Scrapy Documentation

リリース *1.7.3*

Scrapy developers

2019 年 11 月 11 日

入門

Scrapy は高速で高レベルの Web クロール (web crawling) および Web スクレイピング (web scraping) フレームワークであり、Web サイトをクロールし、ページから構造化データを抽出するために使用されます。 データ・マイニングから監視、自動テストまで、幅広い目的に使用できます。

第 1 章

お助け

問題がありますか？ でしたらこれらが助けになるでしょう。

- まず、 *FAQ* を見て下さい。一般的な質問に対する回答があります。
- 特定の情報をお探しですか？ それでしたら genindex や modindex をご覧ください。
- StackOverflow で scrapy タグを付けて質問するか検索します (StackOverflow using the scrapy tag)。
- Scrapy subreddit で質問するか検索してください。
- scrapy-users mailing list のアーカイブで質問を検索します。
- #scrapy IRC channel で質問します。
- issue tracker で Scrapy のバグを報告してください。

第 2 章

入門

2.1 Scrapy を 3 行で説明シル

Scrapy は、Web サイトをクロールし、構造化されたデータを抽出するためのアプリケーション・フレームワークです。データ・フレームワークは、データ・マイニング、情報処理、履歴アーカイブなど、さまざまな有用なアプリケーションに使用できます。

Scrapy は元々「ウェブ・スクレイピング (web scraping)」用に設計されていましたが、API(Amazon Associates Web Services など) を使用してデータを抽出したり、汎用の Web クローラーとして使用することもできます。

2.1.1 スパイダー例概観

Scrapy の恩恵を示すために、最も簡単な方法でスパイダーを実行する Scrapy Spider の例を紹介します。

以下は、Web サイト http://quotes.toscrape.com から有名な引用をスクレイピングするスパイダーのコードです:

```
import scrapy


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

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

(次のページに続く)

(前のページからの続き)

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

これをテキスト・ファイルに入れ、「quotes_spider.py」などの名前を付けて、 *runspider* コマンドを使用してスパイダーを実行します:

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

これが完了すると、quotes.json ファイルに、以下のようなテキストと著者を含む JSON 形式の引用のリストができます (読みやすくするために整え直してあります):

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

あ…ありのまま 今 起こった事を話すぜ！

あなたがコマンド `scrapy runspider quotes_spider.py` を実行すると、Scrapy はその内部でスパイダー定義を探し、そのクローラー・エンジンを介して実行しました。

クロールは `start_urls` 属性で定義された URL 群 (この場合、 *humor* カテゴリの quote の URL のみ) にリクエストを行うことで開始し、デフォルトのコールバック・メソッド `parse` を呼び出して、引数としてレスポンス・オブジェクトを渡します。 `parse` コールバックでは、CSS セレクターを使用して quote 要素をループし、抽出された quote テキストと著者を含む Python 辞書を生成 (yield) し、次のページへのリンクを探し、同一のコールバック・メソッド `parse` を使用して次のリクエストをスケジュールします。

ここで、Scrapy の主な利点の 1 つに気付きます。リクエストの **スケジューリングと処理は非同期** です。つまり、Scrapy はリクエストが終了して処理されるのを待つ必要がなく、その間に別のリクエストを送信したり、他のことを実行したりできます。 これは、一部のリクエストが失敗したり、処理中にエラーが発生した場合でも、他のリクエストが続行できることも意味します。

これにより、非常に高速なクロール (フォールトトレラント (訳注:その構成部品の一部が故障しても正常に処理を

続行するシステム) な方法で複数の同時要求を同時に送信) が可能になるけれども、更に Scrapy では **いくつかの設定** を通してクロールのポライトネス (訳注:円滑な人間関係を確立・維持するための言語行動) を制御することもできます。各リクエスト間にダウンロード遅延を設定したり、ドメインごとまたは IP ごとの同時リクエストの量を制限したり、これらを自動的に把握しようとする **自動スロットル拡張の使用** も可能です。

注釈: これは **フィード・エクスポート** を使用して JSON ファイルを生成します。エクスポート形式 (例えば XML や CSV など) またはストレージ・バックエンド (例えば (FTP や Amazon S3)。あなたは **アイテム・パイプライン** を記述して、アイテムをデータベースに保存することもできます。

2.1.2 他に何かある？

Scrapy を使用して Web サイトからアイテムを抽出および保存する方法を見てきましたが、これはほんのさわりです。 Scrapy はスクレイピングを簡単かつ効率的にするための次のような強力な機能を多数提供します:

- HTML/XML ソースからの、データ **選択と抽出** のための拡張 CSS セレクターと XPath 式の使用と、正規表現を使用して抽出するヘルパー・メソッドを組み込みでサポート。
- **対話シェル** (IPython 対応) は、CSS および XPath 式を試してデータをスクレイピングするためのもので、スパイダーを作成またはデバッグするときに非常に便利です。
- 複数の形式 (JSON、CSV、XML) でのデータ生成と、複数のバックエンド・タイプ (FTP、S3、ローカルファイルシステム) に保存するための **フィード・エクスポート生成** を組み込みでサポート。
- 不明な、非標準や壊れたエンコーディング宣言を処理するための、強力なエンコード支援と自動検出。
- **強力な拡張性サポート** により、 **シグナル** と明確に定義された API(ミドルウェアと **拡張機能** と **パイプライン**) を使用して、あなた独自の機能をプラグインできます。
- 広範な組み込み拡張機能と処理用のミドルウェア:
  - クッキーやセッションの取扱
  - 圧縮、認証、キャッシングなどの HTTP 機能
  - ユーザ・エージェントのなりすまし (spoofing)
  - robots.txt
  - クロールの深さの制限
  - などなど
- *Telnet* は、クローラー内部の調査およびデバッグのために、あなたの Scrapy プロセス内で実行されている Python コンソールにフックします。

- さらに、その他の便利な機能として、サイトマップ (Sitemaps) および XML/CSV フィードからサイトをクロールするための再利用可能なスパイダーや、 スクレイプされたアイテムに関連付けられている、 画像 (または他の媒体) を自動ダウンロードするための **媒体パイプライン** や、DNS リゾルバのキャッシングなど、その他多数あります！

2.1.3 さてお次は？

お次は、 *Scrapy* **インストール** を行い **チュートリアル** で本格的な Scrapy プロジェクトを作成する方法を学び、Scrapy コミュニティに参加 (join the community) します。あなたが Scrapy に興味を持ってくれてありがうございます！

2.2 インストール ガイド

2.2.1 Scrapy のインストール

Scrapy は CPython(デフォルトの Python 実装) の Python 2.7(またはそれ以上) と Python 3.5(またはそれ以上)、または PyPy(PyPy 5.9 以降) で動作します。

あなたが Anaconda または Miniconda を使用している場合、Linux、Windows、および OS X 用の最新のパッケージがある conda-forge チャンネルからパッケージをインストールできます。

`conda` を使用して Scrapy をインストールするには、以下を実行します:

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

代わりに、既に Python パッケージのインストールに精通している場合は、Scrapy とその依存関係を PyPI からインストールできます:

```
pip install Scrapy
```

あなたのご使用のオペレーティングシステムによっては、一部の Scrapy 依存関係のコンパイルの問題を解決する必要がある場合があるため、必ず **プラットフォーム別インストール・ノート** を確認してください。

システムパッケージとの競合を避けるため、Scrapy を **専用の** *virtualenv* **環境** にインストールすることを、私たちは強くお勧めします (訳注:python3 であれば venv 環境)。

より詳細なプラットフォーム固有の手順、およびトラブルシューティング情報については、以下をお読みください。

あなたが知っておくべきこと

Scrapy は純粋な Python で書かれており、いくつかの主要な Python パッケージに依存しています。(とりわけ以下に依存します):

- lxml 、効率的な XML および HTML パーサー
- parsel 、lxml で記述された HTML/XML データ抽出ライブラリ
- w3lib 、URL と Web ページのエンコーディングを扱うための多目的ヘルパー
- twisted 、非同期ネットワーキングフレームワーク
- さまざまなネットワークレベルのセキュリティニーズに対処するための cryptography (暗号化) と pyOpenSSL

Scrapy がテストできる最小バージョンは次のとおりです:

- Twisted 14.0
- lxml 3.4
- pyOpenSSL 0.14

Scrapy はこれらのパッケージの古いバージョンで動作する可能性がありますが、それらに対してテストされていないため、動作を継続する保証はありません。

これらの Python パッケージのいくつかは、プラットフォームによっては追加のインストールが必要な非 Python パッケージに依存しています。 **プラットフォーム固有のガイド** で確認して下さい。

これらの依存関係に関連する問題が発生した場合は、それぞれのインストール手順を参照してください:

- lxml installation
- cryptography installation

仮想環境の使用 (推奨)

一言でいうと、私たちは全てのプラットフォームで仮想環境内に Scrapy をインストールすることをおすすめします。(訳注: python3.3 以降の場合は venv があります。詳しくは Python ドキュメント/Python 標準ライブラリ/venv – 仮想環境の作成 https://docs.python.org/ja/3/library/venv.html 参照)

Python パッケージは、グローバル（システム全体）またはユーザー空間にインストールできます。 システム全体に Scrapy をインストールすることはお勧めしません。

代わりに、いわゆる仮想環境 (virtualenv) 内に Scrapy をインストールすることをお勧めします。 Virtualenv(訳注:python3 なら venv) を使用すると、既にインストールされている Python システムパッケージと競合 (システムツールやスクリプトの一部が破損する可能性があります) することなく、通常は (`sudo` などを使用せずに) `pip` でパッケージをインストールできます。

仮想環境の使用を開始するには、 virtualenv installation instructions を参照してください。 グローバルにインストールする (グローバルにインストールしないと役に立ちません) には、次を実行する必要があります:

```
$ [sudo] pip install virtualenv
```

virtualenv の作成方法については、この user guide を確認してください。

注釈: あなたが Linux または OS X を使用している場合、 virtualenvwrapper は virtualenv を作成するのに便利なツールです。

---

virtualenv を作成したら、他の Python パッケージと同様に `pip` を使用してその中に Scrapy をインストールできます (事前にインストールする必要がある非 Python 依存関係については、以下の **プラットフォーム別ガイド** を参照してください)。

Python virtualenv は、デフォルトで Python 2、またはデフォルトで Python 3 を使用するように作成できます。

- Python3 で Scrapy をインストールする場合は、Python3 の virtualenv 内に Scrapy をインストールします。
- また、Python2 で Scrapy をインストールする場合は、Python2 virtualenv 内に Scrapy をインストールします。

2.2.2 プラットフォーム別インストール・ノート

Windows

pip を使用して Windows に Scrapy をインストールすることは不可能ではありませんが、インストールにまつわる多くの問題を回避するために、 Anaconda または Miniconda をインストールし、 conda-forge チャンネルのパッケージを使用することをお勧めします。

あなたが Anaconda または Miniconda をインストールしたら、次のコマンドで Scrapy をインストールします:

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

Ubuntu 14.04 またはそれ以上

Scrapy は現在、lxml、twisted、pyOpenSSL の割と最近のバージョンでテストされており、最近の Ubuntu ディストリビューションと互換性があります。 しかし、TLS 接続に潜在的な問題がある Ubuntu 14.04 などの古いバージョンの Ubuntu もサポートされるべきです。

Ubuntu が提供する `python-scrapy` パッケージを **使用しないでください** 。これらはたいてい古すぎて、最新の Scrapy に追いつくことができていません。

Ubuntu(または Ubuntu ベース) システムに Scrapy をインストールするには、以下の依存関係をインストールする必要があります:

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

- `python-dev`、 `zlib1g-dev`、 `libxml2-dev`、 `libxslt1-dev` は、 `lxml` の為に必要です。
- `libssl-dev` と `libffi-dev` は `cryptography` の為に必要です。

Python3 に Scrapy をインストールする場合は、Python3 開発ヘッダーも必要になります:

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

その後、 *virtualenv* 内で `pip` で Scrapy をインストールできます:

```
pip install scrapy
```

注釈: Python 以外の同様の依存関係を使用して、Debian Jessie(8.0) 以降に Scrapy をインストールできます。

Mac OS X

Scrapy の依存関係を構築するには、C コンパイラと開発ヘッダーが必要です。 OS X では、通常これは Apple の Xcode 開発ツールによって提供されます。 Xcode コマンドラインツールをインストールするには、ターミナルウィンドウを開き、以下を実行します:

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

`pip` によるシステムパッケージの更新を妨げる既知の問題 (known issue) があります。 Scrapy とその依存関係を正常にインストールするには、これに対処する必要があります。 以下にいくつかの解決策を示します:

- **(推奨) システムの python を使用しないでください。** システムの他の部分と競合しない新しい更新バージョンをインストールしてください。 homebrew パッケージマネージャーを使用して行う方法は次のとおりです:
  - https://brew.sh/ の指示に従って homebrew をインストールします。
  - `PATH` 環境変数を更新して、システムパッケージより先に homebrew パッケージを使用するように指定します (デフォルトのシェルとして zsh を使用している場合は、 `.bashrc` を `.zshrc` に変更します):

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

  - `.bashrc` をリロードして、変更が行われたことを確認します:

    ```
    source ~/.bashrc
    ```

  - python のインストール:

```
brew install python
```

  - python の最新バージョンには `pip` がバンドルされているため、個別にインストールする必要はありません。 そうでない場合は、python をアップグレードします:

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

- *(**オプション**)* 隔離された python 環境内に Scrapy をインストールします。

  この方法は、上記の OS X の問題の回避策ですが、依存関係を管理するための全体的なグッドプラクティスであり、最初の方法を補完できます。

  virtualenv は、Python で仮想環境を作成するために使用できるツールです。 http://docs.python-guide.org/en/latest/dev/virtualenvs/ のようなチュートリアルを読むことをお勧めします。

これらの回避策のいずれかを行った後、Scrapy をインストールできるはずです:

```
pip install Scrapy
```

PyPy

私たちは最新の PyPy バージョンを使用することをお勧めします。 テストされたバージョンは 5.9.0 です。 PyPy3 では、Linux インストールのみがテストされました。

現在、Scrapy が依存するほとんどの依存コンポーネントには、CPython 用のバイナリホイール形式のパッケージがありますが、PyPy 用ではありません。 これは、これらの依存関係がインストール中に構築されることを意味します。 OS X では、暗号化の依存関係の構築に関する問題に直面する可能性があります。この問題の解決策は こちら で説明されています。 `brew install openssl` してから、このコマンドが推奨するフラグをエクスポートします (scrapy のインストール時にのみ必要)。 Linux へのインストールには、ビルドの依存関係のインストール以外に特別な問題はありません。 Windows で PyPy を使用した Scrapy のインストールはテストされていません。

あなたは、 `scrapy bench` を実行して、scrapy が正しくインストールされていることを確認できます。 このコマンドが `TypeError: ... got 2 " "unexpected keyword arguments` のようなエラーを出す場合、これは setuptools が 1 つの PyPy 固有の依存関係を選択できなかったことを意味します。 この問題を修正するには、 `pip install 'PyPyDispatcher>=2.1.0'` を実行します。

2.2.3 トラブルシューティング

AttributeError: 'module' object has no attribute 'OP_NO_TLSv1_1'

Scrapy、Twisted、または pyOpenSSL をインストールまたはアップグレードした後、トレースバックで以下の例外が発生する場合があります:

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

この例外が発生する理由は、Twisted のバージョンがサポートしていない pyOpenSSL のバージョンがシステムまたは仮想環境にあるためです。

Twisted のバージョンがサポートする pyOpenSSL のバージョンをインストールするには、`tls` 追加オプションで Twisted を再インストールします:

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

詳細は Issue #2473 をご覧ください。

2.3 Scrapy チュートリアル

このチュートリアルでは、Scrapy がシステムに既にインストールされていると仮定します。 そうでない場合は、**インストール ガイド** を参照してください。

ここでは quotes.toscrape.com という、有名な著者からの引用をリストするウェブサイトをスクレイピングします。

このチュートリアルでは以下の作業について説明します。

1. 新しい Scrapy プロジェクトの作成
2. **スパイダー** *(spider)* を作成してサイトをクロールし、データを抽出します。
3. コマンドラインを使用してスクレイピングされたデータをエクスポートする。
4. 再帰的にリンクをたどるようにスパイダーを変更する。
5. スパイダー引数の使用

Scrapy は Python で書かれています。 この言語を初めて使用する場合は、Scrapy を最大限に活用するために、この言語がどのようなものかを理解することから始めてください。

すでに他の言語に精通しており、Python をすばやく学習したい場合は、 Python Tutorial (訳注:日本語版 https://docs.python.org/ja/3/tutorial/) が優れた文書です。

プログラミングが初めてで、Python を使い始めたい場合は、以下の書籍が役立ちます:

- Automate the Boring Stuff With Python (邦訳： 退屈なことは Python にやらせよう――ノンプログラマーにもできる自動化処理プログラミング Al Sweigart　著、相川 愛三　訳 2017 年 06 月 発行 ISBN978-4-87311-778-2 オライリー・ジャパン)

- How To Think Like a Computer Scientist (訳注：ThinkPython の日本語版ページ http://www.cauldron.sakura.ne.jp/thinkpython/)
- Learn Python 3 The Hard Way (邦訳：Learn Python 3 the Hard Way 書いて覚える Python 入門 堂阪　真司 訳 丸善出版 2019 年 01 月 ISBN 978-4-621-30328-3)

また、 this list of Python resources for non-programmers (この非プログラマ向けの Python リソースのリスト) と、suggested resources in the learnpython-subreddit (learnpython-subreddit の推奨リソース) を参照することもできます。

2.3.1 プロジェクトの作成

スクレイピングを開始する前に、新しい Scrapy プロジェクトをセットアップする必要があります。 あなたのコードを保存して実行するディレクトリを入力してください:

```
scrapy startproject tutorial
```

これにより、以下の内容の `tutorial` ディレクトリが作成されます:

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

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

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

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

2.3.2 私たちの最初のスパイダー

スパイダーはユーザが定義するクラスであり、Scrapy は Web サイト (または Web サイトのグループ) から情報をスクレイピングするために使用します。 `scrapy.Spider` をサブクラス化し、最初のリクエストを作成し、オプションでページ内のリンクをたどる方法、およびダウンロードしたページ内容をパースしてデータを抽出する方法を定義する必要があります。

以下は、最初のスパイダーのコードです。 プロジェクトの `tutorial/spiders` ディレクトリの下の

quotes_spider.py という名前のファイルに保存します:

```
import scrapy


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

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

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

ご覧のとおり、スパイダーは *scrapy.Spider* をサブクラス化し、いくつかの属性とメソッドを定義しています:

- *name* は、スパイダーを識別します。 プロジェクト内で一意である必要があります。つまり、異なるスパイダーに同じ名前を設定することはできません。
- *start_requests()* は、スパイダーがクロールを開始するリクエストの反復可能オブジェクト (iterable) を返す必要があります (リクエストのリストを返すか、ジェネレーター関数を作成できます)。これらの初期リクエストから後続のリクエストが連続して生成されます (訳注:iterable の意味は Python ドキュメント/用語集 https://docs.python.org/ja/3/glossary.html 参照)。
- *parse()* は、行われたリクエストごとにダウンロードされたレスポンスを処理するために呼び出されるメソッドです。 リクエスト・パラメーターは *TextResponse* のインスタンスで、ページ内容を保持し、さらにそれを処理するための便利なメソッドを持っています。

  parse() メソッドは通常、レスポンスをパースし、スクレイプされたデータを辞書として抽出し、追跡する新しい URL を見つけて、それらから新しいリクエスト (*Request*) を作成します。

私たちのスパイダーの実行方法

スパイダーを動作させるには、プロジェクトの最上位ディレクトリに移動して、以下を実行します:

```
scrapy crawl quotes
```

このコマンドは、追加したばかりの quotes という名前のスパイダーを実行し、 quotes.toscrape.com ドメ

インへのリクエストを送信します。以下のような出力が得られます:

```
... (omitted for brevity)
2016-12-16 21:24:05 [scrapy.core.engine] INFO: Spider opened
2016-12-16 21:24:05 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/
↪min), scraped 0 items (at 0 items/min)
2016-12-16 21:24:05 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.
↪0.0.1:6023
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://quotes.
↪toscrape.com/robots.txt> (referer: None)
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.
↪toscrape.com/page/1/> (referer: None)
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.
↪toscrape.com/page/2/> (referer: None)
2016-12-16 21:24:05 [quotes] DEBUG: Saved file quotes-1.html
2016-12-16 21:24:05 [quotes] DEBUG: Saved file quotes-2.html
2016-12-16 21:24:05 [scrapy.core.engine] INFO: Closing spider (finished)
...
```

次に、現在のディレクトリのファイルを確認します。 `parse` メソッドが指示するように、それぞれの URL のコンテンツを持つ 2 つの新しいファイル *quotes-1.html* と *quotes-2.html* が作成されていることに気付くはずです。

注釈: なぜまだ HTML をパースしていないのかって？順番に説明するからもうちょい待ってくれ。

一体全体どういう仕組みなのか？

Scrapy は、スパイダーの `start_requests` メソッドによって返される *scrapy.Request* オブジェクトをスケジュールします。 それぞれのレスポンスを受信すると、 *Response* オブジェクトをインスタンス化し、リクエストに関連付けられたコールバック・メソッド (この場合は `parse` メソッド) を呼び出して、レスポンスを引数として渡します。

start_requests メソッドへのショートカット

URL から *start_requests()* オブジェクトを生成する *start_requests()* メソッドを実装する代わりに、単に URL のリストで *start_urls* を定義できます。 このリストはそれから *start_requests()* のデフォルト実装で使用され、あなたのスパイダーの初期リクエストを作成します:

```
import scrapy


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

(次のページに続く)

(前のページからの続き)

```
        'http://quotes.toscrape.com/page/1/',
        'http://quotes.toscrape.com/page/2/',
    ]

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

parse() メソッドは、Scrapy に明示的に指示していない場合でも、これらの URL の各リクエストを処理するために呼び出されます。 これは、 *parse()* が Scrapy のデフォルトのコールバック・メソッドであり、明示的にコールバックが割り当てられていないリクエストに対して呼び出されるためです。

データの抽出

Scrapy でデータを抽出する方法を学ぶ最良の方法は、 *Scrapy* **シェル** を使用してセレクターを試すことです。 以下のように実行します:

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

注釈: コマンドラインから Scrapy シェルを実行するときは、常に URL をクォーテーションで囲むことを忘れないでください。そうしないと、引数 (つまり、 & キャラクタ) を含む URL は機能しません。

