELNETCONSOLE_PORT

默认: `[6023, 6073]`

telnet终端使用的端口范围。如果设置为 `None` 或 `0` ， 则使用动态分配的端口。更多内容请查看 *Telnet终端(Telnet Console)* 。

TEMPLATES_DIR

默认: scrapy模块内部的 `templates`

使用 **`startproject`** 命令创建项目时查找模板的目录。

URLLENGTH_LIMIT

默认: `2083`

Scope: `contrib.spidermiddleware.urllength`

爬取URL的最大长度。更多关于该设定的默认值信息请查看: http://www.boutell.com/newfaq/misc/urllength.html

USER_AGENT

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

爬取的默认User-Agent，除非被覆盖。

讨论

信号(Signals)

Scrapy使用信号来通知事情发生。您可以在您的Scrapy项目中捕捉一些信号(使用 *extension*)来完成额外的工作或添加额外的功能，扩展Scrapy。

虽然信号提供了一些参数，不过处理函数不用接收所有的参数 - 信号分发机制(singal dispatching mechanism)仅仅提供处理器(handler)接受的参数。

您可以通过 *信号(Signals) API* 来连接(或发送您自己的)信号。

延迟的信号处理器(Deferred signal handlers)

有些信号支持从处理器返回 Twisted deferreds [http://twistedmatrix.com/documents/current/core/howto/defer.html] ，参考下边的 *内置信号参考手册(Built-in signals reference)* 来了解哪些支持。

内置信号参考手册(Built-in signals reference)

以下给出Scrapy内置信号的列表及其意义。

engine_started

scrapy.signals.**engine_started**()

当Scrapy引擎启动爬取时发送该信号。

该信号支持返回deferreds。

注解

该信号可能会在信号 `spider_opened` 之后被发送，取决于spider的启动方式。 所以不要 **依赖** 该信号会比 **spider-opened** 更早被发送。

engine_stopped

scrapy.signals.**engine_stopped**()

当Scrapy引擎停止时发送该信号(例如，爬取结束)。

该信号支持返回deferreds。

item_scraped

scrapy.signals.**item_scraped**(*item*, *response*, *spider*)

当item被爬取，并通过所有 *Item Pipeline* 后(没有被丢弃(dropped)，发送该信号。

该信号支持返回deferreds。

参数：
- **item** (`Item` 对象) – 爬取到的item
- **spider** (`Spider` 对象) – 爬取item的spider
- **response** (`Response` 对象) – 提取item的response

item_dropped

scrapy.signals.**item_dropped**(*item*, *exception*, *spider*)

当item通过 *Item Pipeline* ，有些pipeline抛出 `DropItem` 异常，丢弃item时，该信号被发送。

该信号支持返回deferreds。

参数：
- **item** (`Item` 对象) – *Item Pipeline* 丢弃的item
- **spider** (`Spider` 对象) – 爬取item的spider
- **exception** (`DropItem` 异常) – 导致item被丢弃的异常(必须是 `DropItem` 的子类)

spider_closed

scrapy.signals.**spider_closed**(*spider*, *reason*)

当某个spider被关闭时，该信号被发送。该信号可以用来释放每个spider在 `spider_opened` 时占用的资源。

该信号支持返回deferreds。

参数：
- **spider** (Spider 对象) – 关闭的spider
- **reason** (*str*) – 描述spider被关闭的原因的字符串。如果spider是由于完成爬取而被关闭，则其为 `'finished'` 。否则，如果spider是被引擎的 `close_spider` 方法所关闭，则其为调用该方法时传入的 `reason` 参数(默认为 `'cancelled'`)。如果引擎被关闭(例如， 输入Ctrl-C)，则其为 `'shutdown'` 。

spider_opened

`scrapy.signals.`**`spider_opened`**(*spider*)

当spider开始爬取时发送该信号。该信号一般用来分配spider的资源，不过其也能做任何事。

该信号支持返回deferreds。

参数： **spider** (Spider 对象) – 开启的spider

spider_idle

`scrapy.signals.`**`spider_idle`**(*spider*)

当spider进入空闲(idle)状态时该信号被发送。空闲意味着:

- requests正在等待被下载
- requests被调度
- items正在item pipeline中被处理

当该信号的所有处理器(handler)被调用后，如果spider仍然保持空闲状态， 引擎将会关闭该spider。当spider被关闭后，spider_closed 信号将被发送。

您可以，比如，在 spider_idle 处理器中调度某些请求来避免spider被关闭。

该信号 **不支持** 返回deferreds。

参数： **spider** (Spider 对象) – 空闲的spider

spider_error

`scrapy.signals.`**`spider_error`**(*failure*, *response*, *spider*)

当spider的回调函数产生错误时(例如，抛出异常)，该信号被发送。

参数：
- **failure** (Failure 对象) – 以Twisted Failure 对象抛出的异常
- **response** (Response 对象) – 当异常被抛出时被处理的response
- **spider** (Spider 对象) – 抛出异常的spider

request_scheduled

`scrapy.signals.`**`request_scheduled`**(*request*, *spider*)

当引擎调度一个 Request 对象用于下载时，该信号被发送。

该信号 **不支持** 返回deferreds。

参数：
- **request** (Request 对象) – 到达调度器的request
- **spider** (Spider 对象) – 产生该request的spider

request_dropped

`scrapy.signals.`**`request_dropped`**(*request*, *spider*)

Sent when a Request, scheduled by the engine to be downloaded later, is rejected by the scheduler.

The signal does not support returning deferreds from their handlers.

参数：
- **request** (Request object) – the request that reached the scheduler
- **spider** (Spider object) – the spider that yielded the request

response_received

`scrapy.signals.`**`response_received`**(*response*, *request*, *spider*)

当引擎从downloader获取到一个新的 Response 时发送该信号。

该信号 **不支持** 返回deferreds。

参数：
- **response** (Response 对象) – 接收到的response
- **request** (Request 对象) – 生成response的request
- **spider** (Spider 对象) – response所对应的spider

response_downloaded

scrapy.signals.**response_downloaded**(*response*, *request*, *spider*)

当一个 HTTPResponse 被下载时，由downloader发送该信号。

该信号 不支持 返回deferreds。

参数：
- **response** (Response 对象) – 下载的response
- **request** (Request 对象) – 生成response的request
- **spider** (Spider 对象) – response所对应的spider

讨论

异常(Exceptions)

内置异常参考手册(Built-in Exceptions reference)

下面是Scrapy提供的异常及其用法。

DropItem

exception scrapy.exceptions.**DropItem**

该异常由item pipeline抛出，用于停止处理item。详细内容请参考 *Item Pipeline* 。

CloseSpider

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

该异常由spider的回调函数(callback)抛出，来暂停/停止spider。支持的参数:

参数： **reason** (*str*) – 关闭的原因

样例:

```
def parse_page(self, response):
    if 'Bandwidth exceeded' in response.body:
        raise CloseSpider('bandwidth_exceeded')
```

IgnoreRequest

exception scrapy.exceptions.**IgnoreRequest**

该异常由调度器(Scheduler)或其他下载中间件抛出，声明忽略该request。

NotConfigured

exception scrapy.exceptions.**NotConfigured**

该异常由某些组件抛出，声明其仍然保持关闭。这些组件包括:

- Extensions
- Item pipelines
- Downloader middlwares
- Spider middlewares

该异常必须由组件的构造器(constructor)抛出。

NotSupported

exception scrapy.exceptions.**NotSupported**

该异常声明一个不支持的特性。

讨论

Item Exporters

当你抓取了你要的数据(Items)，你就会想要将他们持久化或导出它们，并应用在其他的程序。这是整个抓取过程的目的。

为此，Scrapy提供了Item Exporters 来创建不同的输出格式，如XML，CSV或JSON。

使用 Item Exporter

如果你很忙，只想使用 Item Exporter 输出数据，请查看 *Feed exports*. 相反，如果你想知道Item Exporter 是如何工作的，或需要更多的自定义功能（不包括默认的 exports），请继续阅读下文。

为了使用 Item Exporter，你必须对 Item Exporter 及其参数 (args) 实例化。每个 Item Exporter 需要不同的参数，详细请查看 *Item Exporters 参考资料* 。在实例化了 exporter 之后，你必须：

1. 调用方法 `start_exporting()` 以标识 exporting 过程的开始。
2. 对要导出的每个项目调用 `export_item()` 方法。
3. 最后调用 `finish_exporting()` 表示 exporting 过程的结束

这里，你可以看到一个 *Item Pipeline* ，它使用 Item Exporter 导出 items 到不同的文件，每个 spider 一个:

```
from scrapy import signals
from scrapy.contrib.exporter import XmlItemExporter

class XmlExportPipeline(object):

    def __init__(self):
        self.files = {}

     @classmethod
     def from_crawler(cls, crawler):
         pipeline = cls()
         crawler.signals.connect(pipeline.spider_opened, signals.spider_opened)
         crawler.signals.connect(pipeline.spider_closed, signals.spider_closed)
         return pipeline

    def spider_opened(self, spider):
        file = open('%s_products.xml' % spider.name, 'w+b')
        self.files[spider] = file
        self.exporter = XmlItemExporter(file)
        self.exporter.start_exporting()

    def spider_closed(self, spider):
        self.exporter.finish_exporting()
        file = self.files.pop(spider)
        file.close()

    def process_item(self, item, spider):
        self.exporter.export_item(item)
        return item
```

序列化 item fields

B默认情况下，该字段值将不变的传递到序列化库，如何对其进行序列化的决定被委托给每一个特定的序列化库。

但是，你可以自定义每个字段值如何序列化在它被传递到序列化库中之前。

有两种方法可以自定义一个字段如何被序列化，请看下文。

1. 在 field 类中声明一个 serializer

您可以在 *field metadata* 声明一个 serializer。该 serializer 必须可调用，并返回它的序列化形式。

实例:

```
import scrapy

def serialize_price(value):
    return '$ %s' % str(value)

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field(serializer=serialize_price)
```

2. 覆盖(overriding) serialize_field() 方法

你可以覆盖 `serialize_field()` 方法来自定义如何输出你的数据。

在你的自定义代码后确保你调用父类的 `serialize_field()` 方法。

实例:

```
from scrapy.contrib.exporter import XmlItemExporter

class ProductXmlExporter(XmlItemExporter):

    def serialize_field(self, field, name, value):
        if field == 'price':
            return '$ %s' % str(value)
        return super(Product, self).serialize_field(field, name, value)
```

Item Exporters 参考资料

下面是一些Scrapy内置的 Item Exporters类. 其中一些包括了实例, 假设你要输出以下2个Items:

```
Item(name='Color TV', price='1200')
Item(name='DVD player', price='200')
```

BaseItemExporter

class `scrapy.contrib.exporter.`**`BaseItemExporter`**(*fields_to_export=None*, *export_empty_fields=False*, *encoding='utf-8'*)

这是一个对所有 Item Exporters 的(抽象)父类。它对所有(具体) Item Exporters 提供基本属性，如定义export什么fields, 是否export空fields, 或是否进行编码。

你可以在构造器中设置它们不同的属性值: `fields_to_export` , `export_empty_fields`, `encoding`.

`export_item`(*item*)

输出给定item. 此方法必须在子类中实现.

`serialize_field`(*field*, *name*, *value*)

返回给定field的序列化值. 你可以覆盖此方法来控制序列化或输出指定的field.

默认情况下, 此方法寻找一个 serializer *在 item field 中声明* 并返回它的值. 如果没有发现 serializer, 则值不会改变，除非你使用 `unicode` 值并编码到 `str`， 编码可以在 `encoding` 属性中声明.

参数:
- **field** (`Field` object) – the field being serialized
- **name** (*str*) – the name of the field being serialized
- **value** – the value being serialized

`start_exporting`()

表示exporting过程的开始. 一些exporters用于产生需要的头元素(例如 `XmlItemExporter`). 在实现exporting item前必须调用此方法.

`finish_exporting`()

表示exporting过程的结束. 一些exporters用于产生需要的尾元素 (例如 `XmlItemExporter`). 在完成exporting item后必须调用此方法.

`fields_to_export`

列出export什么fields值, None表示export所有fields. 默认值为None.

一些 exporters (例如 `CsvItemExporter`) 按照定义在属性中fields的次序依次输出.

`export_empty_fields`

是否在输出数据中包含为空的item fields. 默认值是 `False`. 一些 exporters (例如 `CsvItemExporter`) 会忽略此属性并输出所有fields.

`encoding`

Encoding 属性将用于编码 unicode 值. (仅用于序列化字符串).其他值类型将不变的传递到指定的序列化库.

XmlItemExporter

class `scrapy.contrib.exporter.`**`XmlItemExporter`**(*file*, *item_element='item'*, *root_element='items'*, ***kwargs*)

以XML格式 exports Items 到指定的文件类.