Windows では代わりにダブルクォーテーションを使って下さい:

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

あなたは以下のようなものを見る事になるでしょう:

```
[ ... Scrapy log here ... ]
2016-09-19 12:09:27 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.
↪toscrape.com/page/1/> (referer: None)
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x7fa91d888c90>
[s]   item       {}
[s]   request    <GET http://quotes.toscrape.com/page/1/>
[s]   response   <200 http://quotes.toscrape.com/page/1/>
[s]   settings   <scrapy.settings.Settings object at 0x7fa91d888c10>
[s]   spider     <DefaultSpider 'default' at 0x7fa91c8af990>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
```

(次のページに続く)

(前のページからの続き)

```
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser
>>>
```

シェルを使用して、あなたはレスポンス・オブジェクトで CSS を使用して要素の選択を試す事ができます:

```
>>> response.css('title')
[<Selector xpath='descendant-or-self::title' data='<title>Quotes to Scrape</title>'>]
```

`response.css('title')` を実行した結果は、 `SelectorList` というリストのようなオブジェクトになり、これは XML/HTML 要素をラップし、さらにクエリを実行して選択範囲を細かくしたり、データを抽出したりできるオブジェクトである `Selector` のリストになっています。

上記のタイトルからテキストを抽出するには、以下のようにします:

```
>>> response.css('title::text').getall()
['Quotes to Scrape']
```

ここで注意すべき点が 2 つあります。1 つは、CSS クエリに `::text` を追加したことです。これは、 `<title>` 要素内のテキスト要素のみを直接選択することを意味します。 `::text` を指定しない場合、そのタグを含む完全なタイトル要素を取得します:

```
>>> response.css('title').getall()
['<title>Quotes to Scrape</title>']
```

もう 1 つは、 `.getall()` を呼び出した結果がリストであるということです。セレクターが複数の結果を返す可能性があり、そしてそれらの全てを抽出します。この場合のように、最初の結果だけが必要であることがわかったら、次の操作を実行できます:

```
>>> response.css('title::text').get()
'Quotes to Scrape'
```

代わりに以下のように書くこともできます:

```
>>> response.css('title::text')[0].get()
'Quotes to Scrape'
```

けれども、 `SelectorList` インスタンスで `.get()` を直接使用すると、 `IndexError` を回避し、セレクターに一致する要素が見つからない場合 `None` を返します。

ここに教訓があります。ほとんどのスクレイピングコードでは、ページ上で見つからないものに起因するエラーに対して回復力を持たせ、一部のスクレイピングに失敗した場合でも、少なくとも **いくつかの** データを取得できるようにします。

getall() メソッドや get() メソッドに加えて、正規表現 (regular expressions) により抽出する re() メソッドも使用できます:

```
>>> response.css('title::text').re(r'Quotes.*')
['Quotes to Scrape']
>>> response.css('title::text').re(r'Q\w+')
['Quotes']
>>> response.css('title::text').re(r'(\w+) to (\w+)')
['Quotes', 'Scrape']
```

使用する適切な CSS セレクターを見つけるには、 view(response) を使用して Web ブラウザーのシェルからレスポンス・ページを開くと便利です。 ブラウザの開発ツールを使用して HTML を調査し、セレクターを作成できます (*Web* **ブラウザの開発ツールを使ってスクレイピングする** 参照)。

Selector Gadget という、多くのブラウザで動作する、選択された要素の CSS セレクターを視覚的にすばやく探せる素晴らしいツールもあります。

XPath の簡単な紹介

CSS に加えて、Scrapy セレクターは XPath 式の使用もサポートしています:

```
>>> response.xpath('//title')
[<Selector xpath='//title' data='<title>Quotes to Scrape</title>'>]
>>> response.xpath('//title/text()').get()
'Quotes to Scrape'
```

XPath 式は非常に強力であり、Scrapy セレクターの基盤です。 実際、CSS セレクターは内部で XPath に変換されます。 シェル内のセレクター・オブジェクトのテキスト表現をよく読んでいれば、あなたはそれに気付く事ができるでしょう。

CSS セレクターほど一般的ではないかもしれませんが、XPath 式は構造をナビゲートするだけでなく、内容を探すことができるため、より強力になります。 XPath を使用すると、「『Next Page』というテキストを含むリンクを選択」というような事ができます。これにより、XPath はスクレイピングのタスクに非常に適合します。CSS セレクターの構築方法を既に知っている場合でも、XPath を学ぶことをお勧めします。

ここでは XPath についてはあまり取り上げませんが、 *Scrapy* **セレクターで** *XPath* **を使用** に詳しく載っています。XPath の詳細については、 this tutorial to learn XPath through examples と this tutorial to learn "how to think in XPath" を私たちはお勧めします。

引用と著者の抽出

選択と抽出について少し理解できたので、Web ページから引用を抽出するコードを作成して、スパイダーを完成させましょう。

http://quotes.toscrape.com の各引用は、次のような HTML 要素で表されます:

```
<div class="quote">
    <span class="text">“The world as we have created it is a process of our
    thinking. It cannot be changed without changing our thinking.”</span>
    <span>
        by <small class="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
    </span>
    <div class="tags">
        Tags:
        <a class="tag" href="/tag/change/page/1/">change</a>
        <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
        <a class="tag" href="/tag/thinking/page/1/">thinking</a>
        <a class="tag" href="/tag/world/page/1/">world</a>
    </div>
</div>
```

Scrapy シェルで少しいじって、必要なデータを抽出する方法を見つけましょう:

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

引用 HTML 要素のセレクターのリストを取得します:

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

上記のクエリによって返された各セレクタを使用すると、サブ要素に対してさらにクエリを実行できます。 最初のセレクターを変数に割り当てて、特定の引用で CSS セレクターを直接実行できるようにします:

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

それでは、作成したばかりの `quote` オブジェクトを使用して、その引用から `text` と `author` と `tags` を抽出しましょう:

```
>>> text = quote.css("span.text::text").get()
>>> text
'“The world as we have created it is a process of our thinking. It cannot be changed␣
↪without changing our thinking.”'
>>> author = quote.css("small.author::text").get()
>>> author
'Albert Einstein'
```

タグは文字列のリストになっているので、 `.getall()` メソッドを使用してそれらすべてを取得できます:

```
>>> tags = quote.css("div.tags a.tag::text").getall()
>>> tags
['change', 'deep-thoughts', 'thinking', 'world']
```

各パーツを抽出する方法を考え出したので、すべての引用要素を反復処理して、それらを Python 辞書にまとめる

ことができます:

```
>>> for quote in response.css("div.quote"):
...     text = quote.css("span.text::text").get()
...     author = quote.css("small.author::text").get()
...     tags = quote.css("div.tags a.tag::text").getall()
...     print(dict(text=text, author=author, tags=tags))
{'tags': ['change', 'deep-thoughts', 'thinking', 'world'], 'author': 'Albert Einstein',
 'text': '“The world as we have created it is a process of our thinking. It cannot be␣
↪changed without changing our thinking.”'}
{'tags': ['abilities', 'choices'], 'author': 'J.K. Rowling', 'text': '“It is our␣
↪choices, Harry, that show what we truly are, far more than our abilities.”'}
    ... a few more of these, omitted for brevity
>>>
```

私たちのスパイダーでデータを抽出する

私たちのスパイダーに戻ります。これまでは、特にデータを抽出せず、HTML ページ全体をローカルファイルに保存するだけでした。それでは、ここで、上記の抽出ロジックを私たちのスパイダーに組み込みましょう。

Scrapy スパイダーは通常、ページから抽出されたデータを含む多くの Python 辞書を生成します。 これを行うには、以下に示すように、コールバックで Python キーワード `yield` を使用します:

```
import scrapy


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

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }
```

あなたがこのスパイダーを実行すると、抽出されたデータが log とともに出力されます:

```
2016-09-19 18:57:19 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.
↪toscrape.com/page/1/>
{'tags': ['life', 'love'], 'author': 'Andr^^c3^^a9 Gide', 'text': '“It is better to␣
↪be hated for what you are than to be loved for what you are not.”'}
```

(次のページに続く)

(前のページからの続き)

```
2016-09-19 18:57:19 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.
↪toscrape.com/page/1/>
{'tags': ['edison', 'failure', 'inspirational', 'paraphrased'], 'author': 'Thomas A.
 Edison', 'text': "“I have not failed. I've just found 10,000 ways that won't work.”"}
```

2.3.3 スクレイピングしたデータの格納

スクレイピングされたデータを保存する最も簡単な方法は、 **フィード・エクスポート** を以下のコマンドで使用することです:

```
scrapy crawl quotes -o quotes.json
```

それにより、スクレイピングされたすべてのアイテムを含み JSON でシリアライズされた `quotes.json` ファイルを生成します。

歴史的な理由により、Scrapy はファイルの内容を上書きする代わりに、指定されたファイルに追加します。 2 回目の前にファイルを削除せずにこのコマンドを 2 回実行すると、JSON ファイルが壊れてしまいます。

JSON Lines のような他の形式を使用することもできます:

```
scrapy crawl quotes -o quotes.jl
```

JSON Lines フォーマットは、ストリームに似ているため便利です。新しいレコードを簡単に追加できます。 2 回実行する場合、JSON と違って壊れる事はありません。また、各レコードは個別の行であるため、メモリにすべてを収めなくても大きなファイルを処理できます。コマンドラインでそれを行うのに役立つ JQ などのツールがあります。

小さなプロジェクト (このチュートリアルのようなプロジェクト) では、これで十分です。 ただし、スクレイピングされたアイテムを使用してより複雑な操作を実行する場合は、 **アイテム・パイプライン** を記述できます。 アイテム・パイプラインのプレースホルダーファイルは、プロジェクトの作成時に `tutorial/pipelines.py` に設定されています。 ただし、スクレイピングされたアイテムを保存するだけの場合は、アイテム・パイプラインを実装する必要はありません。

2.3.4 リンクを辿る

たとえば、http：//quotes.toscrape.com の最初の 2 ページの内容を単にスクレイピングするのではなく、Web サイトのすべてのページの引用が必要な場合を考えます。

ページからデータを抽出する方法がわかったので、ページからリンクをたどる方法を見てみましょう。

まず、辿りたいページへのリンクを抽出します。 ページを調べると、以下のマークアップを含む、次のページへのリンクがあることがわかります:

```
<ul class="pager">
    <li class="next">
        <a href="/page/2/">Next <span aria-hidden="true">&rarr;</span></a>
    </li>
</ul>
```

私たちは Scrapy シェルでこれの抽出を試してみることができます:

```
>>> response.css('li.next a').get()
'<a href="/page/2/">Next <span aria-hidden="true">→</span></a>'
```

これはアンカー要素を取得しますが、属性 `href` が必要です。 そのために、Scrapy は、次のように属性の内容を選択できる CSS 拡張機能をサポートしています:

```
>>> response.css('li.next a::attr(href)').get()
'/page/2/'
```

`attrib` プロパティもあります (詳細については **要素属性の選択** 参照):

```
>>> response.css('li.next a').attrib['href']
'/page/2'
```

次のページへのリンクを再帰的にたどって、そこからデータを抽出するように修正した私たちのスパイダーを見てみましょう:

```
import scrapy


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

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }

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

ここで、データを抽出した後、 parse() メソッドは次のページへのリンクを探し、 *urljoin()* メソッドを使用して完全な絶対 URL を構築します (リンクは相対的な場合があります)、そして次のページへの新しいリクエストを生成し、次のページのデータ抽出を処理し、すべてのページをクロールし続けるために、コールバックとして登録します。

ここに表示されるのは、リンクをたどる Scrapy のメカニズムです。コールバック・メソッドでリクエストを生成すると、Scrapy はそのリクエストの送信をスケジュールし、リクエストが終了したときに実行されるコールバック・メソッドを登録します。

これを使用して、定義したルールに従ってリンクをたどる複雑なクローラーを構築し、アクセスしているページに応じてさまざまな種類のデータを抽出できます。

この例では、次のページへのすべてのリンクをたどるループを作成します。ループが見つからなくなるまで、ブログやフォーラムやその他、ページ分けのあるサイトをクロールするのに便利です。

リクエストを作成するためのショートカット

Request オブジェクトを作成するためのショートカットとして、 response.follow を使用できます:

```
import scrapy


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

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('span small::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }

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

scrapy.Request とは異なり、 response.follow は相対 URL に対応しています。つまり、urljoin を呼び出す必要はありません。 response.follow は Request インスタンスを返すだけであることに注意してください。scrapy.Request と同様、あなたはこのリクエストを作用 (yield) させる必要があります。

文字列ではなく response.follow にセレクターを渡すこともできます。 このセレクターは必要な属性を抽出する必要があります:

```
for href in response.css('li.next a::attr(href)'):
    yield response.follow(href, callback=self.parse)
```

<a> 要素のためのショートカットがあります。 response.follow は href 属性を自動的に使用します。 したがって、コードをさらに短くすることができます:

```
for a in response.css('li.next a'):
    yield response.follow(a, callback=self.parse)
```

注釈: response.css は、単一のセレクターではなく、すべての結果のセレクターを持つリストのようなオブジェクトを返すため、 response.follow(response.css('li.next a')) は有効ではありません。上記の例のような for ループ、または response.follow(response.css('li.next a')[0]) は問題ありません。

さらなる例やパターン

コールバックと次のリンクを示す別のスパイダーを次に示します。今回は著者情報をスクレイピングするためのものです:

```
import scrapy


class AuthorSpider(scrapy.Spider):
    name = 'author'

    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        # follow links to author pages
        for href in response.css('.author + a::attr(href)'):
            yield response.follow(href, self.parse_author)

        # follow pagination links
        for href in response.css('li.next a::attr(href)'):
            yield response.follow(href, self.parse)

    def parse_author(self, response):
        def extract_with_css(query):
            return response.css(query).get(default='').strip()

        yield {
            'name': extract_with_css('h3.author-title::text'),
            'birthdate': extract_with_css('.author-born-date::text'),
```

(次のページに続く)

(前のページからの続き)

```
            'bio': extract_with_css('.author-description::text'),
        }
```

このスパイダーはメインページから始まり、各ページの parse_author コールバックを呼び出す作成者ページへのすべてのリンクと、以前解説したように parse コールバックのページ分けリンクをたどります。

ここでは、コードを短くするための位置引数としてコールバックを response.follow に渡しますが、scrapy.Request でも機能します。

parse_author コールバックは、CSS クエリからデータを抽出してクリーンアップするヘルパー関数を定義し、著者データを含む Python 辞書を生成します。

このスパイダーが示すもう 1 つの興味深い点は、同じ著者からの引用が多数ある場合でも、同じ著者ページに何度もアクセスすることを心配する必要がないことです。 デフォルトでは、Scrapy はすでにアクセスした URL への重複したリクエストを除外し、プログラミングのミスによるサーバーへの過剰なアクセスの問題を回避します。 これは設定 *DUPEFILTER_CLASS* で設定できます。

今や、あなたは Scrapy でリンクとコールバックをたどるメカニズムを使う方法を十分に理解できたと思います。

リンクをたどるメカニズムを活用するさらに別のスパイダーの例として、クローラーを作成するために使用できる小さなルールエンジンを実装する汎用スパイダーの *CrawlSpider* クラスを確認してください。

また、一般的なパターンは、 **コールバックに追加のデータを渡す手口** を使用して、複数のページからのデータでアイテムを構築することです。

2.3.5 スパイダー引数の使用

あなたはスパイダーの実行時に -a オプションを使用して、スパイダーにコマンドライン引数を提供できます:

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

これらの引数は Spider の __init__ メソッドに渡され、デフォルトで spider 属性になります。

この例では、 tag 引数に指定された値は self.tag を介して利用できます。 これを使用して、スパイダーに特定のタグを持つ引用のみを読み込み (fetch) させ、引数に基づいて URL を構築できます:

```
import scrapy


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

    def start_requests(self):
        url = 'http://quotes.toscrape.com/'
```

(次のページに続く)

(前のページからの続き)

```
        tag = getattr(self, 'tag', None)
        if tag is not None:
            url = url + 'tag/' + tag
        yield scrapy.Request(url, self.parse)

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

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

あなたがこのスパイダーに tag=humor 引数を渡すと、http://quotes.toscrape.com/tag/humor などの humor タグの URL のみにアクセスすることに気付くでしょう。

スパイダー引数の取扱について更に学ぶ をご覧ください。

2.3.6 さぁてお次は？

このチュートリアルでは、Scrapy の基本のみを説明しましたが、ここには記載されていない他の多くの機能があります。 最も重要なものの簡単な概要については、 *Scrapy* **を３行で説明シル** の章の **他に何かある？** 節を確認してください。

基本の概念 節では続けて、コマンドラインツール、スパイダー、セレクター、およびスクレイプデータのモデリングのようにチュートリアルで扱っていないその他のことについて詳しく知ることができます。 サンプルプロジェクトで遊びたい場合は、 **例** 節を確認してください。

2.4 例

学ぶための最良の方法は例であり、Scrapy も例外ではありません。 このため、 quotesbot という名前のサンプル Scrapy プロジェクトがあります。あなたはこのプロジェクトを使用して、Scrapy をプレイし、学習することができます。 これには、 http://quotes.toscrape.com 2 つのスパイダーが含まれています。1 つは CSS セレクターを使用し、もう 1 つは XPath 式を使用します。

quotesbot プロジェクトは、https://github.com/scrapy/quotesbot で入手できます。 プロジェクトの README で詳細を確認できます。

あなたが git に精通している場合は、コードをチェックアウトできます。 それ以外の場合は、 https://github.com/scrapy/quotesbot/archive/master.zip をクリックして、プロジェクトを zip ファイルとしてダウンロードできます。

Scrapy を３行で説明シル　Scrapy とは何か、それがどのようにあなたに役立つかの理解。

インストール ガイド　Scrapy をあなたのコンピュータにインストール。

Scrapy チュートリアル　あなたは最初の Scrapy プロジェクトを書きます。

例　作成済みの Scrapy プロジェクトを題材に、もっともっと学びます。

第 3 章

基本の概念

3.1 コマンドラインツール

バージョン 0.10 で追加.

Scrapy は、`scrapy` コマンドラインツール（ここでは「Scrapy ツール」と呼びます）を通じて制御され、単に「コマンド」または「Scrapy コマンド」と呼ばれるサブコマンドと区別します。

Scrapy ツールは複数の目的のためにいくつかのコマンドを提供し、それぞれが異なる引数とオプションの組を受け入れます。

(「scrapy deploy」コマンドは 1.0 で削除され、スタンドアロンの「scrapyd-deploy」に置き換えられました。Deploying your project を参照してください。)

3.1.1 構成 (configuration) の設定

Scrapy は、標準の場所にあるはずの、ini ファイルスタイルの `scrapy.cfg` ファイルの構成パラメーターを探します。:

1. `/etc/scrapy.cfg` または `c:\scrapy\scrapy.cfg` (お使いのコンピューターシステム全体の設定)、
2. `~/.config/scrapy.cfg` (`$XDG_CONFIG_HOME`) and `~/.scrapy.cfg` (`$HOME`) は、当該ユーザー全体の設定で、
3. (あなたの)Scrapy プロジェクトのルート内にある `scrapy.cfg` (詳しくは次節参照)。

これらのファイルの設定は、リストされている優先順位でマージされます。ユーザー定義の値は、システム全体のデフォルトよりも優先度が高く、プロジェクト全体の設定は、定義時に他のすべてを上書きします。

Scrapy は、多くの環境変数もまた理解しており、それらを使用して構成できます。 現在、これらは以下のとおりです。:

- `SCRAPY_SETTINGS_MODULE` (**設定の指定** 参照)

- SCRAPY_PROJECT (**プロジェクト間でルートディレクトリを共有する** 参照)
- SCRAPY_PYTHON_SHELL (*Scrapy* **シェル** 参照)

3.1.2 Scrapy プロジェクトのデフォルト構造

コマンドラインツールとそのサブコマンドを掘り下げる前に、まず Scrapy プロジェクトのディレクトリ構造を理解しましょう。

変更も可能ではありますが、すべての Scrapy プロジェクトはデフォルトでは同じファイル構造を持ち、以下のようになります。:

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

scrapy.cfg ファイルが存在するディレクトリは **プロジェクトルートディレクトリ** と呼ばれます。 そのファイルには、プロジェクト設定を定義する python モジュールの名前が含まれています。 以下に例を示します。:

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

3.1.3 プロジェクト間でルートディレクトリを共有する

scrapy.cfg を含むプロジェクトルートディレクトリは、それぞれ独自の設定モジュールを持つ複数の Scrapy プロジェクトで共有できます。

その場合、あなたの scrapy.cfg ファイルの [settings] の下で、これらの設定モジュールの 1 つ以上のエイリアスを定義する必要があります。:

```
[settings]
default = myproject1.settings
project1 = myproject1.settings
project2 = myproject2.settings
```

デフォルトでは、 scrapy コマンドラインツールは default 設定を使用します。 scrapy コマンドラインツールを別プロジェクトのために使うには SCRAPY_PROJECT 環境変数を設定します。:

```
$ scrapy settings --get BOT_NAME
Project 1 Bot
$ export SCRAPY_PROJECT=project2
$ scrapy settings --get BOT_NAME
Project 2 Bot
```

3.1.4 Scrapy ツールの使用

あなたは引数なしで Scrapy ツールを実行することから始める事ができます。そうすると、いくつかの使用方法のヘルプと使用可能なコマンドが出力されます。:

```
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
[...]
```

Scrapy プロジェクト内にいる場合、最初の行は現在アクティブなプロジェクトを印刷します。 上の例では、プロジェクトの外部から実行されました。 プロジェクト内から実行すると、以下のようなものが出力されます。:

```
Scrapy X.Y - project: myproject

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

[...]
```

プロジェクトの作成

Scrapy ツールで通常最初に行うことは、Scrapy プロジェクトの作成です。:

```
scrapy startproject myproject [project_dir]
```

これは project_dir ディレクトリの下に Scrapy プロジェクトを作成します。 project_dir を指定しなかった場合、 project_dir は myproject と同じになります。

次に、あなたは新しいプロジェクトディレクトリ内に移動します。:

```
cd project_dir
```

いまや、 あなたは、 scrapy コマンドを使用して、プロジェクトを管理および制御する準備が整いました。

プロジェクトの制御

あなたはプロジェクト内で Scrapy ツールを使用して、プロジェクトを制御および管理します。

例えば、新しいスパイダーを作成するには、:

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

一部の Scrapy コマンド (*crawl* のような) は、Scrapy プロジェクト内から実行する必要があります。 プロジェクト内から実行する必要があるコマンドと実行しないコマンドの詳細については、 **コマンド リファレンス** をご覧ください。

また、一部のコマンドは、プロジェクト内から実行する場合、動作が若干異なる場合があることに注意してください。 たとえば、フェッチされる URL が特定のスパイダーに関連付けられている場合、fetch コマンドはスパイダーによってオーバーライドされる動作 (user-agent をオーバーライドする user_agent 属性など) を使用します。 fetch コマンドはスパイダーがどのようにページをダウンロードしているかを確認するために使用されることを意図しているため、これは意図的なものです。

3.1.5 利用可能なツールコマンド

この節には、使用可能な組み込みコマンドの一覧とその説明および使用例が含まれています。 以下のコマンドを実行すると、いつでも各コマンドに関する詳細情報を見ることができます。:

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

そして、以下のコマンドですべての利用可能なコマンドの一覧を見ることができます。:

```
scrapy -h
```

コマンドには 2 種類あります。Scrapy プロジェクト内からのみ動作するコマンド（プロジェクト固有のコマンド）とアクティブな Scrapy プロジェクトなしで動作するコマンド（グローバルコマンド）です。ただし、プロジェクト内から実行すると動作が若干異なる場合があります (プロジェクトのオーバーライドされた設定を使用するため)。

グローバルコマンド

- *startproject*
- *genspider*
- *settings*

- *runspider*
- *shell*
- *fetch*
- *view*
- *version*

プロジェクト内のみのコマンド

- *crawl*
- *check*
- *list*
- *edit*
- *parse*
- *bench*

startproject

- 文法: `scrapy startproject <project_name> [project_dir]`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

`project_dir` ディレクトリの下に `project_name` という名前の新しい Scrapy プロジェクトを作成します。`project_dir` を指定しなかった場合、 `project_dir` は `project_name` と同じになります。

使用例:

```
$ scrapy startproject myproject
```

genspider

- 文法: `scrapy genspider [-t template] <name> <domain>`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

プロジェクト内から呼び出された場合、現在のフォルダーまたは現在のプロジェクトの `spiders` フォルダーに新しいスパイダーを作成します。 `<name>` パラメーターはスパイダーの `name` として設定され、 `<domain>``はスパイダー属性の ``allowed_domains` および `start_urls` を生成するために使用されます。

使用例:

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

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

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

これは、事前定義されたテンプレートに基づいてスパイダーを作成する便利なショートカットコマンドですが、スパイダーを作成する唯一の方法ではありません。 このコマンドを使用する代わりに、自分でスパイダーソースコードファイルを作成することもできます。

crawl

- 文法: `scrapy crawl <spider>`
- Scrapy プロジェクト内で実行させる必要があるか: **はい**

スパイダーを使用してクロールを開始します。

使用例:

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

check

- 文法: `scrapy check [-l] <spider>`
- Scrapy プロジェクト内で実行させる必要があるか: **はい**

スパイダーコントラクト チェック を実行します。

使用例:

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

(次のページに続く)

(前のページからの続き)

```
  * parse_item

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

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

list

- 文法: `scrapy list`
- Scrapy プロジェクト内で実行させる必要があるか: **はい**

現在のプロジェクトで利用可能なすべてのスパイダーをリストします。 出力は 1 行につき 1 つのスパイダーです。

使用例:

```
$ scrapy list
spider1
spider2
```

edit

- 文法: `scrapy edit <spider>`
- Scrapy プロジェクト内で実行させる必要があるか: **はい**

`EDITOR` 環境変数または、(未設定の場合) *`EDITOR`* 設定で定義されているエディターを使用して、指定されたスパイダーを編集します。

このコマンドは、最も一般的な場合の便利なショートカットとしてのみ提供されています。もちろん開発者は、スパイダーを作成およびデバッグするためのツールまたは IDE を自由に選択できます。

使用例:

```
$ scrapy edit spider1
```

fetch

- 文法: `scrapy fetch <url>`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

Scrapy ダウンローダーを使用して指定された URL をダウンロードし、コンテンツを標準出力に書き込みます。

このコマンドの興味深い点は、ページを取得するのに、あなたのスパイダーを使ってどのようにダウンロードするかを示すということです。たとえば、スパイダーがユーザーエージェントを上書きする USER_AGENT 属性を持っていた場合、上書きしたその属性を使用します。

したがって、このコマンドを使用して、あなたは、あなたのスパイダーが特定のページを取得する方法を「見る」ことができます。

もし、このコマンドがプロジェクトの外部で使用される場合、特定のスパイダーごとの動作は適用されず、デフォルトの Scrapy ダウンローダー設定が使用されます。

コマンドラインオプション:

- `--spider=SPIDER`: スパイダーの自動検出をバイパスし、指定のスパイダーの使用を強制する。
- `--headers`: レスポンス・ボディではなく、レスポンスの HTTP ヘッダーを出力します。
- `--no-redirect`: HTTP 3xx リダイレクトに従わない (デフォルトではそれらに従う)。

使用例:

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

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

view

- 文法: `scrapy view <url>`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

あなたの Scrapy スパイダーが「見ている」ように、指定された URL をブラウザーで開きます。 スパイダーは通常のユーザーとは異なるページを見ることがあるので、これを使用してスパイダーが「見ている」ものを確認し、期待どおりであることを確認できます。

コマンドラインオプション:

- `--spider=SPIDER`: スパイダーの自動検出をバイパスし、指定のスパイダーの使用を強制する。

- `--no-redirect`: HTTP 3xx リダイレクトに従わない (デフォルトではそれらに従う)。

使用例:

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

shell

- 文法: `scrapy shell [url]`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

(指定されている場合) 指定した URL で Scrapy シェルを開始します。URL が指定されていない場合は空です。 また、UNIX スタイルのローカルファイルパス (「./」または「../」プレフィックス付きの相対パス、または絶対ファイルパス) をサポートします。 詳細については、 *Scrapy* **シェル** を参照してください。

コマンドラインオプション:

- `--spider=SPIDER`: スパイダーの自動検出をバイパスし、指定のスパイダーの使用を強制する。
- `-c code`: シェル内で code を評価し、結果を出力して exit する。
- `--no-redirect`: HTTP 3xx リダイレクトに従いません (デフォルトではそれらに従います)。 これは、コマンドラインで引数として渡すことができる URL にのみ影響します。 シェル内に入ると、 `fetch(url)` はデフォルトで HTTP リダイレクトに従います。

使用例:

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

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

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

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

parse

- 文法: `scrapy parse <url> [options]`
- Scrapy プロジェクト内で実行させる必要があるか: **はい**

指定された URL を取得し、 `--callback` オプションで渡されたメソッドを使用して、または指定されていない場合は `parse` を使用して、それを処理するスパイダーで解析します。

コマンドラインオプション:

- `--spider=SPIDER`: スパイダーの自動検出をバイパスし、指定のスパイダーの使用を強制する。
- `--a NAME=VALUE`: スパイダー引数を設定 (繰り返し指定可能)
- `--callback` または `-c`: レスポンスを解析するためのコールバックとして使用するスパイダーメソッド
- `--meta` または `-m`: コールバックリクエストに渡される追加のリクエスト meta。 これは有効な JSON 文字列でなければなりません。 例: –meta='{"foo" : "bar"}'
- `--cbkwargs`: コールバックに渡される追加のキーワード引数。 これは有効な JSON 文字列でなければなりません。 例: –cbkwargs='{"foo" : "bar"}'
- `--pipelines`: パイプラインを介してアイテムを処理する。
- `--rules` または `-r`: *CrawlSpider* ルールを使用して、レスポンスの解析に使用するコールバック (スパイダーメソッド) を検出します。
- `--noitems`: スクレイプしたアイテムを表示しない。
- `--nolinks`: 抽出したリンクを表示しません。
- `--nocolour`: pygments を使用して出力を色付けするのを回避します。
- `--depth `` または ``-d`: リクエストを再帰的に追跡する深さレベル (デフォルト: 1)。
- `--verbose` または `-v`: 各深度レベルの情報を表示。

使用例:

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

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

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

settings

- 文法: `scrapy settings [options]`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

Scrapy 設定の値を取得します。

プロジェクト内で使用すると、プロジェクト設定値が表示されます。それ以外の場合は、Scrapy でのその設定のデフォルト値を表示します。

使用例:

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

runspider

- 文法: `scrapy runspider <spider_file.py>`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

プロジェクトを作成せずに、Python ファイルに含まれるスパイダーを実行します。

使用例:

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

version

- 文法: `scrapy version [-v]`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

Scrapy バージョンを出力します。 `-v` とともに使用すると、Python、Twisted、Platform の情報も出力します。これはバグレポートに役立ちます。

bench

バージョン 0.17 で追加.

- 文法: `scrapy bench`
- Scrapy プロジェクト内で実行させる必要があるか: **いいえ**

簡単なベンチマークテストを実行します。 **ベンチマーキング**

3.1.6 カスタム プロジェクト コマンド

COMMANDS_MODULE 設定を使用して、カスタムプロジェクトコマンドを追加することもできます。 コマンドの実装方法の例については、 scrapy/commands Scrapy コマンドを参照してください。

COMMANDS_MODULE

デフォルト: '' (空文字列)

カスタムの Scrapy コマンドを検索するために使用するモジュール。 これは、Scrapy プロジェクトにカスタムコマンドを追加するために使用されます。

例:

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

setup.py エントリポイントを介してコマンドを登録する。

注釈: これは実験的な機能です。注意して使用してください。

ライブラリ `setup.py` ファイルのエントリポイントに `scrapy.commands` セクションを追加することにより、外部ライブラリから Scrapy コマンドを追加することもできます。

以下の例は `my_command` コマンドを追加します。:

```
from setuptools import setup, find_packages

setup(name='scrapy-mymodule',
  entry_points={
    'scrapy.commands': [
      'my_command=my_scrapy_module.commands:MyCommand',
    ],
  },
 )
```

3.2 スパイダー

スパイダーは、特定のサイト (またはサイトのグループ) のスクレイピング方法を定義するクラスです。クロールの実行方法 (リンクの追跡など) やページから構造化データを抽出する方法 (アイテムのスクレイピングなど) を含

みます。 つまり、スパイダーは、特定のサイト (場合によってはサイトのグループ) のページをクロールおよび解析するためのカスタム動作を定義する場所です。

スパイダーのためのスクレイピング・サイクルは以下の通りです:

1. 最初のリクエストを生成して最初の URL をクロールし、それらのリクエストからダウンロードされたレスポンスで呼び出されるコールバック関数を指定することから始めます。

   実行する最初のリクエストは、(デフォルトで) *start_urls* で指定された URL の *Request* を生成する *start_requests()* メソッドと、リクエストのコールバック関数として *parse* メソッドを呼び出すことによって取得されます。

2. コールバック関数内では、レスポンス (Web ページ) を解析し、抽出されたデータ、 *Item* オブジェクト、*Request* オブジェクト、またはこれらのオブジェクトの反復可能オブジェクトを含む辞書を返します。 これらのリクエストにはコールバックも含まれ (同じコールバックの場合もあります)、Scrapy によってダウンロードされ、指定されたコールバックによってレスポンスが処理されます。

3. コールバック関数内では、通常 **セレクター** を使用してページ内容をパースし、パースしたデータでアイテムを生成します (しかし、パースには、BeautifulSoup、lxml、または任意のメカニズムを使用することもできます)。

4. 最後に、スパイダーから返されたアイテムは通常、データベースに保存されます (**アイテム パイプライン** を使う事もあります) または **フィード・エクスポート** を使用してファイルに書き込まれます。

このサイクルはあらゆる種類のスパイダーに適用されます。そして更に、さまざまな目的の為のさまざまな種類のデフォルト・スパイダーが Scrapy に同梱されています。以降、これらについても説明します。

3.2.1 scrapy.Spider

class scrapy.spiders.**Spider**

これは最も単純なスパイダーであり、他のすべてのスパイダーの継承元となるものです (Scrapy にバンドルされているスパイダーや、自分で作成したスパイダーを含む)。特別な機能は提供しません。 *start_urls* スパイダー属性からリクエストを送信し、結果の各レスポンスに対してスパイダーのメソッド `parse` を呼び出すデフォルトの *start_requests()* 実装を提供するだけです。

name

このスパイダーの名前を定義する文字列。 スパイダー名は、スパイダーが Scrapy によってどのように配置 (およびインスタンス化) されるかであるため、一意でなければなりません。 ただし、同じスパイダーの複数のインスタンスをインスタンス化することを妨げるものはありません。 これは最も重要なスパイダーの属性であり、必須です。

スパイダーが単一のドメインをスクレイピングする場合、一般的な方法は、TLD の有無にかかわらず、ドメインに基づいてスパイダーに名前を付けることです。よって、たとえば、 `mywebsite.com` をクロールするスパイダーは、しばしば `mywebsite` と呼ばれます。

注釈: Python2 では、これは ASCII 文字のみでなければなりません。

allowed_domains
このスパイダーがクロールできるドメインを含む文字列のオプションのリスト。 *OffsiteMiddleware* が有効になっている場合、このリスト (またはそのサブドメイン) で指定されたドメイン名に属さない URL のリクエストは追跡されません。

あなたのターゲット URL が `https://www.example.com/1.html` である場合、リストに `'example.com'` を追加します。

start_urls
特定の URL が指定されていない場合に、スパイダーがクロールを開始する URL のリスト。 したがって、ダウンロードされる最初のページはここにリストされているページになります。 後続の *Request* は、開始 URL に含まれるデータから連続して生成されます。

custom_settings
このスパイダーを実行するときにプロジェクト全体の設定から上書きされる設定の辞書。 インスタンス化の前に設定が更新されるため、クラス属性として定義する必要があります。

利用可能な組み込み設定のリストについては、 **組み込みの設定リファレンス** を参照してください。

crawler
この属性は、クラスを初期化した後に *from_crawler()* クラスメソッドによって設定され、このスパイダーインスタンスがバインドされている *Crawler* オブジェクトにリンクします。

クローラーは、単一のエントリアクセス (拡張機能、ミドルウェア、シグナルマネージャーなど) のために、プロジェクト内の多くのコンポーネントをカプセル化します。 **クローラー** *API* を参照して、それらの詳細を確認してください。

settings
このスパイダーを実行するための構成 (Configuration)。 これは `Settings` のインスタンスです。この主題の詳細な紹介については **設定** トピックを参照してください。

logger
Spider の *name* で作成された Python ロガー。 **スパイダーからのロギング** で説明されているように、これを使用してログメッセージを送信できます。

from_crawler(*crawler*, **args*, ***kwargs*)
これは、Scrapy がスパイダーを作成するために使用するクラスメソッドです。

デフォルトの実装は `__init__()` メソッドのプロキシとして機能し、指定された引数 `args` および名前付き引数 `kwargs` で呼び出すため、おそらくあなたがこれを直接オーバーライドする必要はありません。

それにもかかわらず、このメソッドは新しいインスタンスで *crawler* および *settings* 属性を設定するため、スパイダーのコード内で後からアクセスできます。

パラメータ

- **crawler** (*Crawler* instance) – スパイダーをバインドするクローラー
- **args** (*list*) – `__init__()` メソッドに渡される引数
- **kwargs** (*dict*) – `__init__()` メソッドに渡されるキーワード引数

start_requests()

このメソッドは、このスパイダーの最初のクロール要求で反復可能オブジェクト (iterable) を返す必要があります。 スパイダーがスクレイピングのために開かれると、Scrapy によって呼び出されます。Scrapy はこれを 1 回だけ呼び出すため、ジェネレータとして *start_requests()* を実装しても安全です。

デフォルトの実装は、 *start_urls* の各 URL に対して `Request(url, dont_filter=True)` を生成します。

ドメインのスクレイピングを開始するために使用されるリクエストを変更する場合、これはオーバーライドするメソッドです。 たとえば、POST 要求を使用してログインすることから開始する必要がある場合は、以下の通りです:

```
class MySpider(scrapy.Spider):
    name = 'myspider'

    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
```

parse(*response*)

これは、リクエストでコールバックが指定されていない場合に、ダウンロードされたレスポンスを処理するために Scrapy が使用するデフォルトのコールバックです。

`parse` メソッドは、レスポンスを処理し、スクレイピングされたデータや後続の URL を返します。 他のリクエストのコールバックには、 *Spider* クラスと同じ必要条件があります。

このメソッドは、他のリクエストコールバックと同様に、 *Request* の反復可能オブジェクト (iterable) and/or 辞書 または *Item* オブジェクトを返さなければなりません。

パラメータ **response** (*Response*) – パース対象のレスポンス

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

Spider の *logger* を介してログメッセージを送信するラッパー。後方互換性のために保持されています。 詳細については、 **スパイダーからのロギング** を参照してください。

closed(*reason*)

スパイダーが閉じるときに呼び出されます。 このメソッドは、 *spider_closed* シグナルの signals.connect() へのショートカットを提供します。

ある例を見てみましょう:

```
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.logger.info('A response from %s just arrived!', response.url)
```

単一のコールバックから複数のリクエストとアイテムを返します:

```
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):
        for h3 in response.xpath('//h3').getall():
            yield {"title": h3}

        for href in response.xpath('//a/@href').getall():
            yield scrapy.Request(response.urljoin(href), self.parse)
```

start_urls の代わりに、あなたは、 *start_requests()* を直接使用することができます。データをさらに

構造化するには、 **アイテム** を使用できます:

```
import scrapy
from myproject.items import MyItem

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']

    def start_requests(self):
        yield scrapy.Request('http://www.example.com/1.html', self.parse)
        yield scrapy.Request('http://www.example.com/2.html', self.parse)
        yield scrapy.Request('http://www.example.com/3.html', self.parse)

    def parse(self, response):
        for h3 in response.xpath('//h3').getall():
            yield MyItem(title=h3)

        for href in response.xpath('//a/@href').getall():
            yield scrapy.Request(response.urljoin(href), self.parse)
```

3.2.2 スパイダー引数

スパイダーは、振る舞いを変更する引数を受け取ることができます。 スパイダー引数の一般的な使用法のいくつかは、開始 URL を定義するか、サイトの特定のセクションへのクロールを制限することですが、スパイダーの機能を構成 (configure) するためでも使用できます。

スパイダー引数は、 *crawl* コマンドの -a コマンドライン・オプションを使用して渡します。

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

スパイダーは、 *__init__* メソッド内の引数にアクセスできます。:

```
import scrapy

class MySpider(scrapy.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]
        # ...
```

デフォルトの *__init__* メソッドはスパイダー引数を取り、それらを属性としてスパイダーにコピーします。 上記の例は次のように書くこともできます:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'myspider'

    def start_requests(self):
        yield scrapy.Request('http://www.example.com/categories/%s' % self.category)
```

スパイダー引数は文字列にすぎないことに注意してください。 スパイダー自身はスパイダー引数文字列の解析を行いません。 コマンドラインから start_urls 属性を設定する場合、 ast.literal_eval や json.loads のようなのを使用して自分でリストに落とし込み、それを属性として設定する必要があります。そうしないと、start_urls 文字列を反復して、各文字が個別の URL として認識されることになります (訳注:python によくある落とし穴で、list('hoge') は ['h','o','g','e'] になる)。

有効なユースケースは、 http 認証資格情報 (auth credentials) に使用される *HttpAuthMiddleware* またはユーザエージェントとして使用される *UserAgentMiddleware* を設定することです。

```
scrapy crawl myspider -a http_user=myuser -a http_pass=mypassword -a user_agent=mybot
```

スパイダー引数は、Scrapyd schedule.json API を介して渡すこともできます。 Scrapyd documentation をご覧ください。

3.2.3 汎用スパイダー

Scrapy には、スパイダーのサブクラス化に使用できる便利な汎用スパイダーがいくつか付属しています。 それらの目的は、特定のルールに基づいてサイト上のすべてのリンクをたどったり、サイトマップからクロールしたり、XML/CSV フィードを解析するなど、いくつかの一般的なスクレイピング・パターンに便利な機能を提供することです。

※この節のスパイダー例は、 myproject.items モジュールで宣言された TestItem を含むプロジェクトがあると仮定しています:

```
import scrapy

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

CrawlSpider

class scrapy.spiders.**CrawlSpider**

これは、一連のルールを定義してリンクをたどる便利なメカニズムを提供するため、通常の Web サイトを

クロールするために最も一般的に使用されるスパイダーです。 特定の Web サイトやプロジェクトには最適ではないかもしれませんが、いくつかのケースでは十分に汎用的であるため、このスパイダーから始めて、必要に応じてカスタム機能をオーバーライドしたり、独自のスパイダーを実装したりできます。

Spider から継承された (指定必須の) 属性以外に、この class は新しい属性をサポートします。:

rules

これは、1 つ (または複数) の *Rule* オブジェクトのリストです。 各 *Rule* サイトをクロールするための特定の動作を定義します。 規則オブジェクトについては以下で説明します。 複数の規則が同じリンクに一致する場合、この属性で定義されている順序に従って、一致する最初の規則が使用されます。

このスパイダーにはオーバーライド可能なメソッドもあります:

parse_start_url(*response*)

このメソッドは、start_urls レスポンスに対して呼び出されます。 最初のレスポンスを解析したら、*Item* オブジェクトまたは *Request* オブジェクトまたは、それらを含む反復可能オブジェクト (iterable) を返さなければなりません。

クロール規則

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

link_extractor は、クロールされた各ページからリンクを抽出する方法を定義する **リンク抽出** オブジェクトです。生成された各リンクは、 *Request* オブジェクトを生成するために使用されます。このオブジェクトでは、meta 辞書 (link_text キー) にリンクのテキストを含みます。

callback は、指定のリンク抽出器で抽出された各リンクに対して呼び出される呼び出し可能オブジェクト (callable) または文字列 (この場合、その名前のスパイダー・オブジェクトのメソッドが使用されます) です。 このコールバックは *Response* を最初の引数として受け取り、単一のインスタンスまたは *Item* の反復可能オブジェクト (iterable) または 辞書 そして/または *Request* オブジェクト (またはそのサブクラス)、のいずれかを返す必要があります。上記のように、受け取った *Response* オブジェクトには、その meta 辞書に *Request* を生成したリンクのテキストを含みます (link_text キー)。