参数:
- **file** – 文件类.
- **root_element** (*str*) – XML 根元素名.
- **item_element** (*str*) – XML item 的元素名.

构造器额外的关键字参数将传给 `BaseItemExporter` 构造器.

一个典型的 exporter 实例:

```
<?xml version="1.0" encoding="utf-8"?>
<items>
```

```
  <item>
    <name>Color TV</name>
    <price>1200</price>
 </item>
  <item>
    <name>DVD player</name>
    <price>200</price>
 </item>
</items>
```

除了覆盖 **serialize_field()** 方法, 多个值的 fields 会转化每个值到 <value> 元素.

例如, item:

```
Item(name=['John', 'Doe'], age='23')
```

将被转化为:

```
<?xml version="1.0" encoding="utf-8"?>
<items>
  <item>
    <name>
      <value>John</value>
      <value>Doe</value>
    </name>
    <age>23</age>
  </item>
</items>
```

CsvItemExporter

class scrapy.contrib.exporter.**CsvItemExporter**(*file*, *include_headers_line=True*, *join_multivalued=','*, ***kwargs*)

输出 csv 文件格式. 如果添加 **fields_to_export** 属性, 它会按顺序定义CSV的列名. **export_empty_fields** 属性在此没有作用.

参数:
- **file** – 文件类.
- **include_headers_line** (*str*) – 启用后 exporter 会输出第一行为列名, 列名从 **BaseItemExporter.fields_to_export** 或第一个 item fields 获取.
- **join_multivalued** – char 将用于连接多个值的fields.

此构造器额外的关键字参数将传给 **BaseItemExporter** 构造器 , 其余的将传给 csv.writer [http://docs.python.org/library/csv.html#csv.writer] 构造器, 以此来定制 exporter.

一个典型的 exporter 实例:

```
product,price
Color TV,1200
DVD player,200
```

PickleItemExporter

class scrapy.contrib.exporter.**PickleItemExporter**(*file*, *protocol=0*, ***kwargs*)

输出 pickle 文件格式.

参数:
- **file** – 文件类.
- **protocol** (*int*) – pickle 协议.

更多信息请看 pickle module documentation [http://docs.python.org/library/pickle.html].

此构造器额外的关键字参数将传给 **BaseItemExporter** 构造器.

Pickle 不是可读的格式，这里不提供实例.

PprintItemExporter

class scrapy.contrib.exporter.**PprintItemExporter**(*file*, ***kwargs*)

输出整齐打印的文件格式.

参数: **file** – 文件类.

此构造器额外的关键字参数将传给 **BaseItemExporter** 构造器.

一个典型的 exporter 实例:

```
{'name': 'Color TV', 'price': '1200'}
{'name': 'DVD player', 'price': '200'}
```

此格式会根据行的长短进行调整.

JsonItemExporter

class scrapy.contrib.exporter.**JsonItemExporter**(*file*, ***kwargs*)

输出 JSON 文件格式, 所有对象将写进一个对象的列表. 此构造器额外的关键字参数将传给 **BaseItemExporter** 构造器, 其余的将传给 JSONEncoder [http://docs.python.org/library/json.html#json.JSONEncoder] 构造器, 以此来定制 exporter.

参数： **file** – 文件类.

一个典型的 exporter 实例:

```
[{"name": "Color TV", "price": "1200"},
{"name": "DVD player", "price": "200"}]
```

警告

JSON 是一个简单而有弹性的格式, 但对大量数据的扩展性不是很好，因为这里会将整个对象放入内存. 如果你要JSON既强大又简单, 可以考虑 **JsonLinesItemExporter** , 或把输出对象分为多个块.

JsonLinesItemExporter

class scrapy.contrib.exporter.**JsonLinesItemExporter**(*file*, ***kwargs*)

输出 JSON 文件格式, 每行写一个 JSON-encoded 项. 此构造器额外的关键字参数将传给 **BaseItemExporter** 构造器, 其余的将传给 JSONEncoder [http://docs.python.org/library/json.html#json.JSONEncoder] 构造器, 以此来定制 exporter.

参数： **file** – 文件类.

一个典型的 exporter 实例:

```
{"name": "Color TV", "price": "1200"}
{"name": "DVD player", "price": "200"}
```

这个类能很好的处理大量数据.

讨论

Release notes

0.24.4 (2014-08-09)

- pem file is used by mockserver and required by scrapy bench (commit 5eddc68 [https://github.com/scrapy/scrapy/commit/5eddc68])
- scrapy bench needs scrapy.tests* (commit d6cb999 [https://github.com/scrapy/scrapy/commit/d6cb999])

0.24.3 (2014-08-09)

- no need to waste travis-ci time on py3 for 0.24 (commit 8e080c1 [https://github.com/scrapy/scrapy/commit/8e080c1])
- Update installation docs (commit 1d0c096 [https://github.com/scrapy/scrapy/commit/1d0c096])
- There is a trove classifier for Scrapy framework! (commit 4c701d7 [https://github.com/scrapy/scrapy/commit/4c701d7])
- update other places where w3lib version is mentioned (commit d109c13 [https://github.com/scrapy/scrapy/commit/d109c13])
- Update w3lib requirement to 1.8.0 (commit 39d2ce5 [https://github.com/scrapy/scrapy/commit/39d2ce5])
- Use w3lib.html.replace_entities() (remove_entities() is deprecated) (commit 180d3ad [https://github.com/scrapy/scrapy/commit/180d3ad])
- set zip_safe=False (commit a51ee8b [https://github.com/scrapy/scrapy/commit/a51ee8b])
- do not ship tests package (commit ee3b371 [https://github.com/scrapy/scrapy/commit/ee3b371])
- scrapy.bat is not needed anymore (commit c3861cf [https://github.com/scrapy/scrapy/commit/c3861cf])
- Modernize setup.py (commit 362e322 [https://github.com/scrapy/scrapy/commit/362e322])
- headers can not handle non-string values (commit 94a5c65 [https://github.com/scrapy/scrapy/commit/94a5c65])
- fix ftp test cases (commit a274a7f [https://github.com/scrapy/scrapy/commit/a274a7f])
- The sum up of travis-ci builds are taking like 50min to complete (commit ae1e2cc [https://github.com/scrapy/scrapy/commit/ae1e2cc])
- Update shell.rst typo (commit e49c96a [https://github.com/scrapy/scrapy/commit/e49c96a])
- removes weird indentation in the shell results (commit 1ca489d [https://github.com/scrapy/scrapy/commit/1ca489d])
- improved explanations, clarified blog post as source, added link for XPath string functions in the spec (commit 65c8f05 [https://github.com/scrapy/scrapy/commit/65c8f05])
- renamed UserTimeoutError and ServerTimeouterror #583 (commit 037f6ab [https://github.com/scrapy/scrapy/commit/037f6ab])
- adding some xpath tips to selectors docs (commit 2d103e0 [https://github.com/scrapy/scrapy/commit/2d103e0])
- fix tests to account for https://github.com/scrapy/w3lib/pull/23 (commit f8d366a [https://github.com/scrapy/scrapy/commit/f8d366a])
- get_func_args maximum recursion fix #728 (commit 81344ea [https://github.com/scrapy/scrapy/commit/81344ea])
- Updated input/ouput processor example according to #560. (commit f7c4ea8 [https://github.com/scrapy/scrapy/commit/f7c4ea8])
- Fixed Python syntax in tutorial. (commit db59ed9 [https://github.com/scrapy/scrapy/commit/db59ed9])
- Add test case for tunneling proxy (commit f090260 [https://github.com/scrapy/scrapy/commit/f090260])
- Bugfix for leaking Proxy-Authorization header to remote host when using tunneling (commit d8793af [https://github.com/scrapy/scrapy/commit/d8793af])
- Extract links from XHTML documents with MIME-Type “application/xml” (commit ed1f376 [https://github.com/scrapy/scrapy/commit/ed1f376])
- Merge pull request #793 from roysc/patch-1 (commit 91a1106 [https://github.com/scrapy/scrapy/commit/91a1106])
- Fix typo in commands.rst (commit 743e1e2 [https://github.com/scrapy/scrapy/commit/743e1e2])
- better testcase for settings.overrides.setdefault (commit e22daaf [https://github.com/scrapy/scrapy/commit/e22daaf])
- Using CRLF as line marker according to http 1.1 definition (commit 5ec430b [https://github.com/scrapy/scrapy/commit/5ec430b])

0.24.2 (2014-07-08)

- Use a mutable mapping to proxy deprecated settings.overrides and settings.defaults attribute (commit e5e8133 [https://github.com/scrapy/scrapy/commit/e5e8133])
- there is not support for python3 yet (commit 3cd6146 [https://github.com/scrapy/scrapy/commit/3cd6146])
- Update python compatible version set to debian packages (commit fa5d76b [https://github.com/scrapy/scrapy/commit/fa5d76b])
- DOC fix formatting in release notes (commit c6a9e20 [https://github.com/scrapy/scrapy/commit/c6a9e20])

0.24.1 (2014-06-27)

- Fix deprecated CrawlerSettings and increase backwards compatibility with .defaults attribute (commit 8e3f20a [https://github.com/scrapy/scrapy/commit/8e3f20a])

0.24.0 (2014-06-26)

Enhancements

- Improve Scrapy top-level namespace (issue 494 [https://github.com/scrapy/scrapy/issues/494], issue 684 [https://github.com/scrapy/scrapy/issues/684])
- Add selector shortcuts to responses (issue 554 [https://github.com/scrapy/scrapy/issues/554], issue 690 [https://github.com/scrapy/scrapy/issues/690])
- Add new lxml based LinkExtractor to replace unmantained SgmlLinkExtractor (issue 559 [https://github.com/scrapy/scrapy/issues/559], issue 761 [https://github.com/scrapy/scrapy/issues/761], issue 763 [https://github.com/scrapy/scrapy/issues/763])
- Cleanup settings API - part of per-spider settings **GSoC project** (issue 737 [https://github.com/scrapy/scrapy/issues/737])
- Add UTF8 encoding header to templates (issue 688 [https://github.com/scrapy/scrapy/issues/688], issue 762 [https://github.com/scrapy/scrapy/issues/762])

- Telnet console now binds to 127.0.0.1 by default (issue 699 [https://github.com/scrapy/scrapy/issues/699])
- Update debian/ubuntu install instructions (issue 509 [https://github.com/scrapy/scrapy/issues/509], issue 549 [https://github.com/scrapy/scrapy/issues/549])
- Disable smart strings in lxml XPath evaluations (issue 535 [https://github.com/scrapy/scrapy/issues/535])
- Restore filesystem based cache as default for http cache middleware (issue 541 [https://github.com/scrapy/scrapy/issues/541], issue 500 [https://github.com/scrapy/scrapy/issues/500], issue 571 [https://github.com/scrapy/scrapy/issues/571])
- Expose current crawler in Scrapy shell (issue 557 [https://github.com/scrapy/scrapy/issues/557])
- Improve testsuite comparing CSV and XML exporters (issue 570 [https://github.com/scrapy/scrapy/issues/570])
- New *offsite/filtered* and *offsite/domains* stats (issue 566 [https://github.com/scrapy/scrapy/issues/566])
- Support process_links as generator in CrawlSpider (issue 555 [https://github.com/scrapy/scrapy/issues/555])
- Verbose logging and new stats counters for DupeFilter (issue 553 [https://github.com/scrapy/scrapy/issues/553])
- Add a mimetype parameter to *MailSender.send()* (issue 602 [https://github.com/scrapy/scrapy/issues/602])
- Generalize file pipeline log messages (issue 622 [https://github.com/scrapy/scrapy/issues/622])
- Replace unencodeable codepoints with html entities in SGMLLinkExtractor (issue 565 [https://github.com/scrapy/scrapy/issues/565])
- Converted SEP documents to rst format (issue 629 [https://github.com/scrapy/scrapy/issues/629], issue 630 [https://github.com/scrapy/scrapy/issues/630], issue 638 [https://github.com/scrapy/scrapy/issues/638], issue 632 [https://github.com/scrapy/scrapy/issues/632], issue 636 [https://github.com/scrapy/scrapy/issues/636], issue 640 [https://github.com/scrapy/scrapy/issues/640], issue 635 [https://github.com/scrapy/scrapy/issues/635], issue 634 [https://github.com/scrapy/scrapy/issues/634], issue 639 [https://github.com/scrapy/scrapy/issues/639], issue 637 [https://github.com/scrapy/scrapy/issues/637], issue 631 [https://github.com/scrapy/scrapy/issues/631], issue 633 [https://github.com/scrapy/scrapy/issues/633], issue 641 [https://github.com/scrapy/scrapy/issues/641], issue 642 [https://github.com/scrapy/scrapy/issues/642])
- Tests and docs for clickdata's nr index in FormRequest (issue 646 [https://github.com/scrapy/scrapy/issues/646], issue 645 [https://github.com/scrapy/scrapy/issues/645])
- Allow to disable a downloader handler just like any other component (issue 650 [https://github.com/scrapy/scrapy/issues/650])
- Log when a request is discarded after too many redirections (issue 654 [https://github.com/scrapy/scrapy/issues/654])
- Log error responses if they are not handled by spider callbacks (issue 612 [https://github.com/scrapy/scrapy/issues/612], issue 656 [https://github.com/scrapy/scrapy/issues/656])
- Add content-type check to http compression mw (issue 193 [https://github.com/scrapy/scrapy/issues/193], issue 660 [https://github.com/scrapy/scrapy/issues/660])
- Run pypy tests using latest pypi from ppa (issue 674 [https://github.com/scrapy/scrapy/issues/674])
- Run test suite using pytest instead of trial (issue 679 [https://github.com/scrapy/scrapy/issues/679])
- Build docs and check for dead links in tox environment (issue 687 [https://github.com/scrapy/scrapy/issues/687])
- Make scrapy.version_info a tuple of integers (issue 681 [https://github.com/scrapy/scrapy/issues/681], issue 692 [https://github.com/scrapy/scrapy/issues/692])
- Infer exporter's output format from filename extensions (issue 546 [https://github.com/scrapy/scrapy/issues/546], issue 659 [https://github.com/scrapy/scrapy/issues/659], issue 760 [https://github.com/scrapy/scrapy/issues/760])
- Support case-insensitive domains in *url_is_from_any_domain()* (issue 693 [https://github.com/scrapy/scrapy/issues/693])
- Remove pep8 warnings in project and spider templates (issue 698 [https://github.com/scrapy/scrapy/issues/698])
- Tests and docs for *request_fingerprint* function (issue 597 [https://github.com/scrapy/scrapy/issues/597])
- Update SEP-19 for GSoC project *per-spider settings* (issue 705 [https://github.com/scrapy/scrapy/issues/705])
- Set exit code to non-zero when contracts fails (issue 727 [https://github.com/scrapy/scrapy/issues/727])
- Add a setting to control what class is instanciated as Downloader component (issue 738 [https://github.com/scrapy/scrapy/issues/738])
- Pass response in *item_dropped* signal (issue 724 [https://github.com/scrapy/scrapy/issues/724])
- Improve *scrapy check* contracts command (issue 733 [https://github.com/scrapy/scrapy/issues/733], issue 752 [https://github.com/scrapy/scrapy/issues/752])
- Document *spider.closed()* shortcut (issue 719 [https://github.com/scrapy/scrapy/issues/719])
- Document *request_scheduled* signal (issue 746 [https://github.com/scrapy/scrapy/issues/746])
- Add a note about reporting security issues (issue 697 [https://github.com/scrapy/scrapy/issues/697])
- Add LevelDB http cache storage backend (issue 626 [https://github.com/scrapy/scrapy/issues/626], issue 500 [https://github.com/scrapy/scrapy/issues/500])
- Sort spider list output of *scrapy list* command (issue 742 [https://github.com/scrapy/scrapy/issues/742])
- Multiple documentation enhancemens and fixes (issue 575 [https://github.com/scrapy/scrapy/issues/575], issue 587 [https://github.com/scrapy/scrapy/issues/587], issue 590 [https://github.com/scrapy/scrapy/issues/590], issue 596 [https://github.com/scrapy/scrapy/issues/596], issue 610 [https://github.com/scrapy/scrapy/issues/610], issue 617 [https://github.com/scrapy/scrapy/issues/617], issue 618 [https://github.com/scrapy/scrapy/issues/618], issue 627 [https://github.com/scrapy/scrapy/issues/627], issue 613 [https://github.com/scrapy/scrapy/issues/613], issue 643 [https://github.com/scrapy/scrapy/issues/643], issue 654 [https://github.com/scrapy/scrapy/issues/654], issue 675 [https://github.com/scrapy/scrapy/issues/675], issue 663 [https://github.com/scrapy/scrapy/issues/663], issue 711 [https://github.com/scrapy/scrapy/issues/711], issue 714 [https://github.com/scrapy/scrapy/issues/714])

Bugfixes

- Encode unicode URL value when creating Links in RegexLinkExtractor (issue 561 [https://github.com/scrapy/scrapy/issues/561])
- Ignore None values in ItemLoader processors (issue 556 [https://github.com/scrapy/scrapy/issues/556])
- Fix link text when there is an inner tag in SGMLLinkExtractor and HtmlParserLinkExtractor (issue 485 [https://github.com/scrapy/scrapy/issues/485], issue 574 [https://github.com/scrapy/scrapy/issues/574])
- Fix wrong checks on subclassing of deprecated classes (issue 581 [https://github.com/scrapy/scrapy/issues/581], issue 584 [https://github.com/scrapy/scrapy/issues/584])
- Handle errors caused by inspect.stack() failures (issue 582 [https://github.com/scrapy/scrapy/issues/582])
- Fix a reference to unexistent engine attribute (issue 593 [https://github.com/scrapy/scrapy/issues/593], issue 594 [https://github.com/scrapy/scrapy/issues/594])
- Fix dynamic itemclass example usage of type() (issue 603 [https://github.com/scrapy/scrapy/issues/603])

- Use lucasdemarchi/codespell to fix typos (issue 628 [https://github.com/scrapy/scrapy/issues/628])
- Fix default value of attrs argument in SgmlLinkExtractor to be tuple (issue 661 [https://github.com/scrapy/scrapy/issues/661])
- Fix XXE flaw in sitemap reader (issue 676 [https://github.com/scrapy/scrapy/issues/676])
- Fix engine to support filtered start requests (issue 707 [https://github.com/scrapy/scrapy/issues/707])
- Fix offsite middleware case on urls with no hostnames (issue 745 [https://github.com/scrapy/scrapy/issues/745])
- Testsuite doesn't require PIL anymore (issue 585 [https://github.com/scrapy/scrapy/issues/585])

0.22.2 (released 2014-02-14)

- fix a reference to unexistent engine.slots. closes #593 (commit 13c099a [https://github.com/scrapy/scrapy/commit/13c099a])
- downloaderMW doc typo (spiderMW doc copy remnant) (commit 8ae11bf [https://github.com/scrapy/scrapy/commit/8ae11bf])
- Correct typos (commit 1346037 [https://github.com/scrapy/scrapy/commit/1346037])

0.22.1 (released 2014-02-08)

- localhost666 can resolve under certain circumstances (commit 2ec2279 [https://github.com/scrapy/scrapy/commit/2ec2279])
- test inspect.stack failure (commit cc3eda3 [https://github.com/scrapy/scrapy/commit/cc3eda3])
- Handle cases when inspect.stack() fails (commit 8cb44f9 [https://github.com/scrapy/scrapy/commit/8cb44f9])
- Fix wrong checks on subclassing of deprecated classes. closes #581 (commit 46d98d6 [https://github.com/scrapy/scrapy/commit/46d98d6])
- Docs: 4-space indent for final spider example (commit 13846de [https://github.com/scrapy/scrapy/commit/13846de])
- Fix HtmlParserLinkExtractor and tests after #485 merge (commit 368a946 [https://github.com/scrapy/scrapy/commit/368a946])
- BaseSgmlLinkExtractor: Fixed the missing space when the link has an inner tag (commit b566388 [https://github.com/scrapy/scrapy/commit/b566388])
- BaseSgmlLinkExtractor: Added unit test of a link with an inner tag (commit c1cb418 [https://github.com/scrapy/scrapy/commit/c1cb418])
- BaseSgmlLinkExtractor: Fixed unknown_endtag() so that it only set current_link=None when the end tag match the opening tag (commit 7e4d627 [https://github.com/scrapy/scrapy/commit/7e4d627])
- Fix tests for Travis-CI build (commit 76c7e20 [https://github.com/scrapy/scrapy/commit/76c7e20])
- replace unencodeable codepoints with html entities. fixes #562 and #285 (commit 5f87b17 [https://github.com/scrapy/scrapy/commit/5f87b17])
- RegexLinkExtractor: encode URL unicode value when creating Links (commit d0ee545 [https://github.com/scrapy/scrapy/commit/d0ee545])
- Updated the tutorial crawl output with latest output. (commit 8da65de [https://github.com/scrapy/scrapy/commit/8da65de])
- Updated shell docs with the crawler reference and fixed the actual shell output. (commit 875b9ab [https://github.com/scrapy/scrapy/commit/875b9ab])
- PEP8 minor edits. (commit f89efaf [https://github.com/scrapy/scrapy/commit/f89efaf])
- Expose current crawler in the scrapy shell. (commit 5349cec [https://github.com/scrapy/scrapy/commit/5349cec])
- Unused re import and PEP8 minor edits. (commit 387f414 [https://github.com/scrapy/scrapy/commit/387f414])
- Ignore None's values when using the ItemLoader. (commit 0632546 [https://github.com/scrapy/scrapy/commit/0632546])
- DOC Fixed HTTPCACHE_STORAGE typo in the default value which is now Filesystem instead Dbm. (commit cde9a8c [https://github.com/scrapy/scrapy/commit/cde9a8c])
- show ubuntu setup instructions as literal code (commit fb5c9c5 [https://github.com/scrapy/scrapy/commit/fb5c9c5])
- Update Ubuntu installation instructions (commit 70fb105 [https://github.com/scrapy/scrapy/commit/70fb105])
- Merge pull request #550 from stray-leone/patch-1 (commit 6f70b6a [https://github.com/scrapy/scrapy/commit/6f70b6a])
- modify the version of scrapy ubuntu package (commit 725900d [https://github.com/scrapy/scrapy/commit/725900d])
- fix 0.22.0 release date (commit af0219a [https://github.com/scrapy/scrapy/commit/af0219a])
- fix typos in news.rst and remove (not released yet) header (commit b7f58f4 [https://github.com/scrapy/scrapy/commit/b7f58f4])

0.22.0 (released 2014-01-17)

Enhancements

- [**Backwards incompatible**] Switched HTTPCacheMiddleware backend to filesystem (issue 541 [https://github.com/scrapy/scrapy/issues/541]) To restore old backend set *HTTPCACHE_STORAGE* to *scrapy.contrib.httpcache.DbmCacheStorage*
- Proxy https:// urls using CONNECT method (issue 392 [https://github.com/scrapy/scrapy/issues/392], issue 397 [https://github.com/scrapy/scrapy/issues/397])
- Add a middleware to crawl ajax crawleable pages as defined by google (issue 343 [https://github.com/scrapy/scrapy/issues/343])
- Rename scrapy.spider.BaseSpider to scrapy.spider.Spider (issue 510 [https://github.com/scrapy/scrapy/issues/510], issue 519 [https://github.com/scrapy/scrapy/issues/519])
- Selectors register EXSLT namespaces by default (issue 472 [https://github.com/scrapy/scrapy/issues/472])
- Unify item loaders similar to selectors renaming (issue 461 [https://github.com/scrapy/scrapy/issues/461])
- Make *RFPDupeFilter* class easily subclassable (issue 533 [https://github.com/scrapy/scrapy/issues/533])
- Improve test coverage and forthcoming Python 3 support (issue 525 [https://github.com/scrapy/scrapy/issues/525])
- Promote startup info on settings and middleware to INFO level (issue 520 [https://github.com/scrapy/scrapy/issues/520])
- Support partials in *get_func_args* util (issue 506 [https://github.com/scrapy/scrapy/issues/506], issue:*504*)
- Allow running indiviual tests via tox (issue 503 [https://github.com/scrapy/scrapy/issues/503])
- Update extensions ignored by link extractors (issue 498 [https://github.com/scrapy/scrapy/issues/498])
- Add middleware methods to get files/images/thumbs paths (issue 490 [https://github.com/scrapy/scrapy/issues/490])
- Improve offsite middleware tests (issue 478 [https://github.com/scrapy/scrapy/issues/478])
- Add a way to skip default Referer header set by RefererMiddleware (issue 475 [https://github.com/scrapy/scrapy/issues/475])
- Do not send *x-gzip* in default *Accept-Encoding* header (issue 469 [https://github.com/scrapy/scrapy/issues/469])
- Support defining http error handling using settings (issue 466 [https://github.com/scrapy/scrapy/issues/466])
- Use modern python idioms wherever you find legacies (issue 497 [https://github.com/scrapy/scrapy/issues/497])
- Improve and correct documentation (issue 527 [https://github.com/scrapy/scrapy/issues/527], issue 524