> **警告:** *CrawlSpider* はロジックを実装するために parse メソッド自体を使用するため、クロール・スパイダー規則を記述するときは、コールバックとして parse を使用しないでください。つまり、あなたが parse メソッドをオーバーライドしちゃうと、クロール・スパイダーは機能しなくなります。

cb_kwargs は、コールバック関数に渡されるキーワード引数を含む辞書です。

follow は、このルールで抽出された各レスポンスからリンクをたどるかどうかを指定するブール値です。callback が None の場合、 follow のデフォルトは True になります。それ以外の場合、デフォルトは False になります。

process_links は呼び出し可能オブジェクト (callable)、または指定された link_extractor を使用して各レスポンスから抽出されたリンクのリストごとに呼び出される文字列 (この場合、その名前のスパイダー・オブジェクトのメソッドが使用されます) です。これは主にフィルタリングの目的で使用されます。

process_request は、この規則によって抽出されたすべての *Request* に対して呼び出される呼び出し可能オブジェクト (callable)(または文字列、その場合はその名前のスパイダー・オブジェクトのメソッドが使用されます) です。 この呼び出し可能オブジェクト (callable) は、最初の引数としてリクエストを受け取り、2 番目の引数としてリクエストの発信元である *Response* を受け取る必要があります。 Request オブジェクト、または None を返す必要があります (リクエストを除外するため)。

CrawlSpider 例

では、規則を使用した CrawlSpider の例を見てみましょう:

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

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

    rules = (
        # Extract links matching 'category.php' (but not matching 'subsection.php')
        # and follow links from them (since no callback means follow=True by default).
        Rule(LinkExtractor(allow=('category\.php', ), deny=('subsection\.php', ))),

        # Extract links matching 'item.php' and parse them with the spider's method␣
↪parse_item
        Rule(LinkExtractor(allow=('item\.php', )), callback='parse_item'),
    )

    def parse_item(self, response):
        self.logger.info('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()').get()
        item['description'] = response.xpath('//td[@id="item_description"]/text()').
↪get()
        item['link_text'] = response.meta['link_text']
        return item
```

このスパイダーは example.com のホームページのクロールを開始し、カテゴリ・リンクとアイテム・リンクを収集し、後者を parse_item メソッドでパースします。 各アイテムのレスポンスに対して、XPath を使用して HTML からいくつかのデータを抽出し、 *Item* は抽出されたデータで満たされます。

XMLFeedSpider

class scrapy.spiders.**XMLFeedSpider**

XMLFeedSpider は、特定のノード名で XML フィードを反復処理することにより、XML フィードをパースするために設計されています。 イテレータは、「iternodes」、「xml」、および「html」から選択できます。 `xml` および `html` イテレータはパースするために一度 DOM 全体を生成します。そのため、パフォーマンス上の理由から `iternodes` イテレータを使用することをお勧めします。 ただし、不正なマークアップを使用した XML を解析する場合は、イテレータとして `html` を使用すると便利です。

イテレータとタグ名を設定するには、以下のクラス属性を定義する必要があります:

iterator

使用するイテレータを定義する文字列。 以下のいずれかです:

- `'iternodes'` - 正規表現に基づく高速イテレータ
- `'html'` - `Selector` を使用するイテレータ。 これは DOM 解析を使用し、すべての DOM をメモリにロードする必要があることに注意してください。これは大きなフィードの場合に問題になる可能性があります。
- `'xml'` - `Selector` を使用するイテレータ。 これは DOM 解析を使用し、すべての DOM をメモリにロードする必要があることに注意してください。これは大きなフィードの場合に問題になる可能性があります。

デフォルトは `'iternodes'` です。

itertag

反復するノード (または要素) の名前を表す文字列。例:

```
itertag = 'product'
```

namespaces

このスパイダーで処理されるドキュメントで利用可能な名前空間を定義する `(prefix, uri)` タプルのリスト。 `prefix` と `uri` は、 `register_namespace()` メソッドを使用して名前空間を自動的に登録するために使用されます。

あなたは、それから、 *itertag* 属性に名前空間を持つノードを指定できます。

例:

```
class YourSpider(XMLFeedSpider):

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

これらの新しい属性とは別に、このスパイダーには以下のオーバーライド可能なメソッドもあります。:

adapt_response(*response*)
スパイダー・ミドルウェアから到着するとすぐに、スパイダーがパース開始する前に、レスポンスを受信するメソッド。 パース前にレスポンス・ボディを変更するために使用できます。 このメソッドはレスポンスを受け取り、レスポンスを返します (同じ、または別のレスポンスになる可能性があります)。

parse_node(*response*, *selector*)
このメソッドは、指定されたタグ名 (itertag) に一致するノードに対して呼び出されます。 各ノードのレスポンス Selector を受け取ります。 このメソッドのオーバーライドは必須です。 そうしないと、このスパイダーは動作しません。 このメソッドは、 *Item* オブジェクトまたは、 *Request* オブジェクト、またはそれらのいずれかを含む反復可能オブジェクト (iterable) のいずれかを返す必要があります。

process_results(*response*, *results*)
このメソッドは、スパイダーによって返された各結果 (アイテムまたはリクエスト) に対して呼び出され、結果をフレームワーク・コアに返す前に必要な最後の処理 (アイテム ID の設定など) を実行することを目的としています。 結果のリストと、それらの結果を生成したレスポンスを受け取ります。 結果 (アイテムまたはリクエスト) のリストを返す必要があります。

XMLFeedSpider の例

これらのスパイダーは非常に使いやすいので、例を見てみましょう:

```
from scrapy.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):
        self.logger.info('Hi, this is a <%s> node!: %s', self.itertag, ''.join(node.
↪getall()))

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

私たちがここで行ったことは、基本的には、指定した start_urls からフィードをダウンロードし、それぞれの item タグを反復処理し、それらを出力し、いくつかのランダムなデータを *Item* に保存するスパイダーを作成することです。

CSVFeedSpider

class scrapy.spiders.**CSVFeedSpider**

このスパイダーは XMLFeedSpider に非常に似ていますが、ノードではなく行を反復処理する点が異なります。 各反復で呼び出されるメソッドは *parse_row()* です。

delimiter

CSV ファイルの各フィールドを区切る文字 (文字列)。デフォルトは ',' (カンマ)。

quotechar

CSV ファイルの各フィールドを囲い込む文字 (文字列)。デフォルトは '\"' (ダブルクォーテーション)。

headers

CSV ファイルの列名のリスト。

parse_row(*response*, *row*)

CSV ファイルの、レスポンスと、提供された (または検出された) ヘッダー行ごとにキーを持つ、(各行を表す) 辞書を受け取ります。 このスパイダーは、前処理および後処理のために adapt_response および process_results メソッドをオーバーライドする機会も与えます。

CSVFeedSpider 例

いささか前の例に似ているけれども、 *CSVFeedSpider* を使用している例を見てみましょう:

```
from scrapy.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):
        self.logger.info('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.spiders.**SitemapSpider**

SitemapSpider では、 Sitemaps を使用して URL を検出することにより、サイトをクロールできます。

ネストされたサイトマップをサポートし、 robots.txt からサイトマップの URL を検出します。

sitemap_urls

あなたがクロールしたいサイトマップの URL を指定する URL のリスト。

また、 あなたは robots.txt を指定することもできます。robots.txt は、サイトマップの URL をパースするために解析されます。

sitemap_rules

タプル (regex, callback) のリスト。その内訳は以下の通りです:

- regex は、サイトマップから抽出する URL に一致する正規表現です。 regex は文字列またはコンパイル済みの正規表現オブジェクトのいずれかです。
- callback は、正規表現に一致する URL の処理に使用するコールバックです。 callback は文字列 (スパイダーメソッドの名前を示す) または呼び出し可能オブジェクト (callable) です。

例えば:

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

順番に規則の適用を試み、一致する最初の規則のみが使用されます。

あなたがこの属性を省略すると、サイトマップで見つかったすべての URL は parse コールバックで処理されます。

sitemap_follow

追跡すべきサイトマップの正規表現のリスト。 これは、他のサイトマップファイルを指す Sitemap index files を使用するサイト専用です。

デフォルトでは、すべてのサイトマップが追跡されます。

sitemap_alternate_links

ある url の代替リンクをたどるかどうかを指定します。 これらは、同じ url ブロック内で渡される別の言語の同じ Web サイトへのリンクです。

例えば:

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

sitemap_alternate_links を設定すると、両方の URL が取得されます。sitemap_alternate_links を無効にすると、http://example.com/ のみが取得されます。

デフォルトでは sitemap_alternate_links は無効です。

sitemap_filter(*entries*)

これは、属性に基づいてサイトマップ・エントリを選択するためにオーバーライドできるフィルター関数です。

例えば:

```
<url>
    <loc>http://example.com/</loc>
    <lastmod>2005-01-01</lastmod>
</url>
```

私たちは、日付で entries をフィルタリングする sitemap_filter 関数を定義できます:

```
from datetime import datetime
from scrapy.spiders import SitemapSpider

class FilteredSitemapSpider(SitemapSpider):
    name = 'filtered_sitemap_spider'
    allowed_domains = ['example.com']
    sitemap_urls = ['http://example.com/sitemap.xml']

    def sitemap_filter(self, entries):
        for entry in entries:
            date_time = datetime.strptime(entry['lastmod'], '%Y-%m-%d')
            if date_time.year >= 2005:
                yield entry
```

これにより、2005 年以降に変更された entries のみが取得されます。

エントリは、サイトマップ・ドキュメントから抽出された辞書オブジェクトです。 通常、キーはタグ名で、値はその中のテキストです。

重要な注意 :

- loc 属性が必要なため、このタグのないエントリは破棄されます。
- 代替リンクはキー alternate でリストに保存されます (sitemap_alternate_links 参照)
- 名前空間が削除されるため、{namespace}tagname という名前の lxml タグは tagname のみになります。

あなたがこのメソッドを省略すると、サイトマップで見つかったすべてのエントリが処理され、他の属性とその設定を参照します。

SitemapSpider 例

最も単純な例: parse コールバックを使用して、サイトマップを通じて検出されたすべての URL を処理します:

```
from scrapy.spiders import SitemapSpider

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

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

特定のコールバックでいくつかの URL を処理し、別個のコールバックでその他の URL を処理します:

```
from scrapy.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 ...
```

robots.txt ファイルで定義されたサイトマップに従い、URL に /sitemap_shop が含まれるサイトマップのみを追跡します:

```
from scrapy.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 ...
```

SitemapSpider と urls の他のソースを組み合わせます:

```
from scrapy.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 ...
```

3.3 セレクター

あなたが Web ページをスクレイピングする場合、実行する必要がある最も一般的なタスクは、HTML ソースからデータを抽出することです。 これを実現するために利用可能ないくつかのライブラリがあります。:

- BeautifulSoup は、Python プログラマーの間で非常に人気のある Web スクレイピングライブラリであり、HTML コードの構造に基づいて Python オブジェクトを構築し、悪いマークアップも合理的に処理しますが、1 つの欠点があります。遅いんです。
- lxml は、 ElementTree に基づいた Python API を備えた XML パースライブラリ (HTML もパースします) です。(lxml は Python 標準ライブラリの一部ではありません。)

Scrapy には、データを抽出するための独自のメカニズムが備わっています。 これらは、 XPath または CSS 式で指定された HTML ドキュメントの特定の部分を「選択 (select)」するため、セレクター (selector) と呼ばれます。

XPath は、XML ドキュメントでノードを選択するための言語であり、HTML でも使用できます。 CSS は、HTML ドキュメントにスタイルを適用するための言語です。 これらのスタイルを特定の HTML 要素に関連付けるセレクターを定義します。

注釈: Scrapy セレクターは、 parsel ライブラリの薄いラッパーです。 このラッパーの目的は、Scrapy Response オブジェクトとの統合を改善することです。

parsel は、Scrapy なしで使用できるスタンドアロンの Web スクレイピングライブラリです。 内部で lxml ライブ

ラリを使用し、lxml API の上に簡単な API を実装します。 これは、Scrapy セレクターの速度と解析精度が、lxml に非常に似ていることを意味します。

3.3.1 セレクターの使用

セレクターの構築

Response オブジェクトは `.selector` 属性で `Selector` インスタンスを公開します。:

```
>>> response.selector.xpath('//span/text()').get()
'good'
```

XPath と CSS を使用したレスポンスのクエリは非常によく使われるので、レスポンスにはさらに 2 つのショートカットが含まれます。 `response.xpath()` と `response.css()` です。:

```
>>> response.xpath('//span/text()').get()
'good'
>>> response.css('span::text').get()
'good'
```

Scrapy セレクターは、*TextResponse* オブジェクトまたは Unicode 文字列でマークアップを渡す (`text` 引数で) ことで構築された `Selector` クラスのインスタンスです。通常、Scrapy セレクターを手動で作成する必要はありません。 `response` オブジェクトは Spider コールバックで使用できるため、ほとんどの場合、`response.css()` ショートカットと `response.xpath()` ショートカットを使用する方が便利です。 `response.selector` またはこれらのショートカットのいずれかを使用することで、レスポンス・ボディが 1 回だけパースされることを確認することもできます。

ただし、必要に応じて、セレクターを直接使用することができます。 テキストから構築する場合は以下です。:

```
>>> from scrapy.selector import Selector
>>> body = '<html><body><span>good</span></body></html>'
>>> Selector(text=body).xpath('//span/text()').get()
'good'
```

レスポンスから構築する場合、 *HtmlResponse* は *TextResponse* のサブクラスの 1 つです:

```
>>> from scrapy.selector import Selector
>>> from scrapy.http import HtmlResponse
>>> response = HtmlResponse(url='http://example.com', body=body)
>>> Selector(response=response).xpath('//span/text()').get()
'good'
```

セレクターは、入力タイプに基づいて最適なパース・ルール (XML か HTML) を自動的に選択します。

セレクターの使用

セレクターの使用方法を説明するために、私たちは、「Scrapy シェル」(対話的なテストを提供します) と Scrapy ドキュメントサーバーにあるサンプルページを使用します。:

https://docs.scrapy.org/en/latest/_static/selectors-sample1.html

完全を期すために、完全な HTML コードを次に示します。:

```
<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>
```

まず、シェルを開きましょう。:

```
scrapy shell https://docs.scrapy.org/en/latest/_static/selectors-sample1.html
```

次に、シェルがロードされると、レスポンスが `response` シェル変数として使用可能になり、 `response.selector` 属性にそのセレクターが当てはめられます。

HTML を扱っているため、セレクターは自動的に HTML パーサーを使用します。

それでは、そのページの *HTML* **コード** を見て、タイトルタグ内のテキストを選択するための XPath を作成しましょう。:

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

テキストデータを実際に抽出するには、次のようにセレクタ `.get()` または `.getall()` メソッドを呼び出す必要があります。:

```
>>> response.xpath('//title/text()').getall()
['Example website']
>>> response.xpath('//title/text()').get()
'Example website'
```

.get() は常に単一の結果を返します。 複数の一致がある場合、最初の一致のコンテンツが返されます。 一致するものがない場合は None が返されます。 .getall() はすべての結果を含むリストを返します。

CSS セレクターは、CSS3 疑似要素を使用してテキストまたは属性ノードを選択できることに注意してください。:

```
>>> response.css('title::text').get()
'Example website'
```

あなたがご覧のとおり、 .xpath() と .css() メソッドは SelectorList のインスタンスを返します。これは新しいセレクターのリストです。 この API は、ネストされたデータをすばやく選択するために使用できます。:

```
>>> response.css('img').xpath('@src').getall()
['image1_thumb.jpg',
 'image2_thumb.jpg',
 'image3_thumb.jpg',
 'image4_thumb.jpg',
 'image5_thumb.jpg']
```

あなたが最初に一致した要素のみを抽出したい場合は、セレクタ .get() (または以前の Scrapy バージョンで一般的に使用されていたエイリアス .extract_first()) を呼び出すことができます。:

```
>>> response.xpath('//div[@id="images"]/a/text()').get()
'Name: My image 1 '
```

要素が見つからなかった場合は None を返します。:

```
>>> response.xpath('//div[@id="not-exists"]/text()').get() is None
True
```

None の代わりに使用されるデフォルトの戻り値を引数として提供できます。:

```
>>> response.xpath('//div[@id="not-exists"]/text()').get(default='not-found')
'not-found'
```

例えば '@src' のような XPath を使用する代わりに、 Selector の .attrib プロパティを使用して属性を問い合わせることができます。:

```
>>> [img.attrib['src'] for img in response.css('img')]
['image1_thumb.jpg',
 'image2_thumb.jpg',
 'image3_thumb.jpg',
 'image4_thumb.jpg',
 'image5_thumb.jpg']
```

ショートカットとして、 .attrib は SelectorList でも直接利用できます。 最初に一致する要素の属性を返します。:

```
>>> response.css('img').attrib['src']
'image1_thumb.jpg'
```

これは、単一の結果のみが予想される場合に最も役立ちます。例えば ID で選択する場合、または Web ページ上の一意の要素を選択する場合。:

```
>>> response.css('base').attrib['href']
'http://example.com/'
```

今や、私たちは、ベース URL といくつかの画像リンクを取得します。:

```
>>> response.xpath('//base/@href').get()
'http://example.com/'

>>> response.css('base::attr(href)').get()
'http://example.com/'

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

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

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

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

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

CSS セレクターの拡張機能

W3C 標準では、 CSS selectors はテキストノードまたは属性値の選択をサポートしていません。 しかし、これらを選択することは、Web スクレイピングコンテキストでは非常に重要であるため、Scrapy(parsel) はいくつかの **非標準の擬似要素** を実装しています。:

- テキストノードを選択するには `::text` を使用します
- 属性値を選択するには `::attr(name)` を使用します。 *name* は、あなたが値を取得したい属性の名前です

> **警告:** これらの擬似要素は Scrapy/Parsel 固有です。 ほとんどの場合、 lxml や PyQuery などの他のライブラリでは動作しません。

例:

- `title::text` は `<title>` の子孫のテキストノードを選択します。:

```
>>> response.css('title::text').get()
'Example website'
```

- `*::text` は、現在のセレクターコンテキストの全ての子孫テキストノードを選択します。:

```
>>> response.css('#images *::text').getall()
['\n   ',
 'Name: My image 1 ',
 '\n   ',
 'Name: My image 2 ',
 '\n   ',
 'Name: My image 3 ',
 '\n   ',
 'Name: My image 4 ',
 '\n   ',
 'Name: My image 5 ',
 '\n  ']
```

- `foo::text` は、 `foo` 要素が存在するが、テキストを含まない場合 (つまり、テキストが空の場合) の場合、結果を返しません。

```
>>> response.css('img::text').getall()
[]
```

  つまり、 `.css('foo::text').get()` は、要素が存在する場合でも None を返す可能性があることを意味します。 常に文字列が必要な場合は `default=''` 引数を使用してください。:

```
>>> response.css('img::text').get()
>>> response.css('img::text').get(default='')
''
```

- `a::attr(href)` は、リンクにぶら下がってる *href* 属性値を選択します。:

```
>>> response.css('a::attr(href)').getall()
['image1.html',
 'image2.html',
 'image3.html',
 'image4.html',
 'image5.html']
```

注釈: **要素属性の選択** も参照下さい。

注釈: あなたは、これらの擬似要素をチェインさせることはできません。 ただし、実際にはあまり意味がありません。テキストノードには属性がなく、属性値は既に文字列値であり、子ノードはありません。

セレクターを入れ子にする

選択メソッド (`.xpath()` または `.css()`) は同じタイプのセレクターのリストを返すため、これらのセレクターの選択メソッドも呼び出すことができます。 以下に例を示します。:

```
>>> links = response.xpath('//a[contains(@href, "image")]')
>>> links.getall()
['<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>']

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

Link number 0 points to url 'image1.html' and image 'image1_thumb.jpg'
Link number 1 points to url 'image2.html' and image 'image2_thumb.jpg'
Link number 2 points to url 'image3.html' and image 'image3_thumb.jpg'
Link number 3 points to url 'image4.html' and image 'image4_thumb.jpg'
Link number 4 points to url 'image5.html' and image 'image5_thumb.jpg'
```

要素属性の選択

属性の値を取得する方法はいくつかあります。 まず、XPath 構文を使用できます。:

```
>>> response.xpath("//a/@href").getall()
['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
```

XPath 構文にはいくつかの利点があります。これは標準の XPath 機能であり、 `@attributes` は XPath 式の他の部分で使用できます。 属性値でフィルタリングすることが可能です。

Scrapy は、属性値を取得できる CSS セレクター (`::attr(...)`) の拡張機能も提供します。:

```
>>> response.css('a::attr(href)').getall()
['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
```

それに加えて、Selector の `.attrib` プロパティがあります。 XPath または CSS 拡張機能を使用せずに、Python コードで属性を検索する場合に使用できます。:

```
>>> [a.attrib['href'] for a in response.css('a')]
['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
```

このプロパティは SelectorList でも使用できます。 最初に一致した要素の属性を持つ辞書を返します。 セレクターが単一の結果を返すと予想される場合 (たとえば、要素 ID で選択する場合、またはページ上の一意の要素を選択する場合) に使用すると便利です。:

```
>>> response.css('base').attrib
{'href': 'http://example.com/'}
>>> response.css('base').attrib['href']
'http://example.com/'
```

空の SelectorList の `.attrib` プロパティは空です。:

```
>>> response.css('foo').attrib
{}
```

セレクターで正規表現を使う

`Selector` には、正規表現を使用してデータを抽出する `.re()` メソッドもあります。 ただし、 `.xpath()` または `.css()` メソッドを使用するのとは異なり、 `.re()` は Unicode 文字列のリストを返します。 したがって、ネストした `.re()` 呼び出しを構築することはできません。

上記の *HTML* **コード** から画像名を抽出する例を次に示します。:

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

(次のページに続く)

(前のページからの続き)

```
 'My image 2',
 'My image 3',
 'My image 4',
 'My image 5']
```

.re() のために .get() (およびそのエイリアス .extract_first()) に対応する追加のヘルパーがあり、名前は .re_first() です。 これを使用して、最初に一致する文字列のみを抽出します。:

```
>>> response.xpath('//a[contains(@href, "image")]/text()').re_first(r'Name:\s*(.*)')
'My image 1'
```

extract() と extract_first()

あなたが長年の Scrapy ユーザーなら、おそらく .extract() と .extract_first() セレクターメソッドに慣れているでしょう。 多くのブログ投稿とチュートリアルも同様にそれらを使用しています。 これらのメソッドはまだ Scrapy でサポートされており、それらを **非推奨にする計画はありません** 。

けれども、Scrapy の使用法の文書は .get() と .getall() メソッドを使用して記述されるようになりました。私たちは、これらの新しいメソッドは、より簡潔で読みやすいコードになると思います。