[https://github.com/scrapy/scrapy/issues/524], issue 521 [https://github.com/scrapy/scrapy/issues/521], issue 517 [https://github.com/scrapy/scrapy/issues/517], issue 512 [https://github.com/scrapy/scrapy/issues/512], issue 505 [https://github.com/scrapy/scrapy/issues/505], issue 502 [https://github.com/scrapy/scrapy/issues/502], issue 489 [https://github.com/scrapy/scrapy/issues/489], issue 465 [https://github.com/scrapy/scrapy/issues/465], issue 460 [https://github.com/scrapy/scrapy/issues/460], issue 425 [https://github.com/scrapy/scrapy/issues/425], issue 536 [https://github.com/scrapy/scrapy/issues/536])

Fixes

- Update Selector class imports in CrawlSpider template (issue 484 [https://github.com/scrapy/scrapy/issues/484])
- Fix unexistent reference to *engine.slots* (issue 464 [https://github.com/scrapy/scrapy/issues/464])
- Do not try to call *body_as_unicode()* on a non-TextResponse instance (issue 462 [https://github.com/scrapy/scrapy/issues/462])
- Warn when subclassing XPathItemLoader, previously it only warned on instantiation. (issue 523 [https://github.com/scrapy/scrapy/issues/523])
- Warn when subclassing XPathSelector, previously it only warned on instantiation. (issue 537 [https://github.com/scrapy/scrapy/issues/537])
- Multiple fixes to memory stats (issue 531 [https://github.com/scrapy/scrapy/issues/531], issue 530 [https://github.com/scrapy/scrapy/issues/530], issue 529 [https://github.com/scrapy/scrapy/issues/529])
- Fix overriding url in *FormRequest.from_response()* (issue 507 [https://github.com/scrapy/scrapy/issues/507])
- Fix tests runner under pip 1.5 (issue 513 [https://github.com/scrapy/scrapy/issues/513])
- Fix logging error when spider name is unicode (issue 479 [https://github.com/scrapy/scrapy/issues/479])

0.20.2 (released 2013-12-09)

- Update CrawlSpider Template with Selector changes (commit 6d1457d [https://github.com/scrapy/scrapy/commit/6d1457d])
- fix method name in tutorial. closes GH-480 (commit b4fc359 [https://github.com/scrapy/scrapy/commit/b4fc359]

0.20.1 (released 2013-11-28)

- include_package_data is required to build wheels from published sources (commit 5ba1ad5 [https://github.com/scrapy/scrapy/commit/5ba1ad5])
- process_parallel was leaking the failures on its internal deferreds. closes #458 (commit 419a780 [https://github.com/scrapy/scrapy/commit/419a780])

0.20.0 (released 2013-11-08)

Enhancements

- New Selector's API including CSS selectors (issue 395 [https://github.com/scrapy/scrapy/issues/395] and issue 426 [https://github.com/scrapy/scrapy/issues/426]),
- Request/Response url/body attributes are now immutable (modifying them had been deprecated for a long time)
- `ITEM_PIPELINES` is now defined as a dict (instead of a list)
- Sitemap spider can fetch alternate URLs (issue 360 [https://github.com/scrapy/scrapy/issues/360])
- *Selector.remove_namespaces()* now remove namespaces from element's attributes. (issue 416 [https://github.com/scrapy/scrapy/issues/416])
- Paved the road for Python 3.3+ (issue 435 [https://github.com/scrapy/scrapy/issues/435], issue 436 [https://github.com/scrapy/scrapy/issues/436], issue 431 [https://github.com/scrapy/scrapy/issues/431], issue 452 [https://github.com/scrapy/scrapy/issues/452])
- New item exporter using native python types with nesting support (issue 366 [https://github.com/scrapy/scrapy/issues/366])
- Tune HTTP1.1 pool size so it matches concurrency defined by settings (commit b43b5f575 [https://github.com/scrapy/scrapy/commit/b43b5f575])
- scrapy.mail.MailSender now can connect over TLS or upgrade using STARTTLS (issue 327 [https://github.com/scrapy/scrapy/issues/327])
- New FilesPipeline with functionality factored out from ImagesPipeline (issue 370 [https://github.com/scrapy/scrapy/issues/370], issue 409 [https://github.com/scrapy/scrapy/issues/409])
- Recommend Pillow instead of PIL for image handling (issue 317 [https://github.com/scrapy/scrapy/issues/317])
- Added debian packages for Ubuntu quantal and raring (commit 86230c0 [https://github.com/scrapy/scrapy/commit/86230c0])
- Mock server (used for tests) can listen for HTTPS requests (issue 410 [https://github.com/scrapy/scrapy/issues/410])
- Remove multi spider support from multiple core components (issue 422 [https://github.com/scrapy/scrapy/issues/422], issue 421 [https://github.com/scrapy/scrapy/issues/421], issue 420 [https://github.com/scrapy/scrapy/issues/420], issue 419 [https://github.com/scrapy/scrapy/issues/419], issue 423 [https://github.com/scrapy/scrapy/issues/423], issue 418 [https://github.com/scrapy/scrapy/issues/418])
- Travis-CI now tests Scrapy changes against development versions of *w3lib* and *queuelib* python packages.
- Add pypy 2.1 to continuous integration tests (commit ecfa7431 [https://github.com/scrapy/scrapy/commit/ecfa7431])
- Pylinted, pep8 and removed old-style exceptions from source (issue 430 [https://github.com/scrapy/scrapy/issues/430], issue 432 [https://github.com/scrapy/scrapy/issues/432])
- Use importlib for parametric imports (issue 445 [https://github.com/scrapy/scrapy/issues/445])
- Handle a regression introduced in Python 2.7.5 that affects XmlItemExporter (issue 372 [https://github.com/scrapy/scrapy/issues/372])
- Bugfix crawling shutdown on SIGINT (issue 450 [https://github.com/scrapy/scrapy/issues/450])
- Do not submit *reset* type inputs in FormRequest.from_response (commit b326b87 [https://github.com/scrapy/scrapy/commit/b326b87])
- Do not silence download errors when request errback raises an exception (commit 684cfc0 [https://github.com/scrapy/scrapy/commit/684cfc0])

Bugfixes

- Fix tests under Django 1.6 (commit b6bed44c [https://github.com/scrapy/scrapy/commit/b6bed44c])

- Lot of bugfixes to retry middleware under disconnections using HTTP 1.1 download handler
- Fix inconsistencies among Twisted releases (issue 406 [https://github.com/scrapy/scrapy/issues/406])
- Fix scrapy shell bugs (issue 418 [https://github.com/scrapy/scrapy/issues/418], issue 407 [https://github.com/scrapy/scrapy/issues/407])
- Fix invalid variable name in setup.py (issue 429 [https://github.com/scrapy/scrapy/issues/429])
- Fix tutorial references (issue 387 [https://github.com/scrapy/scrapy/issues/387])
- Improve request-response docs (issue 391 [https://github.com/scrapy/scrapy/issues/391])
- Improve best practices docs (issue 399 [https://github.com/scrapy/scrapy/issues/399], issue 400 [https://github.com/scrapy/scrapy/issues/400], issue 401 [https://github.com/scrapy/scrapy/issues/401], issue 402 [https://github.com/scrapy/scrapy/issues/402])
- Improve django integration docs (issue 404 [https://github.com/scrapy/scrapy/issues/404])
- Document *bindaddress* request meta (commit 37c24e01d7 [https://github.com/scrapy/scrapy/commit/37c24e01d7])
- Improve *Request* class documentation (issue 226 [https://github.com/scrapy/scrapy/issues/226])

Other

- Dropped Python 2.6 support (issue 448 [https://github.com/scrapy/scrapy/issues/448])
- Add cssselect [https://github.com/SimonSapin/cssselect] python package as install dependency
- Drop libxml2 and multi selector's backend support, lxml [http://lxml.de/] is required from now on.
- Minimum Twisted version increased to 10.0.0, dropped Twisted 8.0 support.
- Running test suite now requires *mock* python library (issue 390 [https://github.com/scrapy/scrapy/issues/390])

Thanks

Thanks to everyone who contribute to this release!

List of contributors sorted by number of commits:

```
69 Daniel Graña <dangra@...>
37 Pablo Hoffman <pablo@...>
13 Mikhail Korobov <kmike84@...>
 9 Alex Cepoi <alex.cepoi@...>
 9 alexanderlukanin13 <alexander.lukanin.13@...>
 8 Rolando Espinoza La fuente <darkrho@...>
 8 Lukasz Biedrycki <lukasz.biedrycki@...>
 6 Nicolas Ramirez <nramirez.uy@...>
 3 Paul Tremberth <paul.tremberth@...>
 2 Martin Olveyra <molveyra@...>
 2 Stefan <misc@...>
 2 Rolando Espinoza <darkrho@...>
 2 Loren Davie <loren@...>
 2 irgmedeiros <irgmedeiros@...>
 1 Stefan Koch <taikano@...>
 1 Stefan <cct@...>
 1 scraperdragon <dragon@...>
 1 Kumara Tharmalingam <ktharmal@...>
 1 Francesco Piccinno <stack.box@...>
 1 Marcos Campal <duendex@...>
 1 Dragon Dave <dragon@...>
 1 Capi Etheriel <barraponto@...>
 1 cacovsky <amarquesferraz@...>
 1 Berend Iwema <berend@...>
```

0.18.4 (released 2013-10-10)

- IPython refuses to update the namespace. fix #396 (commit 3d32c4f [https://github.com/scrapy/scrapy/commit/3d32c4f])
- Fix AlreadyCalledError replacing a request in shell command. closes #407 (commit b1d8919 [https://github.com/scrapy/scrapy/commit/b1d8919])
- Fix start_requests laziness and early hangs (commit 89faf52 [https://github.com/scrapy/scrapy/commit/89faf52])

0.18.3 (released 2013-10-03)

- fix regression on lazy evaluation of start requests (commit 12693a5 [https://github.com/scrapy/scrapy/commit/12693a5])
- forms: do not submit reset inputs (commit e429f63 [https://github.com/scrapy/scrapy/commit/e429f63])
- increase unittest timeouts to decrease travis false positive failures (commit 912202e [https://github.com/scrapy/scrapy/commit/912202e])
- backport master fixes to json exporter (commit cfc2d46 [https://github.com/scrapy/scrapy/commit/cfc2d46])
- Fix permission and set umask before generating sdist tarball (commit 06149e0 [https://github.com/scrapy/scrapy/commit/06149e0])

0.18.2 (released 2013-09-03)

- Backport *scrapy check* command fixes and backward compatible multi crawler process(issue 339 [https://github.com/scrapy/scrapy/issues/339])

0.18.1 (released 2013-08-27)

- remove extra import added by cherry picked changes (commit d20304e [https://github.com/scrapy/scrapy/commit/d20304e])
- fix crawling tests under twisted pre 11.0.0 (commit 1994f38 [https://github.com/scrapy/scrapy/commit/1994f38])

- py26 can not format zero length fields {} (commit abf756f [https://github.com/scrapy/scrapy/commit/abf756f])
- test PotentiaDataLoss errors on unbound responses (commit b15470d [https://github.com/scrapy/scrapy/commit/b15470d])
- Treat responses without content-length or Transfer-Encoding as good responses (commit c4bf324 [https://github.com/scrapy/scrapy/commit/c4bf324])
- do no include ResponseFailed if http11 handler is not enabled (commit 6cbe684 [https://github.com/scrapy/scrapy/commit/6cbe684])
- New HTTP client wraps connection losts in ResponseFailed exception. fix #373 (commit 1a20bba [https://github.com/scrapy/scrapy/commit/1a20bba])
- limit travis-ci build matrix (commit 3b01bb8 [https://github.com/scrapy/scrapy/commit/3b01bb8])
- Merge pull request #375 from peterarenot/patch-1 (commit fa766d7 [https://github.com/scrapy/scrapy/commit/fa766d7])
- Fixed so it refers to the correct folder (commit 3283809 [https://github.com/scrapy/scrapy/commit/3283809])
- added quantal & raring to support ubuntu releases (commit 1411923 [https://github.com/scrapy/scrapy/commit/1411923])
- fix retry middleware which didn't retry certain connection errors after the upgrade to http1 client, closes GH-373 (commit bb35ed0 [https://github.com/scrapy/scrapy/commit/bb35ed0])
- fix XmlItemExporter in Python 2.7.4 and 2.7.5 (commit de3e451 [https://github.com/scrapy/scrapy/commit/de3e451])
- minor updates to 0.18 release notes (commit c45e5f1 [https://github.com/scrapy/scrapy/commit/c45e5f1])
- fix contributters list format (commit 0b60031 [https://github.com/scrapy/scrapy/commit/0b60031])

0.18.0 (released 2013-08-09)

- Lot of improvements to testsuite run using Tox, including a way to test on pypi
- Handle GET parameters for AJAX crawleable urls (commit 3fe2a32 [https://github.com/scrapy/scrapy/commit/3fe2a32])
- Use lxml recover option to parse sitemaps (issue 347 [https://github.com/scrapy/scrapy/issues/347])
- Bugfix cookie merging by hostname and not by netloc (issue 352 [https://github.com/scrapy/scrapy/issues/352])
- Support disabling *HttpCompressionMiddleware* using a flag setting (issue 359 [https://github.com/scrapy/scrapy/issues/359])
- Support xml namespaces using *iternodes* parser in *XMLFeedSpider* (issue 12 [https://github.com/scrapy/scrapy/issues/12])
- Support *dont_cache* request meta flag (issue 19 [https://github.com/scrapy/scrapy/issues/19])
- Bugfix *scrapy.utils.gz.gunzip* broken by changes in python 2.7.4 (commit 4dc76e [https://github.com/scrapy/scrapy/commit/4dc76e])
- Bugfix url encoding on *SgmlLinkExtractor* (issue 24 [https://github.com/scrapy/scrapy/issues/24])
- Bugfix *TakeFirst* processor shouldn't discard zero (0) value (issue 59 [https://github.com/scrapy/scrapy/issues/59])
- Support nested items in xml exporter (issue 66 [https://github.com/scrapy/scrapy/issues/66])
- Improve cookies handling performance (issue 77 [https://github.com/scrapy/scrapy/issues/77])
- Log dupe filtered requests once (issue 105 [https://github.com/scrapy/scrapy/issues/105])
- Split redirection middleware into status and meta based middlewares (issue 78 [https://github.com/scrapy/scrapy/issues/78])
- Use HTTP1.1 as default downloader handler (issue 109 [https://github.com/scrapy/scrapy/issues/109] and issue 318 [https://github.com/scrapy/scrapy/issues/318])
- Support xpath form selection on *FormRequest.from_response* (issue 185 [https://github.com/scrapy/scrapy/issues/185])
- Bugfix unicode decoding error on *SgmlLinkExtractor* (issue 199 [https://github.com/scrapy/scrapy/issues/199])
- Bugfix signal dispatching on pypi interpreter (issue 205 [https://github.com/scrapy/scrapy/issues/205])
- Improve request delay and concurrency handling (issue 206 [https://github.com/scrapy/scrapy/issues/206])
- Add RFC2616 cache policy to *HttpCacheMiddleware* (issue 212 [https://github.com/scrapy/scrapy/issues/212])
- Allow customization of messages logged by engine (issue 214 [https://github.com/scrapy/scrapy/issues/214])
- Multiples improvements to *DjangoItem* (issue 217 [https://github.com/scrapy/scrapy/issues/217], issue 218 [https://github.com/scrapy/scrapy/issues/218], issue 221 [https://github.com/scrapy/scrapy/issues/221])
- Extend Scrapy commands using setuptools entry points (issue 260 [https://github.com/scrapy/scrapy/issues/260])
- Allow spider *allowed_domains* value to be set/tuple (issue 261 [https://github.com/scrapy/scrapy/issues/261])
- Support *settings.getdict* (issue 269 [https://github.com/scrapy/scrapy/issues/269])
- Simplify internal *scrapy.core.scraper* slot handling (issue 271 [https://github.com/scrapy/scrapy/issues/271])
- Added *Item.copy* (issue 290 [https://github.com/scrapy/scrapy/issues/290])
- Collect idle downloader slots (issue 297 [https://github.com/scrapy/scrapy/issues/297])
- Add *ftp://* scheme downloader handler (issue 329 [https://github.com/scrapy/scrapy/issues/329])
- Added downloader benchmark webserver and spider tools *Benchmarking*
- Moved persistent (on disk) queues to a separate project (queuelib [https://github.com/scrapy/queuelib]) which scrapy now depends on
- Add scrapy commands using external libraries (issue 260 [https://github.com/scrapy/scrapy/issues/260])
- Added `--pdb` option to `scrapy` command line tool
- Added **`XPathSelector.remove_namespaces()`** which allows to remove all namespaces from XML documents for convenience (to work with namespace-less XPaths). Documented in *选择器(Selectors)*.
- Several improvements to spider contracts
- New default middleware named MetaRefreshMiddldeware that handles meta-refresh html tag redirections,
- MetaRefreshMiddldeware and RedirectMiddleware have different priorities to address #62
- added from_crawler method to spiders
- added system tests with mock server
- more improvements to Mac OS compatibility (thanks Alex Cepoi)
- several more cleanups to singletons and multi-spider support (thanks Nicolas Ramirez)
- support custom download slots
- added –spider option to "shell" command.
- log overridden settings when scrapy starts

Thanks to everyone who contribute to this release. Here is a list of contributors sorted by number of commits:

```
130 Pablo Hoffman <pablo@...>
 97 Daniel Graña <dangra@...>
 20 Nicolás Ramírez <nramirez.uy@...>
 13 Mikhail Korobov <kmike84@...>
 12 Pedro Faustino <pedrobandim@...>
```

```
11 Steven Almeroth <sroth77@...>
 5 Rolando Espinoza La fuente <darkrho@...>
 4 Michal Danilak <mimino.coder@...>
 4 Alex Cepoi <alex.cepoi@...>
 4 Alexandr N Zamaraev (aka tonal) <tonal@...>
 3 paul <paul.tremberth@...>
 3 Martin Olveyra <molveyra@...>
 3 Jordi Llonch <llonchj@...>
 3 arijitchakraborty <myself.arijit@...>
 2 Shane Evans <shane.evans@...>
 2 joehillen <joehillen@...>
 2 Hart <HartSimha@...>
 2 Dan <ellisd23@...>
 1 Zuhao Wan <wanzuhao@...>
 1 whodatninja <blake@...>
 1 vkrest <v.krestiannykov@...>
 1 tpeng <pengtaoo@...>
 1 Tom Mortimer-Jones <tom@...>
 1 Rocio Aramberri <roschegel@...>
 1 Pedro <pedro@...>
 1 notsobad <wangxiaohugg@...>
 1 Natan L <kuyanatan.nlao@...>
 1 Mark Grey <mark.grey@...>
 1 Luan <luanpab@...>
 1 Libor Nenadál <libor.nenadal@...>
 1 Juan M Uys <opyate@...>
 1 Jonas Brunsgaard <jonas.brunsgaard@...>
 1 Ilya Baryshev <baryshev@...>
 1 Hasnain Lakhani <m.hasnain.lakhani@...>
 1 Emanuel Schorsch <emschorsch@...>
 1 Chris Tilden <chris.tilden@...>
 1 Capi Etheriel <barraponto@...>
 1 cacovsky <amarquesferraz@...>
 1 Berend Iwema <berend@...>
```

0.16.5 (released 2013-05-30)

- obey request method when scrapy deploy is redirected to a new endpoint (commit 8c4fcee [https://github.com/scrapy/scrapy/commit/8c4fcee])
- fix inaccurate downloader middleware documentation. refs #280 (commit 40667cb [https://github.com/scrapy/scrapy/commit/40667cb])
- doc: remove links to diveintopython.org, which is no longer available. closes #246 (commit bd58bfa [https://github.com/scrapy/scrapy/commit/bd58bfa])
- Find form nodes in invalid html5 documents (commit e3d6945 [https://github.com/scrapy/scrapy/commit/e3d6945])
- Fix typo labeling attrs type bool instead of list (commit a274276 [https://github.com/scrapy/scrapy/commit/a274276])

0.16.4 (released 2013-01-23)

- fixes spelling errors in documentation (commit 6d2b3aa [https://github.com/scrapy/scrapy/commit/6d2b3aa])
- add doc about disabling an extension. refs #132 (commit c90de33 [https://github.com/scrapy/scrapy/commit/c90de33])
- Fixed error message formatting. log.err() doesn't support cool formatting and when error occurred, the message was: "ERROR: Error processing %(item)s" (commit c16150c [https://github.com/scrapy/scrapy/commit/c16150c])
- lint and improve images pipeline error logging (commit 56b45fc [https://github.com/scrapy/scrapy/commit/56b45fc])
- fixed doc typos (commit 243be84 [https://github.com/scrapy/scrapy/commit/243be84])
- add documentation topics: Broad Crawls & Common Practies (commit 1fbb715 [https://github.com/scrapy/scrapy/commit/1fbb715])
- fix bug in scrapy parse command when spider is not specified explicitly. closes #209 (commit c72e682 [https://github.com/scrapy/scrapy/commit/c72e682])
- Update docs/topics/commands.rst (commit 28eac7a [https://github.com/scrapy/scrapy/commit/28eac7a])

0.16.3 (released 2012-12-07)

- Remove concurrency limitation when using download delays and still ensure inter-request delays are enforced (commit 487b9b5 [https://github.com/scrapy/scrapy/commit/487b9b5])
- add error details when image pipeline fails (commit 8232569 [https://github.com/scrapy/scrapy/commit/8232569])
- improve mac os compatibility (commit 8dcf8aa [https://github.com/scrapy/scrapy/commit/8dcf8aa])
- setup.py: use README.rst to populate long_description (commit 7b5310d [https://github.com/scrapy/scrapy/commit/7b5310d])
- doc: removed obsolete references to ClientForm (commit 80f9bb6 [https://github.com/scrapy/scrapy/commit/80f9bb6])
- correct docs for default storage backend (commit 2aa491b [https://github.com/scrapy/scrapy/commit/2aa491b])
- doc: removed broken proxyhub link from FAQ (commit bdf61c4 [https://github.com/scrapy/scrapy/commit/bdf61c4])
- Fixed docs typo in SpiderOpenCloseLogging example (commit 7184094 [https://github.com/scrapy/scrapy/commit/7184094])

0.16.2 (released 2012-11-09)

- scrapy contracts: python2.6 compat (commit a4a9199 [https://github.com/scrapy/scrapy/commit/a4a9199])
- scrapy contracts verbose option (commit ec41673 [https://github.com/scrapy/scrapy/commit/ec41673])
- proper unittest-like output for scrapy contracts (commit 86635e4 [https://github.com/scrapy/scrapy/commit/86635e4])
- added open_in_browser to debugging doc (commit c9b690d [https://github.com/scrapy/scrapy/commit/c9b690d])
- removed reference to global scrapy stats from settings doc (commit dd55067 [https://github.com/scrapy/scrapy/commit/dd55067])
- Fix SpiderState bug in Windows platforms (commit 58998f4 [https://github.com/scrapy/scrapy/commit/58998f4])

0.16.1 (released 2012-10-26)

- fixed LogStats extension, which got broken after a wrong merge before the 0.16 release (commit 8c780fd [https://github.com/scrapy/scrapy/commit/8c780fd])
- better backwards compatibility for scrapy.conf.settings (commit 3403089 [https://github.com/scrapy/scrapy/commit/3403089])
- extended documentation on how to access crawler stats from extensions (commit c4da0b5 [https://github.com/scrapy/scrapy/commit/c4da0b5])
- removed .hgtags (no longer needed now that scrapy uses git) (commit d52c188 [https://github.com/scrapy/scrapy/commit/d52c188])
- fix dashes under rst headers (commit fa4f7f9 [https://github.com/scrapy/scrapy/commit/fa4f7f9])
- set release date for 0.16.0 in news (commit e292246 [https://github.com/scrapy/scrapy/commit/e292246])

0.16.0 (released 2012-10-18)

Scrapy changes:

- added *Spiders Contracts*, a mechanism for testing spiders in a formal/reproducible way
- added options `-o` and `-t` to the **`runspider`** command
- documented *自动限速(AutoThrottle)扩展* and added to extensions installed by default. You still need to enable it with **`AUTOTHROTTLE_ENABLED`**
- major Stats Collection refactoring: removed separation of global/per-spider stats, removed stats-related signals (`stats_spider_opened`, etc). Stats are much simpler now, backwards compatibility is kept on the Stats Collector API and signals.
- added **`process_start_requests()`** method to spider middlewares
- dropped Signals singleton. Signals should now be accesed through the Crawler.signals attribute. See the signals documentation for more info.
- dropped Signals singleton. Signals should now be accesed through the Crawler.signals attribute. See the signals documentation for more info.
- dropped Stats Collector singleton. Stats can now be accessed through the Crawler.stats attribute. See the stats collection documentation for more info.
- documented *核心API*
- *lxml* is now the default selectors backend instead of *libxml2*
- ported FormRequest.from_response() to use lxml [http://lxml.de/] instead of ClientForm [http://wwwsearch.sourceforge.net/old/ClientForm/]
- removed modules: `scrapy.xlib.BeautifulSoup` and `scrapy.xlib.ClientForm`
- SitemapSpider: added support for sitemap urls ending in .xml and .xml.gz, even if they advertise a wrong content type (commit 10ed28b [https://github.com/scrapy/scrapy/commit/10ed28b])
- StackTraceDump extension: also dump trackref live references (commit fe2ce93 [https://github.com/scrapy/scrapy/commit/fe2ce93])
- nested items now fully supported in JSON and JSONLines exporters
- added **`cookiejar`** Request meta key to support multiple cookie sessions per spider
- decoupled encoding detection code to w3lib.encoding [https://github.com/scrapy/w3lib/blob/master/w3lib/encoding.py], and ported Scrapy code to use that mdule
- dropped support for Python 2.5. See http://blog.scrapinghub.com/2012/02/27/scrapy-0-15-dropping-support-for-python-2-5/
- dropped support for Twisted 2.5
- added **`REFERER_ENABLED`** setting, to control referer middleware
- changed default user agent to: `Scrapy/VERSION (+http://scrapy.org)`
- removed (undocumented) `HTMLImageLinkExtractor` class from `scrapy.contrib.linkextractors.image`
- removed per-spider settings (to be replaced by instantiating multiple crawler objects)
- `USER_AGENT` spider attribute will no longer work, use `user_agent` attribute instead
- `DOWNLOAD_TIMEOUT` spider attribute will no longer work, use `download_timeout` attribute instead
- removed `ENCODING_ALIASES` setting, as encoding auto-detection has been moved to the w3lib [https://github.com/scrapy/w3lib] library
- promoted *Djangoltem* to main contrib
- LogFormatter method now return dicts(instead of strings) to support lazy formatting (issue 164 [https://github.com/scrapy/scrapy/issues/164], commit dcef7b0 [https://github.com/scrapy/scrapy/commit/dcef7b0])
- downloader handlers (**`DOWNLOAD_HANDLERS`** setting) now receive settings as the first argument of the constructor
- replaced memory usage acounting with (more portable) resource [http://docs.python.org/library/resource.html] module, removed `scrapy.utils.memory` module
- removed signal: `scrapy.mail.mail_sent`
- removed `TRACK_REFS` setting, now *trackrefs* is always enabled
- DBM is now the default storage backend for HTTP cache middleware
- number of log messages (per level) are now tracked through Scrapy stats (stat name: `log_count/LEVEL`)
- number received responses are now tracked through Scrapy stats (stat name: `response_received_count`)
- removed `scrapy.log.started` attribute

0.14.4

- added precise to supported ubuntu distros (commit b7e46df [https://github.com/scrapy/scrapy/commit/b7e46df])
- fixed bug in json-rpc webservice reported in https://groups.google.com/d/topic/scrapy-users/qgVBmFybNAQ/discussion. also removed no longer supported 'run' command from extras/scrapy-ws.py (commit 340fbdb [https://github.com/scrapy/scrapy/commit/340fbdb])
- meta tag attributes for content-type http equiv can be in any order. #123 (commit 0cb68af [https://github.com/scrapy/scrapy/commit/0cb68af])
- replace "import Image" by more standard "from PIL import Image". closes #88 (commit 4d17048 [https://github.com/scrapy/scrapy/commit/4d17048])
- return trial status as bin/runtests.sh exit value. #118 (commit b7b2e7f [https://github.com/scrapy/scrapy/commit/b7b2e7f])

0.14.3

- forgot to include pydispatch license. #118 (commit fd85f9c [https://github.com/scrapy/scrapy/commit/fd85f9c])
- include egg files used by testsuite in source distribution. #118 (commit c897793 [https://github.com/scrapy/scrapy/commit/c897793])
- update docstring in project template to avoid confusion with genspider command, which may be considered as an advanced feature. refs #107 (commit 2548dcc [https://github.com/scrapy/scrapy/commit/2548dcc])
- added note to docs/topics/firebug.rst about google directory being shut down (commit 668e352 [https://github.com/scrapy/scrapy/commit/668e352])
- dont discard slot when empty, just save in another dict in order to recycle if needed again. (commit 8e9f607 [https://github.com/scrapy/scrapy/commit/8e9f607])
- do not fail handling unicode xpaths in libxml2 backed selectors (commit b830e95 [https://github.com/scrapy/scrapy/commit/b830e95])
- fixed minor mistake in Request objects documentation (commit bf3c9ee [https://github.com/scrapy/scrapy/commit/bf3c9ee])
- fixed minor defect in link extractors documentation (commit ba14f38 [https://github.com/scrapy/scrapy/commit/ba14f38])
- removed some obsolete remaining code related to sqlite support in scrapy (commit 0665175 [https://github.com/scrapy/scrapy/commit/0665175])

0.14.2

- move buffer pointing to start of file before computing checksum. refs #92 (commit 6a5bef2 [https://github.com/scrapy/scrapy/commit/6a5bef2])
- Compute image checksum before persisting images. closes #92 (commit 9817df1 [https://github.com/scrapy/scrapy/commit/9817df1])
- remove leaking references in cached failures (commit 673a120 [https://github.com/scrapy/scrapy/commit/673a120])
- fixed bug in MemoryUsage extension: get_engine_status() takes exactly 1 argument (0 given) (commit 11133e9 [https://github.com/scrapy/scrapy/commit/11133e9])
- fixed struct.error on http compression middleware. closes #87 (commit 1423140 [https://github.com/scrapy/scrapy/commit/1423140])
- ajax crawling wasn't expanding for unicode urls (commit 0de3fb4 [https://github.com/scrapy/scrapy/commit/0de3fb4])
- Catch start_requests iterator errors. refs #83 (commit 454a21d [https://github.com/scrapy/scrapy/commit/454a21d])
- Speed-up libxml2 XPathSelector (commit 2fbd662 [https://github.com/scrapy/scrapy/commit/2fbd662])
- updated versioning doc according to recent changes (commit 0a070f5 [https://github.com/scrapy/scrapy/commit/0a070f5])
- scrapyd: fixed documentation link (commit 2b4e4c3 [https://github.com/scrapy/scrapy/commit/2b4e4c3])
- extras/makedeb.py: no longer obtaining version from git (commit caffe0e [https://github.com/scrapy/scrapy/commit/caffe0e])

0.14.1

- extras/makedeb.py: no longer obtaining version from git (commit caffe0e [https://github.com/scrapy/scrapy/commit/caffe0e])
- bumped version to 0.14.1 (commit 6cb9e1c [https://github.com/scrapy/scrapy/commit/6cb9e1c])
- fixed reference to tutorial directory (commit 4b86bd6 [https://github.com/scrapy/scrapy/commit/4b86bd6])
- doc: removed duplicated callback argument from Request.replace() (commit 1aeccdd [https://github.com/scrapy/scrapy/commit/1aeccdd])
- fixed formatting of scrapyd doc (commit 8bf19e6 [https://github.com/scrapy/scrapy/commit/8bf19e6])
- Dump stacks for all running threads and fix engine status dumped by StackTraceDump extension (commit 14a8e6e [https://github.com/scrapy/scrapy/commit/14a8e6e])
- added comment about why we disable ssl on boto images upload (commit 5223575 [https://github.com/scrapy/scrapy/commit/5223575])
- SSL handshaking hangs when doing too many parallel connections to S3 (commit 63d583d [https://github.com/scrapy/scrapy/commit/63d583d])
- change tutorial to follow changes on dmoz site (commit bcb3198 [https://github.com/scrapy/scrapy/commit/bcb3198])
- Avoid _disconnectedDeferred AttributeError exception in Twisted>=11.1.0 (commit 98f3f87 [https://github.com/scrapy/scrapy/commit/98f3f87])
- allow spider to set autothrottle max concurrency (commit 175a4b5 [https://github.com/scrapy/scrapy/commit/175a4b5])

0.14

New features and settings

- Support for AJAX crawleable urls [http://code.google.com/web/ajaxcrawling/docs/getting-started.html]
- New persistent scheduler that stores requests on disk, allowing to suspend and resume crawls (r2737 [http://hg.scrapy.org/scrapy/changeset/2737])
- added `-o` option to `scrapy crawl`, a shortcut for dumping scraped items into a file (or standard output using `-`)
- Added support for passing custom settings to Scrapyd `schedule.json` api (r2779 [http://hg.scrapy.org/scrapy/changeset/2779], r2783 [http://hg.scrapy.org/scrapy/changeset/2783])
- New `ChunkedTransferMiddleware` (enabled by default) to support chunked transfer encoding [http://en.wikipedia.org/wiki/Chunked_transfer_encoding] (r2769 [http://hg.scrapy.org/scrapy/changeset/2769])
- Add boto 2.0 support for S3 downloader handler (r2763 [http://hg.scrapy.org/scrapy/changeset/2763])
- Added marshal [http://docs.python.org/library/marshal.html] to formats supported by feed exports (r2744 [http://hg.scrapy.org/scrapy/changeset/2744])
- In request errbacks, offending requests are now received in *failure.request* attribute (r2738 [http://hg.scrapy.org/scrapy/changeset/2738])

- Big downloader refactoring to support per domain/ip concurrency limits (r2732 [http://hg.scrapy.org/scrapy/changeset/2732])
    - `CONCURRENT_REQUESTS_PER_SPIDER` setting has been deprecated and replaced by:
        - `CONCURRENT_REQUESTS`, `CONCURRENT_REQUESTS_PER_DOMAIN`, `CONCURRENT_REQUESTS_PER_IP`
    - check the documentation for more details
- Added builtin caching DNS resolver (r2728 [http://hg.scrapy.org/scrapy/changeset/2728])
- Moved Amazon AWS-related components/extensions (SQS spider queue, SimpleDB stats collector) to a separate project: [scaws](https://github.com/scrapinghub/scaws) (r2706 [http://hg.scrapy.org/scrapy/changeset/2706], r2714 [http://hg.scrapy.org/scrapy/changeset/2714])
- Moved spider queues to scrapyd: *scrapy.spiderqueue -> scrapyd.spiderqueue* (r2708 [http://hg.scrapy.org/scrapy/changeset/2708])
- Moved sqlite utils to scrapyd: *scrapy.utils.sqlite -> scrapyd.sqlite* (r2781 [http://hg.scrapy.org/scrapy/changeset/2781])
- Real support for returning iterators on *start_requests()* method. The iterator is now consumed during the crawl when the spider is getting idle (r2704 [http://hg.scrapy.org/scrapy/changeset/2704])
- Added `REDIRECT_ENABLED` setting to quickly enable/disable the redirect middleware (r2697 [http://hg.scrapy.org/scrapy/changeset/2697])
- Added `RETRY_ENABLED` setting to quickly enable/disable the retry middleware (r2694 [http://hg.scrapy.org/scrapy/changeset/2694])
- Added `CloseSpider` exception to manually close spiders (r2691 [http://hg.scrapy.org/scrapy/changeset/2691])
- Improved encoding detection by adding support for HTML5 meta charset declaration (r2690 [http://hg.scrapy.org/scrapy/changeset/2690])
- Refactored close spider behavior to wait for all downloads to finish and be processed by spiders, before closing the spider (r2688 [http://hg.scrapy.org/scrapy/changeset/2688])
- Added `SitemapSpider` (see documentation in Spiders page) (r2658 [http://hg.scrapy.org/scrapy/changeset/2658])
- Added `LogStats` extension for periodically logging basic stats (like crawled pages and scraped items) (r2657 [http://hg.scrapy.org/scrapy/changeset/2657])
- Make handling of gzipped responses more robust (#319, r2643 [http://hg.scrapy.org/scrapy/changeset/2643]). Now Scrapy will try and decompress as much as possible from a gzipped response, instead of failing with an *IOError*.
- Simplified !MemoryDebugger extension to use stats for dumping memory debugging info (r2639 [http://hg.scrapy.org/scrapy/changeset/2639])
- Added new command to edit spiders: `scrapy edit` (r2636 [http://hg.scrapy.org/scrapy/changeset/2636]) and *-e* flag to *genspider* command that uses it (r2653 [http://hg.scrapy.org/scrapy/changeset/2653])
- Changed default representation of items to pretty-printed dicts. (r2631 [http://hg.scrapy.org/scrapy/changeset/2631]). This improves default logging by making log more readable in the default case, for both Scraped and Dropped lines.
- Added `spider_error` signal (r2628 [http://hg.scrapy.org/scrapy/changeset/2628])
- Added `COOKIES_ENABLED` setting (r2625 [http://hg.scrapy.org/scrapy/changeset/2625])
- Stats are now dumped to Scrapy log (default value of `STATS_DUMP` setting has been changed to *True*). This is to make Scrapy users more aware of Scrapy stats and the data that is collected there.
- Added support for dynamically adjusting download delay and maximum concurrent requests (r2599 [http://hg.scrapy.org/scrapy/changeset/2599])
- Added new DBM HTTP cache storage backend (r2576 [http://hg.scrapy.org/scrapy/changeset/2576])
- Added `listjobs.json` API to Scrapyd (r2571 [http://hg.scrapy.org/scrapy/changeset/2571])
- `CsvItemExporter`: added `join_multivalued` parameter (r2578 [http://hg.scrapy.org/scrapy/changeset/2578])
- Added namespace support to `xmliter_lxml` (r2552 [http://hg.scrapy.org/scrapy/changeset/2552])
- Improved cookies middleware by making *COOKIES_DEBUG* nicer and documenting it (r2579 [http://hg.scrapy.org/scrapy/changeset/2579])
- Several improvements to Scrapyd and Link extractors