次の例は、これらのメソッドが互いにどのようにマッピングされるかを示しています。

1. SelectorList.get() は SelectorList.extract_first() と同じです:

   ```
   >>> response.css('a::attr(href)').get()
   'image1.html'
   >>> response.css('a::attr(href)').extract_first()
   'image1.html'
   ```

2. SelectorList.getall() は SelectorList.extract() と同じです:

   ```
   >>> response.css('a::attr(href)').getall()
   ['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
   >>> response.css('a::attr(href)').extract()
   ['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
   ```

3. Selector.get() は Selector.extract() と同じです:

   ```
   >>> response.css('a::attr(href)')[0].get()
   'image1.html'
   >>> response.css('a::attr(href)')[0].extract()
   'image1.html'
   ```

4. 一貫性のために、リストを返す Selector.getall() もあります:

```
>>> response.css('a::attr(href)')[0].getall()
['image1.html']
```

したがって、主な違いは、.get() と .getall() メソッドの出力はより予測可能なことです。 .get() ``は常に単一の結果、 ``.getall() は常に抽出されたすべての結果のリストを返します。 .extract() メソッドでは、結果がリストであるかどうかは必ずしも明らかではありませんでした。 単一の結果を得るには、.extract() または .extract_first() を呼び出す必要があります。

3.3.2 XPaths で作業する

Scrapy セレクターで XPath を効果的に使用するのに役立つヒントをいくつか紹介します。 XPath にまだ慣れていない場合は、まず、 XPath tutorial をご覧ください。

注釈: いくつかのヒントは this post from ScrapingHub's blog に基づいています。

相対 XPath で作業する

セレクターをネストし、/ で始まる XPath を使用する場合、その XPath はドキュメントの絶対パスであり、呼び出し元のセレクターに対して相対的ではないことに注意してください。

たとえば、 要素内のすべての <p> 要素を抽出するとします。最初に、すべての 要素を取得します:

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

最初は、以下のアプローチを使用したくなるかもしれませんが、実際には 要素内の要素だけでなく、ドキュメント内すべての <p> 要素を抽出するため、間違っています:

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

以下が適切な方法です (XPath の先頭に . が付いていることに注意してください):

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

もう一つの一般的なやり方は、すべての子 <p> を直接抽出することです:

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

相対 XPath の詳細については、XPath 仕様の Location Paths 節を参照してください。

クラスによってクエリーする場合、CSS の使用を検討してください

要素には複数の CSS クラスを含めることができるため、クラスごとに要素を選択する XPath の方法はかなり冗長です:

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

あなたが `@class='someclass'` を使用すると、他のクラスを持つ要素が欠落する可能性があります。それを補うために、単に `contains(@class, 'someclass')` を使用すると、文字列 `someclass` を共有する別のクラス名がある場合、より多くの要素が必要になる可能性があります。

この場合、Scrapy セレクターを使用するとセレクターをチェインできるため、ほとんどの場合、CSS を使用してクラスごとに選択し、それから必要に応じて XPath に切り替えることができます。:

```
>>> 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').getall()
['2014-07-23 19:00']
```

これは、上記の詳細な XPath トリックを使用するよりもクリーンです。 後に続く XPath 式で `.` を使用することを忘れないでください。

//node[1] と (//node)[1] の違いに注意してください

`//node[1]` は、それぞれの親 (parents) の下で最初に発生するすべてのノードを選択します。

`(//node)[1]` ドキュメント内のすべてのノードを選択し、その最初のノードのみを取得します。

例:

```
>>> 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).getall()
```

これは、`<li>` 要素の親 (parent) である全ての要素の子としてある、`<li>` 要素達の最初のを取得します:

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

ドキュメント全体の <li> 要素の最初のものを返します。:

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

これは ‘‘ ‘‘ の子に ‘‘ ‘‘ があるパターン全てが対象となり、それぞれでの最初の ‘‘<li>‘‘ 要素を取得します。:

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

ドキュメント全体の、 の子に <li> があるパターン全てが対象となり、その中で、一番最初の <li> 要素を取得します:

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

条件によるテキストノードの使用

あなたがテキスト内容を XPath 文字列関数 (XPath string function) の引数として使用する必要がある場合、.//text() の使用を避け、代わりに . のみを使用してください。

これは、.//text() 式が **ノードセット** – テキスト要素のコレクション – を生成するためです。 そして、ノードセットが文字列に変換されるとき、つまり、 contains() または starts-with() のような文字列関数への引数として渡されるとき、最初の要素のテキストのみが渡されます。

例:

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

ノードセット から文字列への変換:

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

ただし、文字列に変換された **ノード** は、それ自体のテキストとそのすべての子孫のテキストを一緒にします。

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

したがって、 `.//text()` ノードセットを使用しても、この場合は何も選択されません:

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

しかし、ノードを意味するために `.` を使用すると、動作します:

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

XPath 式の変数

XPath では、 `$somevariable` 構文を使用して、XPath 式の変数を参照できます。 これは、SQL の世界での、クエリの引数を ? のようなプレースホルダーに置き換え、クエリで渡された値で置換されるパラメータークエリまたはプリペアードステートメントに似ているところがあります。

"id" 属性値に基づいて、ハードコーディングせずに要素をマッチする例を次に示します (ハードコーディングする例は前述しました):

```
>>> # `$val` used in the expression, a `val` argument needs to be passed
>>> response.xpath('//div[@id=$val]/a/text()', val='images').get()
'Name: My image 1 '
```

別の例として、5 つの子 <a> を含む タグの "id" 属性を見つけます (ここでは整数として値 5 を渡します):

```
>>> response.xpath('//div[count(a)=$cnt]/@id', cnt=5).get()
'images'
```

All variable references must have a binding value when calling `.xpath()` (otherwise you'll get a `ValueError: XPath error:` exception). This is done by passing as many named arguments as necessary.

Scapy セレクターを駆動するライブラリである parsel には、XPath 変数 (XPath variables) の詳細と例があります。

名前空間 (namespace) の削除

スクレイピングプロジェクトを処理する場合、名前空間を完全に削除し、要素名を操作して、より単純で便利な XPath を作成すると非常に便利です。それには `Selector.remove_namespaces()` メソッドを使用できます。

Python Insider blog atom フィードでこれを説明する例を示しましょう。

まず、あなたがスクレイプしたい URL でシェルを開きます:

```
$ scrapy shell https://feeds.feedburner.com/PythonInsider
```

これがファイルの開始方法です:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet ...
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/"
      xmlns:blogger="http://schemas.google.com/blogger/2008"
      xmlns:georss="http://www.georss.org/georss"
      xmlns:gd="http://schemas.google.com/g/2005"
      xmlns:thr="http://purl.org/syndication/thread/1.0"
      xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.0">
  ...
```

デフォルトの http://www.w3.org/2005/Atom と、 http://schemas.google.com/g/2005 の gd: プレフィックスを使用する別の宣言を含む、いくつかの名前空間宣言を確認できます。

シェルに入ったら、すべての <link> オブジェクトを選択して、機能しないことを確認できます (Atom XML 名前空間がこれらのノードを難読化しているため):

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

しかし、一度、 Selector.remove_namespaces() メソッドを呼び出すと、すべてのノードに名前で直接アクセスできます:

```
>>> response.selector.remove_namespaces()
>>> response.xpath("//link")
[<Selector xpath='//link' data='<link rel="alternate" type="text/html" h'>,
 <Selector xpath='//link' data='<link rel="next" type="application/atom+'>,
 ...
```

あなたは、名前空間削除手順が手動になっていて、デフォルトで常に呼び出されるとは限らないのを疑問に思うかもしれません。これは 2 つの理由によるものです:

1. 名前空間を削除するには、ドキュメント内のすべてのノードを反復して変更する必要があります。これは、Scrapy によってクロールされたすべてのドキュメントに対してデフォルトで実行するのにかなりコストのかかる操作です
2. いくつかの要素名が名前空間間で衝突する場合、実際には名前空間を使用する必要がある場合があります。ただし、これらのケースは非常にまれです。

EXSLT 拡張機能の使用

lxml の上に構築される Scrapy セレクターは、いくつかの EXSLT 拡張をサポートし、XPath 式で使用できる、事前登録されたこれらの名前空間が付属します:

プレフィックス	名前空間	使い方
re	http://exslt.org/regular-expressions	regular expressions 参照
set	http://exslt.org/sets	set manipulation 参照

正規表現

たとえば、 `test()` 関数は、XPath の `starts-with()` または `contains()` が十分でない場合に非常に便利です。

数字で終わるクラス属性を持つリスト項目内のリンクを選択する例:

```
>>> from scrapy import Selector
>>> doc = u"""
... <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').getall()
['link1.html', 'link2.html', 'link3.html', 'link4.html', 'link5.html']
>>> sel.xpath('//li[re:test(@class, "item-\d$")]//@href').getall()
['link1.html', 'link2.html', 'link4.html', 'link5.html']
>>>
```

> **警告:** C ライブラリ `libxslt` は EXSLT 正規表現をネイティブにサポートしていないため、 lxml の実装は Python の `re` モジュールへのフックを使用します。 したがって、XPath 式で正規表現関数を使用すると、パフォーマンスが若干低下する可能性があります。

組 (set) の操作

これらは、たとえばテキスト要素を抽出する前にドキュメントツリーの一部を除外するのに便利です。

アイテムスコープのグループと対応する itemprop を使用して microdata(http://schema.org/Product から取得したサンプルコンテンツ) を抽出する例:

```
>>> doc = u"""
... <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>
```

(次のページに続く)

(前のページからの続き)

```
...   </div>
...   ...
... </div>
... """
>>> sel = Selector(text=doc, type="html")
>>> for scope in sel.xpath('//div[@itemscope]'):
...     print("current scope:", scope.xpath('@itemtype').getall())
...     props = scope.xpath('''
...                 set:difference(./descendant::*/@itemprop,
...                                .//*[@itemscope]/*/@itemprop)''')
...     print("    properties: %s" % (props.getall()))
...     print("")

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

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

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

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

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

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

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

>>>
```

ここでは、まず itemscope 要素を反復処理し、各要素について、すべての itemscope 要素を探し、別の itemscope 内にある要素を除外します。

その他の XPath 拡張機能

Scrapy セレクターは、指定されたすべての HTML クラスを持つノードに対して True を返す、非常に間違った XPath 拡張関数 has-class も提供します。

次の HTML の場合:

```
<p class="foo bar-baz">First</p>
<p class="foo">Second</p>
<p class="bar">Third</p>
<p>Fourth</p>
```

あなたは以下のように使用できます:

```
>>> response.xpath('//p[has-class("foo")]')
[<Selector xpath='//p[has-class("foo")]' data='<p class="foo bar-baz">First</p>'>,
 <Selector xpath='//p[has-class("foo")]' data='<p class="foo">Second</p>'>]
>>> response.xpath('//p[has-class("foo", "bar-baz")]')
[<Selector xpath='//p[has-class("foo", "bar-baz")]' data='<p class="foo bar-baz">First
↪</p>'>]
>>> response.xpath('//p[has-class("foo", "bar")]')
[]
```

XPath `//p[has-class("foo", "bar-baz")]` は、CSS `p.foo.bar-baz` とほぼ同等です。 CSS 探索は XPath に変換されてより効率的に実行されるのに対し、これは問題のすべてのノードに対して呼び出される純粋な Python 関数であるため、ほとんどの場合は、CSS セレクターではありえないぐらい遅いことに注意してください。

また、Parsel は、独自の XPath 拡張機能の追加も簡単にします。

3.3.3 組み込みセレクタリファレンス

セレクター・オブジェクト

SelectorList オブジェクト

3.3.4 例

HTML レスポンスの Selector の例

ここで、いくつかの概念を説明するため `Selector` の例を示します。すべての場合において、このような *`HtmlResponse`* オブジェクトでインスタンス化された `Selector` が既に存在すると仮定します:

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

1. HTML レスポンス・ボディのすべての `<h1>` 要素を選択し、 `Selector` オブジェクトのリスト (つまり、`SelectorList` オブジェクト) を返します:

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

2. HTML レスポンス・ボディのすべての `<h1>` 要素のテキストを抽出し、Unicode 文字列のリストを返します:

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

3. すべての <p> タグを反復処理し、それらのクラス属性を出力します:

```
for node in sel.xpath("//p"):
    print(node.attrib['class'])
```

XML レスポンスでの Selector 例

XmlResponse オブジェクトでインスタンス化された Selector オブジェクトの概念を説明するための例をいくつか示します:

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

1. XML レスポンス・ボディのすべての <product> 要素を選択し、 Selector オブジェクトのリスト (つまり、 SelectorList オブジェクト) を返します:

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

2. 名前空間の登録が必要な Google Base XML feed からすべての価格を抽出します:

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

3.4 アイテム

スクレイピングの主な目標は、非構造化ソース (通常は Web ページ) から構造化データを抽出することです。Scrapy スパイダーは、抽出したデータを Python の辞書として返すことができます。 Python の辞書には便利で使い慣れていますが、構造が欠けています。特に、多くのスパイダーがいる大規模なプロジェクトでは、容易にフィールド名を typo したり、矛盾したデータを返しちゃったりします。

一般的な出力データ形式を定義するために、Scrapy は *Item* クラスを提供します。 *Item* オブジェクトは、スクレイピングされたデータを収集するために使用される単純なコンテナです。 これらは、利用可能なフィールドを宣言するための便利な構文を持つ「辞書のような API」(dictionary-like API) を提供します。

さまざまな Scrapy コンポーネントは、アイテムによって提供される追加情報を使用します。エクスポーターは、宣言されたフィールドを見てエクスポートする列を見つけます。シリアル化は、アイテムのフィールド・メタ・データを使用してカスタマイズできます。 trackref はアイテムのインスタンスを追跡して、メモリ・リーク (*trackref* **を使用したメモリ・リークのデバッグ** 参照) などを見つけるのに役立ちます。

3.4.1 アイテムの宣言

アイテムは、単純なクラス定義構文と *Field* オブジェクトを使用して宣言します。 以下に例があります:

```
import scrapy

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

注釈: Django に精通している人は、Scrapy アイテムが Django Models と同様に宣言されていることに気付くでしょう。ただし、異なるフィールド型の概念がないため、Scrapy アイテムははるかに単純です。

3.4.2 アイテムのフィールド

Field オブジェクトは、各フィールドのメタ・データを指定するために使用されます。 たとえば、上記の例で示した `last_updated` フィールドのシリアル化関数です。

あなたは各フィールドに任意の種類のメタ・データを指定できます。 *Field* オブジェクトが受け入れる値には制限はありません。これと同じ理由で、利用可能なすべてのメタ・データ・キーの参照リストはありません。 *Field* オブジェクトで定義された各キーは異なるコンポーネントで使用でき、それらのコンポーネントのみがそれについて知っています。プロジェクトで他の *Field* キーを定義して使用することもできます。 *Field* オブジェクトの主な目的は、すべてのフィールド・メタ・データを 1 か所で定義する方法を提供することです。 通常、各フィールドに動作が依存するコンポーネントは、特定のフィールド・キーを使用してその動作を構成します。 各コンポーネントで使用されているメタ・データ・キーを確認するには、ドキュメントを参照する必要があります。

アイテムの宣言に使用される *Field* オブジェクトは、クラス属性として割り当てられたままにならないことに注意することが重要です。 代わりに、 *Item.fields* 属性を介してアクセスできます。

3.4.3 アイテムで作業する

ここで、 **先程宣言した** `Product` アイテムを使って、アイテムで実行される一般的なタスクの例をいくつか示します。 API は辞書 API(dict API) に非常に似ていることに気付くでしょう。

アイテムの作成

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

フィールド値の取得

```
>>> 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
```

フィールド値のセット

```
>>> 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'
```

読み込まれたすべての値へのアクセス

読み込まれたすべての値にアクセスするには、典型的な辞書 API(dict API) を使用するだけです:

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

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

アイテムのコピー

アイテムをコピーするには、あなたは最初に浅いコピーとディープ・コピーのどちらを使用するかを決定する必要があります。

アイテムにリストや辞書などのミュータブル (mutable) 値が含まれている場合、浅いコピーは、すべての異なるコピー間で同じミュータブル値への参照を保持します。

たとえば、タグのリストを持つアイテムがあり、そのアイテムの浅いコピーを作成する場合、元のアイテムとコピーの両方に同じタグのリストがあります。 アイテムの 1 つのリストにタグを追加すると、他のアイテムにもタグが追加されます。

それが望ましい振る舞いでない場合は、代わりにディープ・コピーを使用します。

詳細は documentation of the copy module 参照。

アイテムの浅いコピーを作成するには、あなたは、既存のアイテムで `copy()` を呼び出す (`product2 = product.copy()`) か、あるいは、既存のアイテムからアイテム・クラスをインスタンス化 (`product2 = Product(product)`) のいずれかが可能です。

ディープ・コピーを作成するには、代わりに `deepcopy()` を呼び出します (`product2 = product.deepcopy()`)。

その他の一般的な作業

アイテムから辞書を作成する:

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

辞書からアイテムを作成する:

```
>>> 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'
```

3.4.4 アイテムの拡張

あなたは、元のアイテムのサブクラスを宣言することにより、アイテムを拡張できます (フィールドを追加したり、フィールドのメタ・データを変更したりできます)。

例えば:

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

次のように、あなたは、以前のフィールド・メタ・データを使用して値を追加したり、既存の値を変更したりして、フィールド・メタ・データを拡張することもできます:

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

これは、 name フィールドの serializer メタ・データ・キーを追加 (または置換) し、以前に存在したすべてのメタ・データ値を保持します。

3.4.5 アイテム・オブジェクト

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

オプションで指定した引数によって初期化された新しい Item を返します。

アイテムは、コンストラクタを含む標準の辞書 API(dict API) を複製します。 アイテムによって提供される追加の属性は次のとおりです:

fields
読み込まれたフィールドだけでなく、このアイテムの **すべての宣言済みフィールド** を含む辞書。 キーはフィールド名で、値は **アイテム宣言** で使用される *Field* オブジェクトです。

3.4.6 フィールド・オブジェクト

class scrapy.item.**Field**([*arg*])
Field クラスは組み込みの dict クラスの単なるエイリアスであり、追加の機能や属性を提供しません。 言い換えれば、 *Field* オブジェクトは昔ながらの Python 辞書です。 別のクラスを使用して、クラス属性に基づいて **アイテム宣言構文** をサポートします。

3.5 アイテム・ローダー

アイテムローダーは、スクレイピングされた **アイテム** を生成するための便利なメカニズムを提供します。 アイテムは独自の辞書のような API を使用して入力できますが、アイテムローダーは、生の抽出データを割り当てる前に解析するなどの一般的なタスクを自動化することにより、スクレイピングプロセスからアイテムを入力するための、はるかに便利な API を提供します。

言い換えると、 **アイテム** はスクレイピングされたデータの **コンテナ** を提供し、アイテム・ローダーはそのコンテナに **格納** するメカニズムを提供します。

アイテム・ローダーは、スパイダーまたはソース形式 (HTML、XML など)) によってさまざまなフィールド・パース・ルールを拡張およびオーバーライドするための柔軟で効率的かつ簡単なメカニズムを提供するように設計されています。

3.5.1 アイテムを格納するためにアイテム・ローダーを使う

アイテムローダーを使用するには、最初にインスタンス化する必要があります。 dict のようなオブジェクト (Item または dict など) でインスタンス化するか、オブジェクトなしでインスタンス化できます。この場合、Item は、*ItemLoader.default_item_class* 属性で指定された Item クラスを使用してアイテム・ローダー・コンストラクターで自動的にインスタンス化されます。

それから、通常は **セレクター** を使用して、あなたはアイテム・ローダーへの値の収集を開始します。 同じアイテム・フィールドに複数の値を追加できます。アイテム・ローダーは、適切な処理機能を使用して、それらの値を後で「結合」(join) する方法を知っています。

以下は、 **アイテムの章** で宣言された *Product item* を使用した、**スパイダー** 内での典型的なアイテム・ローダーの使用法です:

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

(次のページに続く)

(前のページからの続き)

```
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()
```

そのコードをざっと見ると、ページ内の 2 つの異なる XPath ロケーションから name フィールドが抽出されていることがわかります:

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

いいかえると、データは、 *add_xpath()* メソッドを使用して、2 つの XPath ロケーションから抽出することで収集されます。 これは後で name フィールドに割り当てられるデータです。

その後、同様の呼び出しが price および stock フィールド (後者は *add_css()* メソッドで CSS セレクターを使用) に対して行われ、おわりに last_update フィールドは *add_value()* という別のメソッドを使用して、リテラル値 (today) を直接入力します:

すべてのデータが収集されると、最後に、 *ItemLoader.load_item()* メソッドが呼び出され、実際に返されるのは、以前に *add_xpath()* や *add_css()* や *add_value()* の呼び出しで収集したデータを格納したアイテムです。

3.5.2 入力プロセッサと出力プロセッサ

アイテムローダーには、各 (アイテム) フィールドごとに 1 つの入力プロセッサと 1 つの出力プロセッサが含まれます。入力プロセッサは、(*add_xpath()* または *add_css()* または *add_value()* メソッドを介して) 受信したデータをすぐに処理し、入力プロセッサの結果が収集されてアイテム・ローダー内に保持されます。すべてのデータを収集した後、 *ItemLoader.load_item()* メソッドが呼び出されてデータを格納し、データが格納された *Item* オブジェクトを取得します。その時点で、以前に収集された (および入力プロセッサを使用して処理された) データを使用して、出力プロセッサが呼び出されます。出力プロセッサの結果は、アイテムに割り当てられる最終値です。

(他の任意のフィールドにも同じことが当てはまりますが) とあるフィールドに対して入力プロセッサおよび出力プロセッサがどのように呼び出されるかを例で見てみましょう:

```
l = ItemLoader(Product(), some_selector)
l.add_xpath('name', xpath1) # (1)
l.add_xpath('name', xpath2) # (2)
```

(次のページに続く)

(前のページからの続き)

```
l.add_css('name', css) # (3)
l.add_value('name', 'test') # (4)
return l.load_item() # (5)
```

以下のステップがあります:

1. xpath1 からのデータが抽出され、 name フィールドの **入力プロセッサ** を通過します。 入力プロセッサの結果が収集され、アイテムローダーに保持されます (ただし、アイテムにはまだ割り当てられていません)。
2. xpath2 からのデータが抽出され、ステップ (1) で使用されたのと同じ **入力プロセッサ** を通過します。 入力プロセッサの結果は、(存在する場合、) ステップ (1) で収集されたデータに追加されます。
3. この場合は、データが css CSS セレクターから抽出され、ステップ (1) とステップ (2) で使用された同じ **入力プロセッサ** を通過することを除いて、以前の場合と似ています。 入力プロセッサの結果は、(存在する場合、) ステップ (1) およびステップ (2) で収集されたデータに追加されます。
4. この場合も以前の場合と似ていますが、XPath 式または CSS セレクターから抽出されるのではなく、収集される値が直接割り当てられる点が異なります。 ただし、値は引き続き入力プロセッサを介して渡されます。 この場合、値は反復可能ではなく (not iterable)、そして、入力プロセッサに常に反復可能要素を受け取るため (always receive iterables)、入力プロセッサに渡す前に単一の要素の反復可能要素に変換されます。
5. ステップ (1)〜(4) で収集されたデータは、 name フィールドの **出力プロセッサ** を介して渡されます。 出力プロセッサの結果は、アイテムの name フィールドに割り当てられた値です。

プロセッサは、呼び出し可能なオブジェクトであり、パースされるデータとともに呼び出され、パースされた値を返すことに注意してください。 したがって、任意の関数を入力プロセッサまたは出力プロセッサとして使用できます。 唯一の要件は、イテレータになる位置引数を 1 つ (そして 1 つだけ) 受け入れる必要があることです。

注釈: 入力プロセッサと出力プロセッサは両方とも、イテレータを最初の引数として受け取る必要があります。 これらの関数の出力は何でもかまいません。 入力プロセッサの結果は、(そのフィールドのために) 収集された値を含む (ローダー内の) 内部リストに追加されます。出力プロセッサの結果は、最終的にアイテムに割り当てられる値です。

あなたが単純な関数をプロセッサとして使用する場合は、最初の引数として self を受け取ることを確認してください:

```
def lowercase_processor(self, values):
    for v in values:
        yield v.lower()

class MyItemLoader(ItemLoader):
    name_in = lowercase_processor
```

なぜなら、これは、関数がクラス変数として割り当てられると常にメソッドになり、呼び出されたときに最初の引数としてインスタンスが渡されるためです。 詳細については、this answer on stackoverflow (https://stackoverflow.com/a/35322635) を参照してください。

もう 1 つ注意する必要があるのは、入力プロセッサから返される値が内部 (リスト) で収集され、出力プロセッサに渡されてフィールドに入力されることです。

最後になりましたが、Scrapy には、便宜上、最小限の **一般的に使われるプロセッサ** が組み込まれています。

3.5.3 アイテム・ローダーの宣言

アイテム・ローダーは、クラス定義構文を使用して、アイテムのように宣言されます。以下に例があります:

```
from scrapy.loader import ItemLoader
from scrapy.loader.processors import TakeFirst, MapCompose, Join

class ProductLoader(ItemLoader):

    default_output_processor = TakeFirst()

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

    price_in = MapCompose(unicode.strip)

    # ...
```

ご覧のように、入力プロセッサは _in 接尾辞を使用して宣言され、出力プロセッサは _out 接尾辞を使用して宣言されています。 また、 *ItemLoader.default_input_processor* と *ItemLoader.default_output_processor* 属性を使用して、デフォルトの入出力プロセッサを宣言することもできます。

3.5.4 入力プロセッサと出力プロセッサの宣言

前述のとおり、入力プロセッサと出力プロセッサはアイテム・ローダー定義で宣言できます。この方法で入力プロセッサを宣言することは非常に一般的です。 ただし、使用する入力プロセッサと出力プロセッサを指定できる場所がもう 1 つあります。 **アイテム・フィールド** メタデータです。以下に例を示します:

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

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

(次のページに続く)

(前のページからの続き)

```
        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.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'}
```

入力プロセッサと出力プロセッサの両方の優先順位は次のとおりです:

1. アイテムローダーのフィールド固有の属性: `field_in` および `field_out` (最優先)
2. フィールド・メタデータ (`input_processor` と `output_processor` キー)
3. アイテム・ローダー デフォルト： *`ItemLoader.default_input_processor()`* と *`ItemLoader.default_output_processor()`* (最も低い優先度)

アイテムローダーの再利用と拡張 も参照下さい。

3.5.5 アイテム・ローダー・コンテキスト

アイテムローダーコンテキストは、アイテムローダーのすべての入力プロセッサおよび出力プロセッサ間で共有される任意のキー・値ペアの辞書です。アイテム・ローダーの宣言、インスタンス化、または使用時に渡すことができます。これらは、入出力プロセッサの動作を変更するために使用されます。

たとえば、テキスト値を受け取り、そこから長さを抽出する `parse_length` 関数があるとします:

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

`loader_context` 引数を受け入れることにより、プロセッサ関数はアイテム・ローダーがアイテム・ローダー・コンテキストを受け取ることができることを明示的に伝えています。そのため、アイテム・ローダーはプロセッサ

関数呼び出し時に現在アクティブなコンテキストを渡します。よってプロセッサ関数 (この場合 parse_length) は現在アクティブなコンテキストを使用できます。

アイテムローダーのコンテキスト値を変更する方法はいくつかあります:

1. 現在アクティブなアイテム・ローダー・コンテキスト (*context* 属性) を変更する:

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

2. アイテム・ローダーのインスタンス化時 (アイテム・ローダー・コンストラクターのキーワード引数は、アイテム・ローダー・コンテキストに保存されます):

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

3. アイテム・ローダーの宣言で、アイテム・ローダー・コンテキストを使用したインスタンス化をサポートする入出力プロセッサ用。 MapCompose はそれらの 1 つです:

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

3.5.6 ItemLoader オブジェクト

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

指定されたアイテムを取り込むための新しいアイテムローダーを返します。 項目が指定されていない場合は、 *default_item_class* のクラスを使用して自動的にインスタンス化されます。

selector または response パラメーターでインスタンス化されると、 *ItemLoader* クラスは **セレクター** を使用して Web ページからデータを抽出する便利なメカニズムを提供します。

パラメータ

- **item** (*Item* object) – *add_xpath()* または *add_css()* または *add_value()* への後続の呼び出しを使用して入力するアイテムインスタンス。
- **selector** (Selector object) – *add_xpath()* (代わりに *add_css()*) または *replace_xpath()* (代わりに *replace_css()*) メソッドを使用する場合にデータを抽出するセレクター。
- **response** (*Response* object) – セレクター引数が指定されていない限り、*default_selector_class* を使用してセレクターを構築するために使用されるレスポンス。この場合、この引数は無視されます。

アイテム、セレクター、レスポンス、および残りのキーワード引数はローダー・コンテキストに割り当てられます (*context* 属性からアクセス可能)。

ItemLoader インスタンスには以下のメソッドがあります:

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

指定した processors とキーワード引数により、指定した value を処理します。

利用可能なキーワード引数:

パラメータ **re** (*str or compiled regex*) – プロセッサの前に適用される extract_regex() メソッドを使用して、指定された値からデータを抽出するために使用する正規表現

例:

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

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

処理してから、指定したフィールドに指定した value を追加します。

値は、processors と kwargs を与える事により、最初に *get_value()* を介して渡され、そして、**フィールド入力プロセッサ** を通過し、そして、その結果は、そのフィールドで収集されたデータに追加されます。フィールドにすでに収集されたデータが含まれている場合、新しいデータが追加されます。

与える field_name は None にすることができます。その場合、複数のフィールドの値が追加されます。 そして、処理された値は、field_name が値にマッピングされた辞書でなければなりません。

例:

```
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*)

add_value() に似ていますが、収集したデータを追加する代わりに新しい値に置き換えます。

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

ItemLoader.get_value() に似ていますが、値の代わりに XPath を受け取ります。これは、この *ItemLoader* に関連付けられたセレクターから Unicode 文字列のリストを抽出するために使用されます。

パラメータ

- **xpath** (*str*) – データを抽出するための XPath

- **re** (*str or compiled regex*) – 選択した XPath 領域からデータを抽出するために使用する正規表現

例:

```
# 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*)

ItemLoader.add_value() に似ていますが、値の代わりに XPath を受け取ります。これは、この *ItemLoader* に関連付けられたセレクターから Unicode 文字列のリストを抽出するために使用されます。

kwargs については *get_xpath()* を参照してください。

パラメータ **xpath** (*str*) – データを抽出するための XPath

例:

```
# 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*)

add_xpath() に似ていますが、収集したデータを追加する代わりに置き換えます。

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

ItemLoader.get_value() に似ていますが、値の代わりに CSS セレクターを受け取ります。これは、この *ItemLoader* に関連付けられたセレクターから Unicode 文字列のリストを抽出するために使用されます。

パラメータ

- **css** (*str*) – データを抽出するための CSS セレクター
- **re** (*str or compiled regex*) – 選択した CSS 領域からデータを抽出するために使用する正規表現

例:

```
# 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*)

ItemLoader.add_value() に似ていますが、値の代わりに CSS セレクターを受け取ります。これは、この *ItemLoader* に関連付けられたセレクターから Unicode 文字列のリストを抽出するために使用されます。

kwargs については *get_css()* を参照してください。

パラメータ **css** (*str*) – データを抽出するための CSS セレクター

例:

```
# 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*)

add_css() に似ていますが、収集したデータを追加する代わりに置き換えます。

load_item()

これまでに収集されたデータをアイテムに入力し、それを返します。 収集されたデータは最初に **出力プロセッサ** に渡され、各項目フィールドに割り当てる最終値を取得します。

nested_xpath(*xpath*)

xpath セレクターでネストされたローダーを作成します。 提供されたセレクターは、この *ItemLoader* に関連付けられたセレクターに対して相対的に適用されます。 ネストされたローダーは Item を親の *ItemLoader* と共有するため、*add_xpath()* 、*add_value()* 、*replace_value()* などの呼び出しは期待どおりに動作します。

nested_css(*css*)

CSS セレクターでネストされたローダーを作成します。 提供されたセレクターは、この *ItemLoader* に関連付けられたセレクターに対して相対的に適用されます。 ネストされたローダーは Item を親の *ItemLoader* と共有するため、*add_xpath()* 、*add_value()* 、*replace_value()* などの呼び出しは期待どおりに動作します。

get_collected_values(*field_name*)

指定のフィールドで収集された値を返します。

get_output_value(*field_name*)

指定されたフィールドについて、出力プロセッサを使用してパースされた収集値を返します。 このメソッドは、アイテムの入力や変更を一切行いません。

get_input_processor(*field_name*)

指定フィールドの入力プロセッサを返します。

get_output_processor(*field_name*)

指定フィールドの出力プロセッサを返します。

ItemLoader インスタンスには次の属性があります:

item
このアイテムローダーによってパースされる *Item* オブジェクト。

context
このアイテム・ローダーの現在アクティブな **コンテキスト** 。

default_item_class
コンストラクターで指定されていないときにアイテムをインスタンス化するために使用される Item クラス (またはファクトリー)。

default_input_processor
入力プロセッサを指定しないフィールドに使用するデフォルトの入力プロセッサ。

default_output_processor
出力プロセッサを指定しないフィールドに使用するデフォルトの出力プロセッサ。

default_selector_class
コンストラクターでレスポンスのみが指定された場合、この *ItemLoader* の *selector* を構築するために使用されるクラス。コンストラクターでセレクターが指定されている場合、この属性は無視されます。 この属性はサブクラスでオーバーライドされる場合があります。

selector
データを抽出する `Selector` オブジェクト。 これは、コンストラクターで指定されたセレクター、またはコンストラクターで *default_selector_class* を使用して指定されたレスポンスから作成されたセレクターです。この属性は読み取り専用です。

3.5.7 ネストされたローダー

ドキュメントのサブセクションから関連する値をパースする場合、ネストされたローダーを作成すると便利です。以下のようなページのフッターから詳細を抽出しているとします:

例:

```
<footer>
    <a class="social" href="https://facebook.com/whatever">Like Us</a>
    <a class="social" href="https://twitter.com/whatever">Follow Us</a>
    <a class="email" href="mailto:whatever@example.com">Email Us</a>
</footer>
```

ネストされたローダーがない場合、あなたは抽出する値ごとに完全な xpath(または css) を指定する必要があります。

例:

```
loader = ItemLoader(item=Item())
# load stuff not in the footer
loader.add_xpath('social', '//footer/a[@class = "social"]/@href')
loader.add_xpath('email', '//footer/a[@class = "email"]/@href')
loader.load_item()
```

代わりに、あなたはフッター・セレクターを使用してネストされたローダーを作成し、フッターに関連する値を追加できます。機能は同じですが、あなたはフッター・セレクターの繰り返しを回避できます。

例:

```
loader = ItemLoader(item=Item())
# load stuff not in the footer
footer_loader = loader.nested_xpath('//footer')
footer_loader.add_xpath('social', 'a[@class = "social"]/@href')
footer_loader.add_xpath('email', 'a[@class = "email"]/@href')
# no need to call footer_loader.load_item()
loader.load_item()
```

あなたはローダーを任意にネストでき、xpath または css セレクターで動作します。 一般的なガイドラインとして、ネストされたローダーを使用してコードを単純化します。ネストしないと、パーサーが読みにくくなる可能性があります。

3.5.8 アイテムローダーの再利用と拡張

あなたのプロジェクトが大きくなり、ますます多くのスパイダーを取得するにつれて、メンテナンスは根本的な問題になります。特に、各スパイダーの多くの異なるパースルールを処理する必要がある場合、多くの例外があります。また、共通のプロセッサを再利用したい場合も同様です。

アイテムローダーは、柔軟性を失うことなく、パースルールのメンテナンスの負担を軽減するように設計されていると同時に、それらを拡張およびオーバーライドするための便利なメカニズムを提供します。このため、アイテムローダーは、特定のスパイダー (またはスパイダーのグループ) の違いを処理するために、従来の Python クラスの継承をサポートしています。

たとえば、ある特定のサイトが製品名を 3 つのダッシュ (たとえば `---Plasma TV---`) で囲んでおり、最終製品名でそれらのダッシュをスクレイピングしたくないと仮定します。

ここで、デフォルトの製品アイテムローダー (`ProductLoader`) を再利用して拡張することで、これらのダッシュを削除する方法を次に示します:

```
from scrapy.loader.processors import MapCompose
from myproject.ItemLoaders import ProductLoader

def strip_dashes(x):
```

(次のページに続く)

(前のページからの続き)

```
    return x.strip('-')

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

アイテムローダーの拡張が非常に役立つ別のケースは、XML や HTML などの複数のソース形式がある場合です。XML バージョンでは、 CDATA の出現を削除することができます。方法の例を次に示します:

```
from scrapy.loader.processors 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)
```

そして、それは、入力プロセッサを拡張する典型的な方法です。

出力プロセッサについては、フィールドメタデータで宣言する方が一般的です。これは、通常、(入力プロセッサのように) 特定の各サイトのパースルールではなく、フィールドのみに依存するためです。 **入力プロセッサと出力プロセッサの宣言** も参照してください。

アイテムローダーを拡張、継承、およびオーバーライドする方法は他にもたくさんあります。さまざまなアイテムローダーの階層は、さまざまなプロジェクトにより適しています。 Scrapy はメカニズムのみを提供します。 ローダーコレクションの特定の構成を強制することはありません。それはあなたとプロジェクトのニーズ次第です。

3.5.9 利用可能な組み込みプロセッサ

呼び出し可能な関数を入力プロセッサおよび出力プロセッサとして使用できますが、Scrapy は一般的に使用されるいくつかのプロセッサを提供します。これらについては以下で説明します。 *MapCompose* (通常は入力プロセッサとして使用される) のようなそれらのいくつかは、最終的にパースされた値を生成するために、順番に実行されるいくつかの関数の出力を構成します。

以下に、すべての組み込みプロセッサのリストがあります:

class scrapy.loader.processors.**Identity**

何もしない最も単純なプロセッサ。元の値を変更せずに返します。コンストラクター引数を受け取らず、ローダーコンテキストも受け入れません。

例:

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

class scrapy.loader.processors.**TakeFirst**

受信した値から最初の非ヌル/空でない (non-null/non-empty) 値を返します。したがって、通常は単一値フィールドへの出力プロセッサとして使用されます。コンストラクター引数を受け取らず、ローダーコンテキストも受け入れません。

例:

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

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

コンストラクタで指定されたセパレータで結合した返します。セパレータのデフォルトは `u' '` (空白 1 文字) です。ローダーコンテキストは受け入れません。

デフォルトのセパレータを使用する場合、このプロセッサは次の関数と同等です: `u' '.join`

例:

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

class scrapy.loader.processors.**Compose**(**functions*, ***default_loader_context*)

与えた関数 (達) の組み合わせで構築されたプロセッサー。つまり、このプロセッサの各入力値は最初の関数に渡され、その関数の結果は 2 番目の関数に渡され、最後の関数がこのプロセッサの出力値を返すまで続きます。

デフォルトでは、 None の値で処理を停止します。この動作は、キーワード引数 `stop_on_none = False` を渡すことで変更できます。

例:

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

各関数はオプションで `loader_context` パラメーターを受け取ることができます。実行する場合、このプロセッサーは現在アクティブな **ローダー・コンテキスト** をそのパラメーターを通して渡します。

コンストラクターに渡されるキーワード引数は、各関数呼び出しに渡されるデフォルトのローダーコンテキ

スト値として使用されます。 ただし、関数に渡される最終的なローダーコンテキスト値は、`ItemLoader.context()` 属性を介してアクセス可能な、現在アクティブなローダーコンテキストでオーバーライドされます。

class scrapy.loader.processors.**MapCompose**(**functions*, ***default_loader_context*)

Compose プロセッサと同様に、指定された関数の組み合わせから構築されるプロセッサ。このプロセッサーとの違いは、内部結果が関数間で渡される方法です。これは次のとおりです:

このプロセッサの入力値は **反復可能** であり、最初の関数が各要素に適用されます。これらの関数呼び出しの結果 (要素ごとに 1 つ）が連結されて新しい反復可能要素が作成され、2 番目の関数の適用に使用されます。収集された値のリストの各値に最後の関数が適用されるまで、それが延々続きます。最後の関数の出力値が連結されて、このプロセッサーの出力が生成されます。

特定の各関数は、値または値のリストを返すことができます。これは、他の入力値に適用された同じ関数によって返される値のリストでフラット化されます。関数は `None` を返すこともできます。その場合、その関数の出力は無視され、以後の関数チェーンでのさらなる処理が行われます。

このプロセッサは、(反復可能要素の代わりに、) 単一の値でのみ機能する関数を構成する便利な方法を提供します。このため、データはしばしば **セレクター** の `extract()` メソッドを使用して抽出され、そのため、通常は *MapCompose* プロセッサが入力プロセッサとして使用されます。Unicode 文字列のリストを返します。

以下の例は、それがどのように機能するかを明確にするはずです:

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

Compose プロセッサと同様に、関数はローダーコンテキストを受け取ることができ、コンストラクターのキーワード引数はデフォルトのコンテキスト値として使用されます。 詳細については、*Compose* プロセッサを参照してください。

class scrapy.loader.processors.**SelectJmes**(*json_path*)

コンストラクターに提供された json パスを使用して値を照会し、出力を返します。 実行するには jmespath(https://github.com/jmespath/jmespath.py) が必要です。このプロセッサは、一度に 1 つの入力のみを受け取ります。

例:

```
>>> from scrapy.loader.processors import SelectJmes, Compose, MapCompose
>>> proc = SelectJmes("foo") #for direct use on lists and dictionaries
>>> proc({'foo': 'bar'})
```

(次のページに続く)

(前のページからの続き)

```
'bar'
>>> proc({'foo': {'bar': 'baz'}})
{'bar': 'baz'}
```

Json の操作:

```
>>> import json
>>> proc_single_json_str = Compose(json.loads, SelectJmes("foo"))
>>> proc_single_json_str('{"foo": "bar"}')
'bar'
>>> proc_json_list = Compose(json.loads, MapCompose(SelectJmes('foo')))
>>> proc_json_list('[{"foo":"bar"}, {"baz":"tar"}]')
['bar']
```

3.6 Scrapy シェル

Scrapy シェルは、スパイダーを実行することなく、非常に迅速にスクレイピングコードを試行およびデバッグできる対話型シェルです。これは、データ抽出コードのテストに使用することを目的としていますが、通常の Python シェルでもあるため、実際にはあらゆる種類のコードのテストに使用できます。

シェルは、XPath 式または CSS 式をテストし、それらがどのように機能するか、およびスクレイピングしようとしている Web ページからどのようなデータを抽出するかを確認するために使用されます。すべての変更をテストするためにスパイダーを実行する必要なく、スパイダーを作成している間に式をインタラクティブにテストできます。

Scrapy シェルに親しむと、スパイダーを開発およびデバッグするための非常に素晴らしいツールであることがわかります。

3.6.1 シェルの構成 (configure)

IPython がインストールされている場合、Scrapy シェルは (標準の Python コンソールの代わりに) それを使用します。 IPython コンソールははるかに強力で、とりわけスマートなオートコンプリートとカラー化された出力を提供します。

特に、(IPython が優れている)Unix システムで作業している場合は、 IPython をインストールすることを強くお勧めします。 詳細については、 IPython installation guide を参照してください。

Scrapy は bpython もサポートしており、 IPython が利用できない場合はそれを使用しようとします。

Scrapy の設定により、インストールされているかどうかに関係なく、 `ipython` または `bpython` または、標準の `python` シェルのいずれかを使用するように設定できます。 これには `SCRAPY_PYTHON_SHELL` 環境変数を設定するか、または *scrapy.cfg* で定義することにより行います。

```
[settings]
shell = bpython
```

3.6.2 シェルを起動する

Scrapy シェルを起動するには、次のように *shell* コマンドを使用します:

```
scrapy shell <url>
```

<url> は、あなたがスクレイプしたい URL です。

shell はローカルファイルでも機能します。 これは、Web ページのローカルコピーで遊んでみたい場合に便利です。 *shell* はローカルファイルの次の構文を理解します:

```
# UNIX-style
scrapy shell ./path/to/file.html
scrapy shell ../other/path/to/file.html
scrapy shell /absolute/path/to/file.html

# File URI
scrapy shell file:///absolute/path/to/file.html
```

注釈: 相対ファイルパスを使用する場合は、明示的に指定し、 `./` (または関連する場合は `../`) を先頭に追加します。 `scrapy shell index.html` はあなたの予想どおりに機能しません (これは設計によるものであり、バグではありません)。