Code rearranged and removed

- Merged item passed and item scraped concepts, as they have often proved confusing in the past. This means: (r2630 [http://hg.scrapy.org/scrapy/changeset/2630])
    - original item_scraped signal was removed
    - original item_passed signal was renamed to item_scraped
    - old log lines `Scraped Item...` were removed
    - old log lines `Passed Item...` were renamed to `Scraped Item...` lines and downgraded to `DEBUG` level

- Reduced Scrapy codebase by striping part of Scrapy code into two new libraries:
    - w3lib [https://github.com/scrapy/w3lib] (several functions from `scrapy.utils.{http,markup,multipart,response,url}`, done in r2584 [http://hg.scrapy.org/scrapy/changeset/2584])
    - scrapely [https://github.com/scrapy/scrapely] (was `scrapy.contrib.ibl`, done in r2586 [http://hg.scrapy.org/scrapy/changeset/2586])
- Removed unused function: *scrapy.utils.request.request_info()* (r2577 [http://hg.scrapy.org/scrapy/changeset/2577])
- Removed googledir project from *examples/googledir*. There's now a new example project called *dirbot* available on github: https://github.com/scrapy/dirbot
- Removed support for default field values in Scrapy items (r2616 [http://hg.scrapy.org/scrapy/changeset/2616])
- Removed experimental crawlspider v2 (r2632 [http://hg.scrapy.org/scrapy/changeset/2632])
- Removed scheduler middleware to simplify architecture. Duplicates filter is now done in the scheduler itself, using the same dupe fltering class as before (*DUPEFILTER_CLASS* setting) (r2640 [http://hg.scrapy.org/scrapy/changeset/2640])
- Removed support for passing urls to `scrapy crawl` command (use `scrapy parse` instead) (r2704 [http://hg.scrapy.org/scrapy/changeset/2704])
- Removed deprecated Execution Queue (r2704 [http://hg.scrapy.org/scrapy/changeset/2704])
- Removed (undocumented) spider context extension (from scrapy.contrib.spidercontext) (r2780 [http://hg.scrapy.org/scrapy/changeset/2780])
- removed `CONCURRENT_SPIDERS` setting (use scrapyd maxproc instead) (r2789 [http://hg.scrapy.org/scrapy/changeset/2789])
- Renamed attributes of core components: downloader.sites -> downloader.slots, scraper.sites -> scraper.slots (r2717 [http://hg.scrapy.org/scrapy/changeset/2717], r2718 [http://hg.scrapy.org/scrapy/changeset/2718])
- Renamed setting `CLOSESPIDER_ITEMPASSED` to **`CLOSESPIDER_ITEMCOUNT`** (r2655 [http://hg.scrapy.org/scrapy/changeset/2655]). Backwards compatibility kept.

0.12

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features and improvements

- Passed item is now sent in the `item` argument of the **`item_passed`** (#273)
- Added verbose option to `scrapy version` command, useful for bug reports (#298)
- HTTP cache now stored by default in the project data dir (#279)
- Added project data storage directory (#276, #277)
- Documented file structure of Scrapy projects (see command-line tool doc)
- New lxml backend for XPath selectors (#147)
- Per-spider settings (#245)
- Support exit codes to signal errors in Scrapy commands (#248)
- Added `-c` argument to `scrapy shell` command
- Made `libxml2` optional (#260)
- New `deploy` command (#261)
- Added **`CLOSESPIDER_PAGECOUNT`** setting (#253)
- Added **`CLOSESPIDER_ERRORCOUNT`** setting (#254)

Scrapyd changes

- Scrapyd now uses one process per spider
- It stores one log file per spider run, and rotate them keeping the lastest 5 logs per spider (by default)
- A minimal web ui was added, available at http://localhost:6800 by default
- There is now a *scrapy server* command to start a Scrapyd server of the current project

Changes to settings

- added *HTTPCACHE_ENABLED* setting (False by default) to enable HTTP cache middleware
- changed *HTTPCACHE_EXPIRATION_SECS* semantics: now zero means "never expire".

Deprecated/obsoleted functionality

- Deprecated `runserver` command in favor of `server` command which starts a Scrapyd server. See also: Scrapyd changes
- Deprecated `queue` command in favor of using Scrapyd `schedule.json` API. See also: Scrapyd changes
- Removed the !LxmlItemLoader (experimental contrib which never graduated to main contrib)

0.10

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features and improvements

- New Scrapy service called `scrapyd` for deploying Scrapy crawlers in production (#218) (documentation available)
- Simplified Images pipeline usage which doesn't require subclassing your own images pipeline now (#217)
- Scrapy shell now shows the Scrapy log by default (#206)
- Refactored execution queue in a common base code and pluggable backends called "spider queues" (#220)
- New persistent spider queue (based on SQLite) (#198), available by default, which allows to start Scrapy in server mode and then schedule spiders to run.
- Added documentation for Scrapy command-line tool and all its available sub-commands. (documentation available)
- Feed exporters with pluggable backends (#197) (documentation available)
- Deferred signals (#193)
- Added two new methods to item pipeline open_spider(), close_spider() with deferred support (#195)
- Support for overriding default request headers per spider (#181)
- Replaced default Spider Manager with one with similar functionality but not depending on Twisted Plugins (#186)
- Splitted Debian package into two packages - the library and the service (#187)
- Scrapy log refactoring (#188)
- New extension for keeping persistent spider contexts among different runs (#203)
- Added *dont_redirect* request.meta key for avoiding redirects (#233)
- Added *dont_retry* request.meta key for avoiding retries (#234)

Command-line tool changes

- New *scrapy* command which replaces the old *scrapy-ctl.py* (#199) - there is only one global *scrapy* command now, instead of one *scrapy-ctl.py* per project - Added *scrapy.bat* script for running more conveniently from Windows
- Added bash completion to command-line tool (#210)
- Renamed command *start* to *runserver* (#209)

API changes

- `url` and `body` attributes of Request objects are now read-only (#230)
- `Request.copy()` and `Request.replace()` now also copies their `callback` and `errback` attributes (#231)
- Removed `UrlFilterMiddleware` from `scrapy.contrib` (already disabled by default)
- Offsite middelware doesn't filter out any request coming from a spider that doesn't have a allowed_domains attribute (#225)
- Removed Spider Manager `load()` method. Now spiders are loaded in the constructor itself.
- Changes to Scrapy Manager (now called "Crawler"):
  - `scrapy.core.manager.ScrapyManager` class renamed to `scrapy.crawler.Crawler`
  - `scrapy.core.manager.scrapymanager` singleton moved to `scrapy.project.crawler`
- Moved module: `scrapy.contrib.spidermanager` to `scrapy.spidermanager`
- Spider Manager singleton moved from `scrapy.spider.spiders` to the `spiders` attribute of ``scrapy.project.crawler`` singleton.
- moved Stats Collector classes: (#204)
  - `scrapy.stats.collector.StatsCollector` to `scrapy.statscol.StatsCollector`
  - `scrapy.stats.collector.SimpledbStatsCollector` to `scrapy.contrib.statscol.SimpledbStatsCollector`
- default per-command settings are now specified in the `default_settings` attribute of command object class (#201)
- changed arguments of Item pipeline `process_item()` method from `(spider, item)` to `(item, spider)`
  - backwards compatibility kept (with deprecation warning)
- moved `scrapy.core.signals` module to `scrapy.signals`
  - backwards compatibility kept (with deprecation warning)
- moved `scrapy.core.exceptions` module to `scrapy.exceptions`
  - backwards compatibility kept (with deprecation warning)
- added `handles_request()` class method to `BaseSpider`
- dropped `scrapy.log.exc()` function (use `scrapy.log.err()` instead)
- dropped `component` argument of `scrapy.log.msg()` function
- dropped `scrapy.log.log_level` attribute
- Added `from_settings()` class methods to Spider Manager, and Item Pipeline Manager

Changes to settings

- Added `HTTPCACHE_IGNORE_SCHEMES` setting to ignore certain schemes on !HttpCacheMiddleware (#225)
- Added `SPIDER_QUEUE_CLASS` setting which defines the spider queue to use (#220)
- Added `KEEP_ALIVE` setting (#220)
- Removed `SERVICE_QUEUE` setting (#220)
- Removed `COMMANDS_SETTINGS_MODULE` setting (#201)
- Renamed `REQUEST_HANDLERS` to `DOWNLOAD_HANDLERS` and make download handlers classes (instead of functions)

0.9

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features and improvements

- Added SMTP-AUTH support to scrapy.mail
- New settings added: `MAIL_USER`, `MAIL_PASS` (r2065 [http://hg.scrapy.org/scrapy/changeset/2065] | #149)
- Added new scrapy-ctl view command - To view URL in the browser, as seen by Scrapy (r2039 [http://hg.scrapy.org/scrapy/changeset/2039])
- Added web service for controlling Scrapy process (this also deprecates the web console. (r2053 [http://hg.scrapy.org/scrapy/changeset/2053] | #167)
- Support for running Scrapy as a service, for production systems (r1988 [http://hg.scrapy.org/scrapy/changeset/1988], r2054 [http://hg.scrapy.org/scrapy/changeset/2054], r2055 [http://hg.scrapy.org/scrapy/changeset/2055], r2056 [http://hg.scrapy.org/scrapy/changeset/2056], r2057 [http://hg.scrapy.org/scrapy/changeset/2057] | #168)
- Added wrapper induction library (documentation only available in source code for now). (r2011 [http://hg.scrapy.org/scrapy/changeset/2011])
- Simplified and improved response encoding support (r1961 [http://hg.scrapy.org/scrapy/changeset/1961], r1969 [http://hg.scrapy.org/scrapy/changeset/1969])
- Added `LOG_ENCODING` setting (r1956 [http://hg.scrapy.org/scrapy/changeset/1956], documentation available)
- Added `RANDOMIZE_DOWNLOAD_DELAY` setting (enabled by default) (r1923 [http://hg.scrapy.org/scrapy/changeset/1923], doc available)
- `MailSender` is no longer IO-blocking (r1955 [http://hg.scrapy.org/scrapy/changeset/1955] | #146)
- Linkextractors and new Crawlspider now handle relative base tag urls (r1960 [http://hg.scrapy.org/scrapy/changeset/1960] | #148)
- Several improvements to Item Loaders and processors (r2022 [http://hg.scrapy.org/scrapy/changeset/2022], r2023 [http://hg.scrapy.org/scrapy/changeset/2023], r2024 [http://hg.scrapy.org/scrapy/changeset/2024], r2025 [http://hg.scrapy.org/scrapy/changeset/2025], r2026 [http://hg.scrapy.org/scrapy/changeset/2026], r2027 [http://hg.scrapy.org/scrapy/changeset/2027], r2028 [http://hg.scrapy.org/scrapy/changeset/2028], r2029 [http://hg.scrapy.org/scrapy/changeset/2029], r2030 [http://hg.scrapy.org/scrapy/changeset/2030])
- Added support for adding variables to telnet console (r2047 [http://hg.scrapy.org/scrapy/changeset/2047] | #165)
- Support for requests without callbacks (r2050 [http://hg.scrapy.org/scrapy/changeset/2050] | #166)

API changes

- Change `Spider.domain_name` to `Spider.name` (SEP-012, r1975 [http://hg.scrapy.org/scrapy/changeset/1975])
- `Response.encoding` is now the detected encoding (r1961 [http://hg.scrapy.org/scrapy/changeset/1961])
- `HttpErrorMiddleware` now returns None or raises an exception (r2006 [http://hg.scrapy.org/scrapy/changeset/2006] | #157)
- `scrapy.command` modules relocation (r2035 [http://hg.scrapy.org/scrapy/changeset/2035], r2036 [http://hg.scrapy.org/scrapy/changeset/2036], r2037 [http://hg.scrapy.org/scrapy/changeset/2037])
- Added `ExecutionQueue` for feeding spiders to scrape (r2034 [http://hg.scrapy.org/scrapy/changeset/2034])
- Removed `ExecutionEngine` singleton (r2039 [http://hg.scrapy.org/scrapy/changeset/2039])
- Ported `S3ImagesStore` (images pipeline) to use boto and threads (r2033 [http://hg.scrapy.org/scrapy/changeset/2033])
- Moved module: `scrapy.management.telnet` to `scrapy.telnet` (r2047 [http://hg.scrapy.org/scrapy/changeset/2047])

Changes to default settings

- Changed default `SCHEDULER_ORDER` to `DFO` (r1939 [http://hg.scrapy.org/scrapy/changeset/1939])

0.8

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features

- Added DEFAULT_RESPONSE_ENCODING setting (r1809 [http://hg.scrapy.org/scrapy/changeset/1809])
- Added `dont_click` argument to `FormRequest.from_response()` method (r1813 [http://hg.scrapy.org/scrapy/changeset/1813], r1816 [http://hg.scrapy.org/scrapy/changeset/1816])
- Added `clickdata` argument to `FormRequest.from_response()` method (r1802 [http://hg.scrapy.org/scrapy/changeset/1802], r1803 [http://hg.scrapy.org/scrapy/changeset/1803])
- Added support for HTTP proxies (`HttpProxyMiddleware`) (r1781 [http://hg.scrapy.org/scrapy/changeset/1781], r1785 [http://hg.scrapy.org/scrapy/changeset/1785])
- Offiste spider middleware now logs messages when filtering out requests (r1841 [http://hg.scrapy.org/scrapy/changeset/1841])

Backwards-incompatible changes

- Changed `scrapy.utils.response.get_meta_refresh()` signature (r1804 [http://hg.scrapy.org/scrapy/changeset/1804])

- Removed deprecated `scrapy.item.ScrapedItem` class - use `scrapy.item.Item instead` (r1838 [http://hg.scrapy.org/scrapy/changeset/1838])
- Removed deprecated `scrapy.xpath` module - use `scrapy.selector` instead. (r1836 [http://hg.scrapy.org/scrapy/changeset/1836])
- Removed deprecated `core.signals.domain_open` signal - use `core.signals.domain_opened` instead (r1822 [http://hg.scrapy.org/scrapy/changeset/1822])
- `log.msg()` now receives a `spider` argument (r1822 [http://hg.scrapy.org/scrapy/changeset/1822])
    - Old domain argument has been deprecated and will be removed in 0.9. For spiders, you should always use the `spider` argument and pass spider references. If you really want to pass a string, use the `component` argument instead.
- Changed core signals `domain_opened`, `domain_closed`, `domain_idle`
- Changed Item pipeline to use spiders instead of domains
    - The `domain` argument of `process_item()` item pipeline method was changed to `spider`, the new signature is: `process_item(spider, item)` (r1827 [http://hg.scrapy.org/scrapy/changeset/1827] | #105)
    - To quickly port your code (to work with Scrapy 0.8) just use `spider.domain_name` where you previously used `domain`.
- Changed Stats API to use spiders instead of domains (r1849 [http://hg.scrapy.org/scrapy/changeset/1849] | #113)
    - `StatsCollector` was changed to receive spider references (instead of domains) in its methods (`set_value`, `inc_value`, etc).
    - added `StatsCollector.iter_spider_stats()` method
    - removed `StatsCollector.list_domains()` method
    - Also, Stats signals were renamed and now pass around spider references (instead of domains). Here's a summary of the changes:
    - To quickly port your code (to work with Scrapy 0.8) just use `spider.domain_name` where you previously used `domain`. `spider_stats` contains exactly the same data as `domain_stats`.
- `CloseDomain` extension moved to `scrapy.contrib.closespider.CloseSpider` (r1833 [http://hg.scrapy.org/scrapy/changeset/1833])
    - Its settings were also renamed:
        - `CLOSEDOMAIN_TIMEOUT` to `CLOSESPIDER_TIMEOUT`
        - `CLOSEDOMAIN_ITEMCOUNT` to `CLOSESPIDER_ITEMCOUNT`
- Removed deprecated `SCRAPYSETTINGS_MODULE` environment variable - use `SCRAPY_SETTINGS_MODULE` instead (r1840 [http://hg.scrapy.org/scrapy/changeset/1840])
- Renamed setting: `REQUESTS_PER_DOMAIN` to `CONCURRENT_REQUESTS_PER_SPIDER` (r1830 [http://hg.scrapy.org/scrapy/changeset/1830], r1844 [http://hg.scrapy.org/scrapy/changeset/1844])
- Renamed setting: `CONCURRENT_DOMAINS` to `CONCURRENT_SPIDERS` (r1830 [http://hg.scrapy.org/scrapy/changeset/1830])
- Refactored HTTP Cache middleware
- HTTP Cache middleware has been heavilty refactored, retaining the same functionality except for the domain sectorization which was removed. (r1843 [http://hg.scrapy.org/scrapy/changeset/1843])
- Renamed exception: `DontCloseDomain` to `DontCloseSpider` (r1859 [http://hg.scrapy.org/scrapy/changeset/1859] | #120)
- Renamed extension: `DelayedCloseDomain` to `SpiderCloseDelay` (r1861 [http://hg.scrapy.org/scrapy/changeset/1861] | #121)
- Removed obsolete `scrapy.utils.markup.remove_escape_chars` function - use `scrapy.utils.markup.replace_escape_chars` instead (r1865 [http://hg.scrapy.org/scrapy/changeset/1865])

0.7

First release of Scrapy.

讨论

Contributing to Scrapy

There are many ways to contribute to Scrapy. Here are some of them:

- Blog about Scrapy. Tell the world how you're using Scrapy. This will help newcomers with more examples and the Scrapy project to increase its visibility.
- Report bugs and request features in the issue tracker [https://github.com/scrapy/scrapy/issues], trying to follow the guidelines detailed in Reporting bugs below.
- Submit patches for new functionality and/or bug fixes. Please read Writing patches and Submitting patches below for details on how to write and submit a patch.
- Join the scrapy-users [http://groups.google.com/group/scrapy-users] mailing list and share your ideas on how to improve Scrapy. We're always open to suggestions.

Reporting bugs

注解

Please report security issues **only** to scrapy-security@googlegroups.com. This is a private list only open to trusted Scrapy developers, and its archives are not public.

Well-written bug reports are very helpful, so keep in mind the following guidelines when reporting a new bug.

- check the *FAQ* first to see if your issue is addressed in a well-known question
- check the open issues [https://github.com/scrapy/scrapy/issues] to see if it has already been reported. If it has, don't dismiss the report but check the ticket history and comments, you may find additional useful information to contribute.
- search the scrapy-users [http://groups.google.com/group/scrapy-users] list to see if it has been discussed there, or if you're not sure if what you're seeing is a bug. You can also ask in the *#scrapy* IRC channel.
- write complete, reproducible, specific bug reports. The smaller the test case, the better. Remember that other developers won't have your project to reproduce the bug, so please include all relevant files required to reproduce it.
- include the output of `scrapy version -v` so developers working on your bug know exactly which version and platform it occurred on, which is often very helpful for reproducing it, or knowing if it was already fixed.

Writing patches

The better written a patch is, the higher chance that it'll get accepted and the sooner that will be merged.

Well-written patches should:

- contain the minimum amount of code required for the specific change. Small patches are easier to review and merge. So, if you're doing more than one change (or bug fix), please consider submitting one patch per change. Do not collapse multiple changes into a single patch. For big changes consider using a patch queue.
- pass all unit-tests. See Running tests below.
- include one (or more) test cases that check the bug fixed or the new functionality added. See Writing tests below.
- if you're adding or changing a public (documented) API, please include the documentation changes in the same patch. See Documentation policies below.

Submitting patches

The best way to submit a patch is to issue a pull request [http://help.github.com/send-pull-requests/] on Github, optionally creating a new issue first.

Remember to explain what was fixed or the new functionality (what it is, why it's needed, etc). The more info you include, the easier will be for core developers to understand and accept your patch.

You can also discuss the new functionality (or bug fix) before creating the patch, but it's always good to have a patch ready to illustrate your arguments and show that you have put some additional thought into the subject. A good starting point is to send a pull request on Github. It can be simple enough to illustrate your idea, and leave documentation/tests for later, after the idea has been validated and proven useful. Alternatively, you can send an email to scrapy-users [http://groups.google.com/group/scrapy-users] to discuss your idea first.

Finally, try to keep aesthetic changes (**PEP 8** [http://www.python.org/dev/peps/pep-0008] compliance, unused imports removal, etc) in separate commits than functional changes. This will make pull requests easier to review and more likely to get merged.

Coding style

Please follow these coding conventions when writing code for inclusion in Scrapy:

- Unless otherwise specified, follow **PEP 8** [http://www.python.org/dev/peps/pep-0008].
- It's OK to use lines longer than 80 chars if it improves the code readability.
- Don't put your name in the code you contribute. Our policy is to keep the contributor's name in the AUTHORS

[https://github.com/scrapy/scrapy/blob/master/AUTHORS] file distributed with Scrapy.

Scrapy Contrib

Scrapy contrib shares a similar rationale as Django contrib, which is explained in this post [http://jacobian.org/writing/what-is-django-contrib/]. If you are working on a new functionality, please follow that rationale to decide whether it should be a Scrapy contrib. If unsure, you can ask in scrapy-users [http://groups.google.com/group/scrapy-users].

Documentation policies

- **Don't** use docstrings for documenting classes, or methods which are already documented in the official (sphinx) documentation. For example, the `ItemLoader.add_value()` method should be documented in the sphinx documentation, not its docstring.
- **Do** use docstrings for documenting functions not present in the official (sphinx) documentation, such as functions from `scrapy.utils` package and its sub-modules.

Tests

Tests are implemented using the Twisted unit-testing framework [http://twistedmatrix.com/documents/current/core/development/policy/test-standard.html], running tests requires tox [https://pypi.python.org/pypi/tox].

Running tests

To run all tests go to the root directory of Scrapy source code and run:

```
tox
```

To run a specific test (say `tests/test_contrib_loader.py`) use:

```
tox -- tests/test_contrib_loader.py
```

Writing tests

All functionality (including new features and bug fixes) must include a test case to check that it works as expected, so please include tests for your patches if you want them to get accepted sooner.

Scrapy uses unit-tests, which are located in the tests/ [https://github.com/scrapy/scrapy/tree/master/tests] directory. Their module name typically resembles the full path of the module they're testing. For example, the item loaders code is in:

```
scrapy.contrib.loader
```

And their unit-tests are in:

```
tests/test_contrib_loader.py
```

讨论

Versioning and API Stability

Versioning

Scrapy uses the odd-numbered versions for development releases [http://en.wikipedia.org/wiki/Software_versioning#Odd-numbered_versions_for_development_releases].

There are 3 numbers in a Scrapy version: *A.B.C*

- *A* is the major version. This will rarely change and will signify very large changes. So far, only zero is available for *A* as Scrapy hasn't yet reached 1.0.
- *B* is the release number. This will include many changes including features and things that possibly break backwards compatibility. Even Bs will be stable branches, and odd Bs will be development.
- *C* is the bugfix release number.

For example:

- *0.14.1* is the first bugfix release of the *0.14* series (safe to use in production)

API Stability

API stability is one of Scrapy major goals for the *1.0* release, which doesn't have a due date scheduled yet.

Methods or functions that start with a single dash (_) are private and should never be relied as stable. Besides those, the plan is to stabilize and document the entire API, as we approach the 1.0 release.

Also, keep in mind that stable doesn't mean complete: stable APIs could grow new methods or functionality but the existing methods should keep working the same way.

讨论

试验阶段特性

这部分介绍一些正处于试验阶段的Scrapy特性， 这些特性所涉及到的函数接口等还不够稳定， 但会在以后的发布版中趋于完善。所以在使用这些特性过程中需更谨慎， 并且最好订阅我们的 邮件列表 [http://scrapy.org/community/] 以便接收任何有关特性改变的通知。

虽然这些特性不会频繁的被修改，但是这部分文档仍有可能是过时的、 不完整的或是与已经稳定的特性文档重复。所以你需要自行承担使用风险。

警告

本部分文档一直处于修改中。请自行承担使用风险。

使用外部库插入命令

你可以使用外部库通过增加 *scrapy.commands* 部分到 *setup.py* 的entry_points中来插入Scrapy命令。

增加 *my_command* 命令的例子:

```
from setuptools import setup, find_packages

setup(name='scrapy-mymodule',
  entry_points={
    'scrapy.commands': [
      'my_command=my_scrapy_module.commands:MyCommand',
    ],
  },
 )
```

讨论

Python 模块索引

s

s

scrapy

索引

_

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X