なぜなら、 *shell* はファイル URI よりも HTTP URL を優先し、 `index.html` は構文的に `example.com` に似ているため、 *shell* は `index.html` をドメイン名とみなして DNS ルックアップし、エラーをトリガします:

```
$ scrapy shell index.html
[ ... scrapy shell starts ... ]
[ ... traceback ... ]
twisted.internet.error.DNSLookupError: DNS lookup failed:
address 'index.html' not found: [Errno -5] No address associated with hostname.
```

shell は `index.html` というファイルが現在のディレクトリに存在するかどうかを事前にテストしません。繰り返しますが、明示的に指定してください。

3.6.3 シェルの使用

Scrapy シェルは、全くもって Python コンソール (または、使用可能な場合は IPython コンソール) です。そして、便利な追加のショートカット機能を提供します。

利用可能なショートカット

- `shelp()` - 利用可能なオブジェクトとショートカットのリストを含むヘルプを出力します
- `fetch(url[, redirect=True])` - 指定された URL から新しいリクエストを取得し、それに応じてすべての関連オブジェクトを更新します。 HTTP 3xx リダイレクトの後に `redirect=False` を渡さないようにオプションで要求できます
- `fetch(request)` - 特定のリクエストから新しいレスポンスを取得し、それに応じてすべての関連オブジェクトを更新します。
- `view(response)` - 検査のために、指定されたレスポンスをローカル Web ブラウザーで開きます。 これにより、外部リンク (画像やスタイルシートなど) が正しく表示されるように、 <base> tag がレスポンス・ボディに追加されます。 ただし、これによりコンピューターに一時ファイルが作成され、自動的には削除されないことに注意してください。

利用可能な Scrapy オブジェクト

Scrapy シェルは、ダウンロードしたページから、 *`Response`* オブジェクトや、(HTML コンテンツと XML コンテンツの両方のために) `Selector` オブジェクトなどの便利なオブジェクトを自動的に作成します。

それらのオブジェクトは次のとおりです:

- `crawler` - 現在の *`Crawler`* オブジェクト。
- `spider` - URL を処理することが知られているスパイダー、または現在の URL 用にスパイダーが見つからない場合は *`Spider`* オブジェクト
- `request` - 最後に取得したページの *`Request`* オブジェクト。 *`replace()`* を使用してこのリクエストを変更するか、 `fetch` ショートカットを使用して、(シェルを離れずに) 新しいリクエストを取得できます。
- `response` - 最後に取得したページを含む *`Response`* オブジェクト
- `settings` - 現在の *Scrapy* **設定**

3.6.4 シェル セッション の例

ここで、 https://scrapy.org ページをスクレイピングすることから始めて、 次に、 https://reddit.com ページのスクレイピングに進む、典型的なシェルセッションの例を以下に示します。 そして最後に、(Reddit) リクエストメソッ

ドを POST に変更し、エラーを取得して再フェッチします。セッションを終了するには、Ctrl-D(Unix システムの場合)、Windows では Ctrl-Z を入力します。

これらのページは静的ではなく、テストするまでに変更されている可能性があるため、ここで抽出したデータは同じではない可能性があることに注意してください。 この例の唯一の目的は、Scrapy シェルの仕組みを理解してもらうことです。

まず、私たちはシェルを起動します:

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

次に、シェルは (Scrapy ダウンローダーを使用して)URL を取得し、使用可能なオブジェクトと便利なショートカットのリストを出力します (これらの行はすべて `[s]` 接頭辞で始まることがわかります):

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x7f07395dd690>
[s]   item       {}
[s]   request    <GET https://scrapy.org>
[s]   response   <200 https://scrapy.org/>
[s]   settings   <scrapy.settings.Settings object at 0x7f07395dd710>
[s]   spider     <DefaultSpider 'default' at 0x7f0735891690>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default,␣
↪redirects are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser

>>>
```

その後、オブジェクトで遊んでみましょう:

```
>>> response.xpath('//title/text()').get()
'Scrapy | A Fast and Powerful Scraping and Web Crawling Framework'

>>> fetch("https://reddit.com")

>>> response.xpath('//title/text()').get()
'reddit: the front page of the internet'

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

>>> fetch(request)

>>> response.status
404
```

(次のページに続く)

(前のページからの続き)

```
>>> from pprint import pprint

>>> pprint(response.headers)
{'Accept-Ranges': ['bytes'],
 'Cache-Control': ['max-age=0, must-revalidate'],
 'Content-Type': ['text/html; charset=UTF-8'],
 'Date': ['Thu, 08 Dec 2016 16:21:19 GMT'],
 'Server': ['snooserv'],
 'Set-Cookie': ['loid=KqNLou0V9SKMX4qb4n; Domain=reddit.com; Max-Age=63071999; Path=/;␣
↪expires=Sat, 08-Dec-2018 16:21:19 GMT; secure',
                'loidcreated=2016-12-08T16%3A21%3A19.445Z; Domain=reddit.com; Max-
↪Age=63071999; Path=/; expires=Sat, 08-Dec-2018 16:21:19 GMT; secure',
                'loid=vi0ZVe4NkxNWdlH7r7; Domain=reddit.com; Max-Age=63071999; Path=/;␣
↪expires=Sat, 08-Dec-2018 16:21:19 GMT; secure',
                'loidcreated=2016-12-08T16%3A21%3A19.459Z; Domain=reddit.com; Max-
↪Age=63071999; Path=/; expires=Sat, 08-Dec-2018 16:21:19 GMT; secure'],
 'Vary': ['accept-encoding'],
 'Via': ['1.1 varnish'],
 'X-Cache': ['MISS'],
 'X-Cache-Hits': ['0'],
 'X-Content-Type-Options': ['nosniff'],
 'X-Frame-Options': ['SAMEORIGIN'],
 'X-Moose': ['majestic'],
 'X-Served-By': ['cache-cdg8730-CDG'],
 'X-Timer': ['S1481214079.394283,VS0,VE159'],
 'X-Ua-Compatible': ['IE=edge'],
 'X-Xss-Protection': ['1; mode=block']}
>>>
```

3.6.5 スパイダーからシェルを呼び出してレスポンスを検査する

あなたが期待するレスポンスがそこに到達していることを確認するためだけに、スパイダーの特定のポイントで処理されているレスポンスを検査したい場合があります。

これは `scrapy.shell.inspect_response` 関数を使用することで実現できます。

スパイダーから呼び出す方法の例を次に示します:

```
import scrapy


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

(次のページに続く)

(前のページからの続き)

```
        "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.
```

スパイダーを実行すると、次のようなものが得られます:

```
2014-01-23 17:48:31-0400 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://example.
↪com> (referer: None)
2014-01-23 17:48:31-0400 [scrapy.core.engine] 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'
```

そこから、抽出コードが機能しているかどうかを確認できます:

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

Orz。期待したとおりではありません。 したがって、あなたは Web ブラウザーでレスポンスを開き、それが期待したレスポンスかどうかを確認できます:

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

最後に、Ctrl-D(Windows では Ctrl-Z) を押してシェルを終了し、クロールを再開します:

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

Scrapy エンジンはシェルによってブロックされているため、ここでは `fetch` ショートカットを使用できないことに注意してください。 ただし、上記のように、シェルを離れた後、スパイダーは停止した場所からクロールを続けます。

3.7 アイテム・パイプライン

アイテムがスパイダーによってスクレイプされた後、アイテムはアイテム・パイプラインに送信され、アイテム・パイプラインは順次実行される複数のコンポーネントを介してアイテムを処理します。

各アイテム・パイプライン・コンポーネント (アイテム・パイプラインとも呼ばれる) は、単純なメソッドを実装する Python クラスです。 それらはアイテムを受け取り、それに対してアクションを実行します。また、そのアイテムがパイプラインで処理を継続されるか、そのアイテムをドロップして処理しなくなくなるかを決定します。

アイテムパイプラインの一般的な用途は次のとおりです:

- HTML データの洗浄 (cleansing)
- スクレイプしたグデータの検証 (アイテムに特定のフィールドが含まれていることを確認)
- 重複のチェック (そして重複分をドロップする)
- スクレイプされたアイテムをデータベースに保存する

3.7.1 あなた独自のアイテム・パイプラインを書く

各アイテム・パイプライン・コンポーネントは、Python クラスであり、次のメソッドを実装する必要があります:

process_item(*self*, *item*, *spider*)
このメソッドは、すべてのアイテム・パイプライン・コンポーネントに対して呼び出されます。 *process_item()* は、データの辞書を返す、あるいは *Item* (または任意の子孫クラス) オブジェクトを返す、あるいは Twisted Deferred を返す、あるいは *DropItem* 例外を発生させる、のいずれかを行う必要があります。ドロップしたアイテムは、それ以降のパイプライン・コンポーネントにでは処理しません。

パラメータ

- **item** (*Item* object or a dict) – スクレイプされたアイテム
- **spider** (*Spider* object) – アイテムをスクレイプしたスパイダー

さらに、以下のメソッドも実装できます:

open_spider(*self*, *spider*)
このメソッドは、スパイダーがオープンされたときに呼び出されます。

パラメータ **spider** (*Spider* object) – オープンされたスパイダー

close_spider(*self*, *spider*)
このメソッドはスパイダーがクローズされたときに呼び出されます。

パラメータ **spider** (*Spider* object) – クローズされたスパイダー

from_crawler(*cls*, *crawler*)

存在する場合、このクラスメソッドは、 *Crawler* からパイプライ・ンインスタンスを作成するために呼び出されます。パイプラインの新しいインスタンスを返す必要があります。クローラー・オブジェクトは、設定やシグナルなどのすべての Scrapy コアコンポーネントへのアクセスを提供します。 つまり、それはパイプラインがそれらにアクセスし、その機能を Scrapy にフックする方法です。

パラメータ **crawler** (*Crawler* object) – このパイプラインを使用するクローラー

3.7.2 アイテム・パイプライン例

価格の検証と価格のないアイテムのドロップ

VAT(訳注:付加価値税) を含まないアイテムの価格 (price_excludes_vat 属性) を調整し、価格を含まないアイテムをドロップする、次の仮想パイプラインを見てみましょう:

```
from scrapy.exceptions import DropItem

class PricePipeline(object):

    vat_factor = 1.15

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

JSON ファイルにアイテムを書き込む

次のパイプラインは、(すべてのスパイダーからの) すべてのスクレイプされたアイテムを、JSON 形式でシリアル化された行ごとに 1 つのアイテムを含む単一の items.jl ファイルに保存します:

```
import json

class JsonWriterPipeline(object):

    def open_spider(self, spider):
        self.file = open('items.jl', 'w')

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

    def process_item(self, item, spider):
```

(次のページに続く)

(前のページからの続き)

```
        line = json.dumps(dict(item)) + "\n"
        self.file.write(line)
        return item
```

注釈: JsonWriterPipeline の目的は、アイテム・パイプラインの記述方法を紹介することだけです。すべてのスクレイプ・アイテムを JSON ファイルに保存する場合は、 **フィード・エクスポート** を使用する必要があります。

MongoDB にアイテムを書き込む

この例では、私たちは pymongo を使用して MongoDB にアイテムを書き込みます。MongoDB アドレスとデータベース名は、Scrapy 設定で指定します。MongoDB コレクションは、アイテム・クラスに基づいて名前が付けられます。

この例の主なポイントは、 *from_crawler()* メソッドの使用方法と、リソースを適切にクリーンアップする方法を示すことです:

```
import pymongo

class MongoPipeline(object):

    collection_name = 'scrapy_items'

    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):
        self.db[self.collection_name].insert_one(dict(item))
        return item
```

アイテムのスクリーンショットをとる

このデモンストレーションは、 *process_item()* メソッドから Deferred を返す方法を示しています。 Splash を使用して、アイテムの URL のスクリーンショットをレンダリングします。 パイプラインは、ローカルで実行されている Splash のインスタンスにリクエストを行います。 リクエストがダウンロードされ、Deferred コールバックが起動した後、アイテムをファイルに保存し、ファイル名をアイテムに追加します。

```
import scrapy
import hashlib
from urllib.parse import quote


class ScreenshotPipeline(object):
    """Pipeline that uses Splash to render screenshot of
    every Scrapy item."""

    SPLASH_URL = "http://localhost:8050/render.png?url={}"

    def process_item(self, item, spider):
        encoded_item_url = quote(item["url"])
        screenshot_url = self.SPLASH_URL.format(encoded_item_url)
        request = scrapy.Request(screenshot_url)
        dfd = spider.crawler.engine.download(request, spider)
        dfd.addBoth(self.return_item, item)
        return dfd

    def return_item(self, response, item):
        if response.status != 200:
            # Error happened, return item.
            return item

        # Save screenshot to file, filename will be hash of url.
        url = item["url"]
        url_hash = hashlib.md5(url.encode("utf8")).hexdigest()
        filename = "{}.png".format(url_hash)
        with open(filename, "wb") as f:
            f.write(response.body)

        # Store filename in item.
        item["screenshot_filename"] = filename
        return item
```

重複フィルター

重複するアイテムを探し、すでに処理されたアイテムをドロップするフィルター。アイテムには一意の ID がありますが、スパイダーは同じ ID の複数のアイテムを返します:

```
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
```

3.7.3 アイテム・パイプライン・コンポーネントのアクティブ化

アイテム・パイプライン・コンポーネントをアクティブにするには、次の例のように、そのクラスを *ITEM_PIPELINES* 設定に追加する必要があります:

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

あなたがこの設定でクラスに割り当てる整数値は、それらが実行される順序を決定します。項目は、低い値のクラスから高い値のクラスへと通過します。 これらの数値は 0〜1000 の範囲で定義するのが慣例です。

3.8 フィード・エクスポート

バージョン 0.10 で追加.

スクレーパーを実装するときに最も頻繁に必要な機能の 1 つは、スクレイピングデータを適切に保存できることです。これは、他のシステムで使用されるスクレイピングデータ (一般に「エクスポートフィード」) を含む「エクスポートファイル」を生成することを意味します。

Scrapy はこの機能をすぐに使えるフィード・エクスポートで提供します。これにより、複数のシリアル化形式とストレージバックエンドを使用して、スクレイプされたアイテムを含むフィードを生成できます。

3.8.1 シリアル化形式

スクレイピングされたデータをシリアル化するために、フィードのエクスポートは **アイテム・エクスポーター** を使用します。 これらの形式はすぐに使用できます:

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

しかし、あなたは、 *FEED_EXPORTERS* 設定を通してサポートされているフォーマットを拡張することもできます。

JSON

- *FEED_FORMAT*: `json`
- 使用されるエクスポーター: *JsonItemExporter*
- JSON を大きなフィードで使用している場合は、 **注意** ：ref： **この警告**<*json-with-large-data*> を参照してください。

JSON lines

- *FEED_FORMAT*: `jsonlines`
- 使用されるエクスポーター: *JsonLinesItemExporter*

CSV

- *FEED_FORMAT*: `csv`
- 使用されるエクスポーター: *CsvItemExporter*
- エクスポートする列とその順序を指定するには、 *FEED_EXPORT_FIELDS* を使用します。 他のフィードエクスポーターもこのオプションを使用できますが、他の多くのエクスポート形式とは異なり、CSV は固定ヘッダーを使用するため、CSV では重要です。

XML

- *FEED_FORMAT*: `xml`
- 使用されるエクスポーター: *XmlItemExporter*

Pickle

- *FEED_FORMAT*: `pickle`

- 使用されるエクスポーター: *PickleItemExporter*

Marshal

- *FEED_FORMAT*: marshal
- 使用されるエクスポーター: MarshalItemExporter

3.8.2 ストレージ

フィード・エクスポートを使用する場合、URI を使用して (*FEED_URI* 設定を使用して) フィードを保存する場所を定義します。フィード・エクスポートは、URI スキームで定義された複数のストレージバックエンドタイプをサポートします。

すぐに使用できるストレージバックエンドは次のとおりです:

- **ローカル・ファイルシステム**
- *FTP*
- *S3* (botocore または boto が必要です)
- **標準出力**

必要な外部ライブラリが利用できない場合、一部のストレージバックエンドは利用できません。 たとえば、S3 バックエンドは、botocore または boto ライブラリがインストールされている場合にのみ使用できます (Scrapy は Python2 でのみ boto をサポートします)。

3.8.3 ストレージ URI パラメーター

ストレージ URI には、フィードの作成時に置換されるパラメーターを含めることもできます。これらのパラメーターは次のとおりです:

- `%(time)s` - フィードの作成時にタイムスタンプに置き換えられます
- `%(name)s` - スパイダー名に置き換えられます

他の名前付きパラメーターは、同じ名前のスパイダー属性に置き換えられます。 たとえば、フィードが作成された瞬間に `%(site_id)s` は `spider.site_id` 属性に置き換えられます。

以下に例を示します:

- スパイダーごとに 1 つのディレクトリを使用して FTP に保存します:
  - `ftp://user:password@ftp.example.com/scraping/feeds/%(name)s/%(time)s.json`

- スパイダーごとに 1 つのディレクトリを使用して S3 に保存します:
  - `s3://mybucket/scraping/feeds/%(name)s/%(time)s.json`

3.8.4 ストレージ・バックエンド

ローカル・ファイルシステム

フィードはローカルファイルシステムに保存されます。

- URI スキーム: `file`
- URI 例: `file:///tmp/export.csv`
- 必要な外部ライブラリ: なし

ローカルファイルシステムストレージ (のみ) の場合、 `/tmp/export.csv` のように絶対パスを指定する場合、スキームを省略できます。 ただし、これは Unix システムでのみ機能します。

FTP

フィードは FTP サーバーに保存されます。

- URI スキーム: `ftp`
- URI 例: `ftp://user:pass@ftp.example.com/path/to/export.csv`
- 必要な外部ライブラリ: なし

FTP は、2 つの異なる接続モードをサポートしています。アクティブまたはパッシブ (active or passive) です。Scrapy はデフォルトでパッシブ接続モードを使用します。 代わりにアクティブな接続モードを使用するには、*FEED_STORAGE_FTP_ACTIVE* 設定を `True` に設定します。

S3

フィードは Amazon S3 に保存されます。

- URI スキーム: `s3`
- URI 例:
  - `s3://mybucket/path/to/export.csv`
  - `s3://aws_key:aws_secret@mybucket/path/to/export.csv`
- 必要な外部ライブラリ: botocore (Python2 および Python3) または boto (Python2 のみ)

AWS 認証情報は、URI でユーザー/パスワードとして渡すことができます。または、以下の設定を介して渡すことができます:

- *AWS_ACCESS_KEY_ID*
- *AWS_SECRET_ACCESS_KEY*

あなたは、この設定を使用して、エクスポートされたフィードのカスタム ACL を定義することもできます:

- *FEED_STORAGE_S3_ACL*

標準出力

フィードは、Scrapy プロセスの標準出力に書き込まれます。

- URI スキーム: `stdout`
- URI 例: `stdout:`
- 必要な外部ライブラリ: なし

3.8.5 設定

これらは、フィードのエクスポートの構成 (configuration) に使用される設定です:

- *FEED_URI* (必須)
- *FEED_FORMAT*
- *FEED_STORAGES*
- *FEED_STORAGE_FTP_ACTIVE*
- *FEED_STORAGE_S3_ACL*
- *FEED_EXPORTERS*
- *FEED_STORE_EMPTY*
- *FEED_EXPORT_ENCODING*
- *FEED_EXPORT_FIELDS*
- *FEED_EXPORT_INDENT*

FEED_URI

デフォルト: `None`

エクスポートフィードの URI。 サポートされている URI スキームについては、 **ストレージ・バックエンド** を参照してください。

この設定は、フィード・エクスポートを有効にするために必要です。

FEED_FORMAT

フィードに使用されるシリアル化形式。 設定可能な値については、 **シリアル化形式** を参照してください。

FEED_EXPORT_ENCODING

デフォルト: `None`

フィードに使用されるエンコード。

設定されていないか `None` (デフォルト) に設定されている場合、歴史的な理由で、安全な数値エンコーディング (`\uXXXX` シーケンス) を使用する JSON 出力を除く、すべてで UTF-8 を使用します。

JSON にも UTF-8 が必要な場合は `utf-8` を使用します。

FEED_EXPORT_FIELDS

デフォルト: `None`

エクスポートするフィールドのリスト。オプション。 例: `FEED_EXPORT_FIELDS = ["foo", "bar", "baz"]`

FEED_EXPORT_FIELDS オプションを使用して、エクスポートするフィールドとその順序を定義します。

FEED_EXPORT_FIELDS が空または None (デフォルト) の場合、Scrapy は辞書または *Item* のサブクラスで定義されたフィールドを使用し、スパイダーが生成します。

エクスポーターがフィールドの固定セットを必要とする場合 (*CSV* エクスポート形式の場合)、FEED_EXPORT_FIELDS が空または None の場合、Scrapy はエクスポートされたデータからフィールド名を推測しようとします。今のところは、最初のアイテムのフィールド名を使用しています。

FEED_EXPORT_INDENT

デフォルト: `0`

各レベルで出力をインデントするために使用されるスペースの量。 `FEED_EXPORT_INDENT` が負でない整数の場合、配列要素とオブジェクトメンバーはそのインデントレベルできれいに印刷されます。インデントレベル `0` (デフォルト)、または負の場合、各アイテムは新しい行に配置されます。 `None` は、最もコンパクトな表現を選択します。

現在、 *JsonItemExporter* と *XmlItemExporter* のみ実装されています。つまり、 .json または .xml にエクスポートする場合です。

FEED_STORE_EMPTY

デフォルト: False

空のフィード (つまり、アイテムのないフィード) をエクスポートするかどうか。

FEED_STORAGES

デフォルト: {}

プロジェクトでサポートされている追加のフィードストレージバックエンドを含む辞書。 キーは URI スキームであり、値はストレージクラスへのパスです。

FEED_STORAGE_FTP_ACTIVE

デフォルト: False

フィードを FTP サーバーにエクスポートするときにアクティブ接続モードを使用するか (True)、代わりにパッシブ接続モードを使用するか (False 、デフォルト)。

FTP 接続モードについては、アクティブ FTP とパッシブ FTP の違いは何ですか？ (What is the difference between active and passive FTP?) を参照してください。

FEED_STORAGE_S3_ACL

デフォルト: '' (空文字列)

プロジェクトによって Amazon S3 にエクスポートされたフィードのカスタム ACL を含む文字列。

利用可能な値の完全なリストについては、Amazon S3 ドキュメントの Canned ACL 節にアクセスしてください。

FEED_STORAGES_BASE

デフォルト:

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

Scrapy がサポートする組み込みのフィードストレージバックエンドを含む辞書。 *FEED_STORAGES* で URI スキームに None を割り当てることで、これらのバックエンドを無効にできます。 たとえば、組み込みの FTP ストレージバックエンドを無効にするには、以下を (置換なしで) ‘‘ settings.py‘‘ に配置します:

```
FEED_STORAGES = {
    'ftp': None,
}
```

FEED_EXPORTERS

デフォルト: {}

プロジェクトでサポートされている追加のエクスポーターを含む辞書。 キーはシリアル化形式で、値は **アイテム・エクスポーター** クラスへのパスです。

FEED_EXPORTERS_BASE

デフォルト:

```
{
    'json': 'scrapy.exporters.JsonItemExporter',
    'jsonlines': 'scrapy.exporters.JsonLinesItemExporter',
    'jl': 'scrapy.exporters.JsonLinesItemExporter',
    'csv': 'scrapy.exporters.CsvItemExporter',
    'xml': 'scrapy.exporters.XmlItemExporter',
    'marshal': 'scrapy.exporters.MarshalItemExporter',
    'pickle': 'scrapy.exporters.PickleItemExporter',
}
```

Scrapy でサポートされている組み込みのフィードエクスポーターを含む辞書。 *FEED_EXPORTERS* のシリアル化形式に None を割り当てることで、これらのエクスポーターを無効にできます。 たとえば、組み込みの CSV エクスポーターを無効にするには、以下を (置換なしで)、 `settings.py` に配置します:

```
FEED_EXPORTERS = {
    'csv': None,
}
```

3.9 リクエストとレスポンス

Scrapy は、Web サイトのクロールに *Request* と *Response* オブジェクトを使用します。

通常、 *Request* オブジェクトはスパイダーで生成され、ダウンローダーに到達するまでシステム内をあちこち旅行 (pass across) します。ダウンローダーはリクエストを実行し、リクエストを発行したスパイダーに *Response* オブジェクトを返します。

Request クラスと *Response* クラスの両方には、基本クラスでは必要のない機能を追加するサブクラスがあります。これらについては、 *Request* **のサブクラス** と *Response* **のサブクラス** で説明しています。

3.9.1 Request オブジェクト

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

Request オブジェクトは HTTP リクエストを表します。これは通常スパイダーで生成され、ダウンローダーによって実行され、そして、 *Response* が生成されます。

パラメータ

- **url** (*string*) – このリクエストの URL
- **callback** (*callable*) – 最初のパラメーターとしてこのリクエストのレスポンス (ダウンロード後) に呼び出される関数。詳細については、以下の **追加のデータをコールバック関数に渡す** を参照してください。リクエストでコールバックが指定されていない場合、スパイダーの *parse()* メソッドが使用されます。 処理中に例外が発生した場合、代わりにエラーバック (errback) が呼び出されることに注意してください。
- **method** (*string*) – このリクエストの HTTP メソッド。デフォルトは `'GET'` です。
- **meta** (*dict*) – *Request.meta* 属性の初期値。指定すると、このパラメーターに渡された辞書は浅いコピー (shallow copy) されます。
- **body** (*str or unicode*) – リクエスト・ボディ。 `unicode` が渡されると、渡された `encoding` (デフォルトは `utf-8`) を使用して `str` にエンコードされます。 `body` が与えられない場合、空の文字列が保存されます。この引数のタイプに関係なく、保存される最終的な値は `str` (決して `unicode` や `None` ではありません)。
- **headers** (*dict*) – このリクエストのヘッダー。 辞書値は、文字列 (単一値のヘッダーの場合) またはリスト (複数値のヘッダーの場合) です。 値として `None` が渡された場合、HTTP ヘッダーはまったく送信されません。
- **cookies** (*dict or list*) – リクエスト・クッキー。これらは 2 つの形式で送信できます。

  1. 辞書の使用:

     ```
     request_with_cookies = Request(url="http://www.example.com",
                                    cookies={'currency': 'USD', 'country':
     ↪'UY'})
     ```

  2. 辞書のリストの使用:

```
request_with_cookies = Request(url="http://www.example.com",
                               cookies=[{'name': 'currency',
                                         'value': 'USD',
                                         'domain': 'example.com',
                                         'path': '/currency'}])
```

後者の形式では、クッキーの domain および path 属性をカスタマイズできます。これは、クッキーが後のリクエストのために保存される場合にのみ役立ちます。 一部のサイトが (レスポンスで) クッキーを返すと、それらはそのドメインのクッキーに保存され、今後のリクエストで再度送信されます。これは通常の Web ブラウザの一般的な動作です。けれども、何らかの理由で既存のクッキーとのマージを避けたい場合は、 *Request.meta* で dont_merge_cookies キーを True に設定することで、Scrapy にそうするよう指示できます。

クッキーをマージしないリクエストの例:

```
request_with_cookies = Request(url="http://www.example.com",
                               cookies={'currency': 'USD', 'country':
↪'UY'},
                               meta={'dont_merge_cookies': True})
```

詳細については、 *CookiesMiddleware* を参照してください。

- **encoding** (*string*) – このリクエストのエンコーディング (デフォルトは 'utf-8')。このエンコードは、URL をパーセントエンコードし、本文を str に変換するために使用されます (unicode として指定された場合)。
- **priority** (*int*) – このリクエストの優先度 (デフォルトは 0)。スケジューラーは優先度を使用して、リクエストの処理に使用される順序を定義します。より高い優先度値を持つリクエストは、より早く実行されます。比較的低い優先度を示すために、負の値が許可されています。
- **dont_filter** (*boolean*) – このリクエストは、スケジューラによってフィルタリングされるべきではないことを示します。 これは、重複フィルターを無視するために、同じリクエストを複数回実行する場合に使用されます。注意して使用しないと、クロールループに陥ります。デフォルトは False です。
- **errback** (*callable*) – リクエストの処理中に例外が発生した場合に呼び出される関数。これには、404 HTTP エラーなどで失敗したページが含まれます。最初のパラメーターとして Twisted Failure インスタンスを受け取ります。 詳細については、以下の **リクエスト処理で例外をキャッチするためにエラーバック *(errback)* を使用する** を参照してください。
- **flags** (*list*) – リクエストに送信されたフラグは、ロギングまたは同様の目的に使用できます。
- **cb_kwargs** (*dict*) – キーワード引数としてリクエストのコールバックに渡される任意の

データを含む辞書。

url
このリクエストの URL を含む文字列。 この属性にはエスケープされた URL が含まれているため、コンストラクターで渡される URL とは異なる場合があることに注意してください。

この属性は読み取り専用です。リクエストの URL を変更するには、 *replace()* を使用します。

method
リクエスト内の HTTP メソッドを表す文字列。 これは大文字であることが保証されています。 例： `"GET"` 、 `"POST"` 、 `"PUT"` など

headers
リクエスト・ヘッダーを含む辞書のようなオブジェクト。

body
リクエスト・ボディを含む文字列 (str)。

この属性は読み取り専用です。リクエストの本文を変更するには、 *replace()* を使用します。

meta
このリクエストの任意のメタデータを含む辞書。 この辞書は、新しいリクエストに対して空であり、通常、さまざまな Scrapy コンポーネント (拡張機能、ミドルウェアなど) によって設定されます。したがって、この辞書に含まれるデータは、有効にした拡張機能によって異なります。

Scrapy によって認識される特殊なメタ・キーのリストについては、 *Request.meta* **特殊キー** を参照してください。

この辞書は `copy()` または `replace()` メソッドを使用してリクエストが複製されたときに浅いコピーされ (shallow copied)、スパイダーで `response.meta` 属性からアクセスすることもできます。

cb_kwargs
このリクエストの任意のメタデータを含む辞書。その内容は、キーワード引数としてリクエストのコールバックに渡されます。新しいリクエストの場合は空です。つまり、デフォルトではコールバックは引数として *Response* オブジェクトのみを取得します。

この辞書は、 `copy()` または `replace()` メソッドを使用してリクエストが複製されたときに浅いコピーされ (shallow copied)、スパイダーで `response.cb_kwargs` 属性からアクセスすることもできます。

copy()
このリクエストのコピーである新しいリクエストを返します。 **追加のデータをコールバック関数に渡す** も参照してください。

replace([*url*, *method*, *headers*, *body*, *cookies*, *meta*, *flags*, *encoding*, *priority*, *dont_filter*, *callback*, *errback*, *cb_kwargs*])
指定されたキーワード引数によって新しい値が指定されたメンバーを除き、同じメンバーを持つリクエ

スト・オブジェクトを返します。 *Request.cb_kwargs* および *Request.meta* 属性は (新しい値が引数として与えられない限り) デフォルトでは浅くコピー (shallow copy) されます。 **追加のデータをコールバック関数に渡す** も参照してください。

追加のデータをコールバック関数に渡す

リクエストのコールバックは、そのリクエストのレスポンスがダウンロードされるときに呼び出される関数です。コールバック関数は、ダウンロードされた *Response* オブジェクトを最初の引数として呼び出されます。

例:

```
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.logger.info("Visited %s", response.url)
```

場合によっては、後で 2 番目のコールバックで引数を受け取ることができるように、これらのコールバック関数に引数を渡すことに興味があるかもしれません。 次の例は、 *Request.cb_kwargs* 属性を使用してこれを実現する方法を示しています:

```
def parse(self, response):
    request = scrapy.Request('http://www.example.com/index.html',
                             callback=self.parse_page2,
                             cb_kwargs=dict(main_url=response.url))
    request.cb_kwargs['foo'] = 'bar'  # add more arguments for the callback
    yield request

def parse_page2(self, response, main_url, foo):
    yield dict(
        main_url=main_url,
        other_url=response.url,
        foo=foo,
    )
```

ご用心: *Request.cb_kwargs* はバージョン 1.7 で導入されました。 それ以前は、コールバックに情報を渡すために *Request.meta* を使用することが推奨されていました。1.7 以降では、 *Request.cb_kwargs* がユーザー情報を処理するための好ましい方法となり、 *Request.meta* は、ミドルウェアや拡張機能などのコンポーネントとの通信のために残されています。

リクエスト処理で例外をキャッチするためにエラーバック (errback) を使用する

リクエストのエラーバック (errback) は、処理中に例外が発生したときに呼び出される関数です。

最初のパラメーターとして Twisted Failure インスタンスを受け取り、接続確立タイムアウト、DNS エラーなどを追跡するために使用できます。

すべてのエラーをログに記録し、必要に応じて特定のエラーをキャッチするスパイダーの例を次に示します:

```
import scrapy

from scrapy.spidermiddlewares.httperror import HttpError
from twisted.internet.error import DNSLookupError
from twisted.internet.error import TimeoutError, TCPTimedOutError

class ErrbackSpider(scrapy.Spider):
    name = "errback_example"
    start_urls = [
        "http://www.httpbin.org/",              # HTTP 200 expected
        "http://www.httpbin.org/status/404",    # Not found error
        "http://www.httpbin.org/status/500",    # server issue
        "http://www.httpbin.org:12345/",        # non-responding host, timeout expected
        "http://www.httphttpbinbin.org/",       # DNS error expected
    ]

    def start_requests(self):
        for u in self.start_urls:
            yield scrapy.Request(u, callback=self.parse_httpbin,
                                 errback=self.errback_httpbin,
                                 dont_filter=True)

    def parse_httpbin(self, response):
        self.logger.info('Got successful response from {}'.format(response.url))
        # do something useful here...

    def errback_httpbin(self, failure):
        # log all failures
        self.logger.error(repr(failure))

        # in case you want to do something special for some errors,
        # you may need the failure's type:

        if failure.check(HttpError):
            # these exceptions come from HttpError spider middleware
            # you can get the non-200 response
            response = failure.value.response
            self.logger.error('HttpError on %s', response.url)

        elif failure.check(DNSLookupError):
```

(次のページに続く)

(前のページからの続き)

```
            # this is the original request
            request = failure.request
            self.logger.error('DNSLookupError on %s', request.url)

        elif failure.check(TimeoutError, TCPTimedOutError):
            request = failure.request
            self.logger.error('TimeoutError on %s', request.url)
```

3.9.2 Request.meta 特殊キー

Request.meta 属性には任意のデータを含めることができますが、Scrapy とその組み込み拡張機能によって認識される特殊なキーがあります。

以下がその特殊キーです:

- *dont_redirect*
- *dont_retry*
- *handle_httpstatus_list*
- *handle_httpstatus_all*
- *dont_merge_cookies*
- *cookiejar*
- *dont_cache*
- *redirect_reasons*
- *redirect_urls*
- *bindaddress*
- *dont_obey_robotstxt*
- *download_timeout*
- *download_maxsize*
- *download_latency*
- *download_fail_on_dataloss*
- *proxy*
- ftp_user (詳細は *FTP_USER* 参照)

- ftp_password (詳細は *FTP_PASSWORD* 参照)
- *referrer_policy*
- *max_retry_times*

bindaddress

リクエストの実行に使用する発信 IP アドレスの IP

download_timeout

ダウンローダーがタイムアウトするまで待機する時間 (秒)。 *DOWNLOAD_TIMEOUT* も参照してください。

download_latency

リクエストが開始されてから、つまりネットワークを介して送信された HTTP メッセージから、レスポンスの取得に費やされた時間。 このメタ・キーは、レスポンスがダウンロードされた場合にのみ使用可能になります。他のほとんどのメタ・キーは Scrapy の動作を制御するために使用されますが、これは読み取り専用であると想定されています。

download_fail_on_dataloss

壊れたレスポンスで失敗するかどうか。 *DOWNLOAD_FAIL_ON_DATALOSS* を参照してください。

max_retry_times

メタ・キーを使用して、リクエストごとに再試行回数を設定します。初期化されると、 *max_retry_times* メタ・キーは *RETRY_TIMES* 設定よりも優先されます。

3.9.3 Request のサブクラス

以下は組み込みの *Request* のサブクラスのリストです。また、サブクラス化して独自のカスタム機能を実装することもできます。

FormRequest オブジェクト

FormRequest クラスは、ベースの *Request* を HTML フォームを処理する機能に関して拡張します。 lxml.html forms を使用して、フォームフィールドに *Response* オブジェクトからのフォームデータを事前入力します。

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

FormRequest クラスはコンストラクターに新しい引数を追加します。残りの引数は *Request* クラスと同じであり、ここでは説明しません。

パラメータ **formdata** (*dict or iterable of tuples*) – これは、URL エンコードされてリクエストの本文に割り当てられる HTML フォームデータを含む辞書 (または (キー, 値) タプルの反復可能要素) です。

FormRequest オブジェクトは、標準の *Request* メソッドに加えて、次のクラスメソッドをサポートします:

classmethod from_response(*response*[, *formname=None*, *formid=None*, *formnumber=0*, *formdata=None*, *formxpath=None*, *formcss=None*, *clickdata=None*, *dont_click=False*, ...])

指定のレスポンスに含まれる HTML `<form>` 要素で見つかった値が事前に入力されたフォームフィールド値を持つ新しい *FormRequest* オブジェクトを返します。例については、 *FormRequest.from_response()* **を使用してユーザーログインをシミュレートする** を参照してください。

ポリシーは、デフォルトでは、 `<input type="submit">` のようにクリック可能に見えるフォームコントロールのクリックを自動的にシミュレートすることです。 これは非常に便利で、多くの場合望ましい動作ですが、時にはデバッグが困難な問題を引き起こす可能性があります。 たとえば、javascript を使用して、入力 and/or 送信されたフォームを操作する場合、デフォルトの *from_response()* 動作は最適ではない場合があります。この動作を無効にするには、 `dont_click` 引数を `True` に設定します。 また、(無効にするのではなく) クリックしたコントロールを変更したい場合は、 `clickdata` 引数を使用することもできます。

ご用心: オプション値に先頭または末尾の空白がある select 要素でこのメソッドを使用すると、lxml 3.8 で修正されるべき lxml のバグ (bug in lxml) のために機能しません。

パラメータ

- **response** (*Response* object) – フォームフィールドに事前入力するために使用される HTML フォームを含むレスポンス
- **formname** (*string*) – 指定した場合、name 属性をこの値に設定したフォームが使用されます。
- **formid** (*string*) – 指定した場合、この値に設定された id 属性を持つフォームが使用されます。
- **formxpath** (*string*) – 指定すると、xpath に一致する最初のフォームが使用されます。
- **formcss** (*string*) – 指定した場合、css セレクターに一致する最初のフォームが使用されます。

- **formnumber** (*integer*) – レスポンスに複数のフォームが含まれる場合に使用するフォームの数。 最初のもの (およびデフォルト) は 0 です。
- **formdata** (*dict*) – フォームデータでオーバーライドするフィールド。レスポンス <form> 要素にフィールドが既に存在する場合、その値はこのパラメーターで渡された値によってオーバーライドされます。このパラメーターに渡された値が None の場合、フィールドはレスポンス <form> 要素に存在していても、リクエストに含まれません。
- **clickdata** (*dict*) – クリックされたコントロールを検索する属性。 指定されていない場合、最初のクリック可能な要素のクリックをシミュレートしてフォームデータが送信されます。 html 属性に加えて、コントロールは nr 属性を介して、フォーム内の他の送信可能な入力に対するゼロベースのインデックスによって識別できます。
- **dont_click** (*boolean*) – True の場合、要素をクリックせずにフォームデータが送信されます。

このクラスメソッドの他のパラメーターは、 *FormRequest* コンストラクターに直接渡されます。

バージョン 0.10.3 で追加: formname パラメータ。

バージョン 0.17 で追加: formxpath パラメータ。

バージョン 1.1.0 で追加: formcss パラメータ。

バージョン 1.1.0 で追加: formid パラメータ。

Request 使用例

HTTP POST 経由でデータを送信するために FormRequest を使う

スパイダーで HTML フォーム POST をシミュレートし、いくつかのキー値フィールドを送信する場合、以下のように (スパイダーから) *FormRequest* オブジェクトを返すことができます:

```
return [FormRequest(url="http://www.example.com/post/action",
                    formdata={'name': 'John Doe', 'age': '27'},
                    callback=self.after_post)]
```

FormRequest.from_response() を使用してユーザーログインをシミュレートする

Web サイトでは通常、セッション関連データや認証トークン (ログインページ用) などの <input type="hidden"> 要素を介して事前入力されたフォームフィールドを提供します。 スクレイピングするとき、これらのフィールドは自動的に事前入力され、ユーザー名やパスワードなどのいくつかのフィールドのみがオーバーライド必須です。この作業には *FormRequest.from_response()* メソッドを使用できます。以下はこれを使用するスパイダーの例です:

```
import scrapy

def authentication_failed(response):
    # TODO: Check the contents of the response and return True if it failed
    # or False if it succeeded.
    pass

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):
        if authentication_failed(response):
            self.logger.error("Login failed")
            return

        # continue scraping with authenticated session...
```

JsonRequest

JsonRequest クラスは、ベースの *Request* クラスに JSON リクエストを処理する機能をくわえます。

class scrapy.http.**JsonRequest**(*url*[, ... *data*, *dumps_kwargs*])

JsonRequest クラスは、コンストラクターに 2 つの新しい引数を追加します。残りの引数は *Request* クラスと同じであり、ここでは説明しません。

JsonRequest を使用すると、Content-Type ヘッダーを application/json にセットし、そして、Accept ヘッダーを application/json, text/javascript, */*; q=0.01 にセットします。

パラメータ

- **data** (*JSON serializable object*) – JSON エンコードして本文に割り当てる必要がある JSON シリアル化可能オブジェクトです。 *Request.body* 引数が指定されている場合、このパラメーターは無視されます。 *Request.body* 引数が提供されておらず、データ引数が提供されている場合、 *Request.method* は 'POST' に自動的に設定されます。
- **dumps_kwargs** (*dict*) – データを JSON 形式にシリアル化するために使用される、基礎となる json.dumps メソッドに渡されるパラメーター。

JsonRequest 使用例

JSON ペイロードを含む JSON POST リクエストを送信する:

```
data = {
    'name1': 'value1',
    'name2': 'value2',
}
yield JsonRequest(url='http://www.example.com/post/action', data=data)
```

3.9.4 Response オブジェクト

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

Response オブジェクトは HTTP レスポンスを表し、通常は (ダウンローダーによって) ダウンロードされ、処理のためにスパイダーに送られます。

パラメータ

- **url** (*string*) – このレスポンスの URL
- **status** (*integer*) – レスポンスの HTTP ステータス。デフォルトは 200 です。
- **headers** (*dict*) – このレスポンスのヘッダー。 辞書値は、文字列 (単一値のヘッダーの場合) またはリスト (複数値のヘッダーの場合) です。
- **body** (*bytes*) – レスポンス・ボディ。 デコードされたテキストに str(Python2 ではユニコード) としてアクセスするには、エンコード対応 (encoding-aware) である、 *TextResponse* のような *Response* **のサブクラス** の response.text を使用できます。
- **flags** (*list*) – *Response.flags* 属性の初期値を含むリストです。 指定すると、リストは浅くコピー (shallow copy) されます。
- **request** (*Request* object) – *Response.request* 属性の初期値。これは、このレスポンスを生成した *Request* を表します。

url

レスポンスの URL を含む文字列。

この属性は読み取り専用です。レスポンスの URL を変更するには、 *replace()* を使用します。

status

レスポンスの HTTP ステータスを表す整数。例： 200 、 404

headers

レスポンス・ヘッダーを含む辞書のようなオブジェクト。値にアクセスするには、 get() を使用し

て指定した名前の最初のヘッダー値を返すか、 `getlist()` を使用して指定した名前のすべてのヘッダー値を返します。たとえば、以下のの呼び出しはヘッダーのすべてのクッキーを提供します:

```
response.headers.getlist('Set-Cookie')
```

body

この Response のボディ。Response.body は常にバイト・オブジェクトであることに注意してください。Unicode バージョンが必要な場合は、 *TextResponse.text* を使用します (*TextResponse* と、そのサブクラスでのみ使用可能)。

この属性は読み取り専用です。 レスポンスのボディを変更するには、 *replace()* を使用します。

request

このレスポンスを生成した *Request* オブジェクト。この属性は、レスポンスとリクエストが、すべての **ダウンローダー・ミドルウェア** を通過した後、Scrapy エンジンで割り当てられます。 特に、これは以下を意味します:

- HTTP リダイレクトにより、元のリクエスト (リダイレクト前の URL へ) がリダイレクトされたレスポンス (リダイレクト後の最終 URL) に割り当てられます。
- Response.request.url は必ずしも Response.url と同じではありません
- この属性は、スパイダー・コード、および **スパイダー・ミドルウェア** でのみ使用できます。ただし、(他の方法でリクエストを使用できる場合の) ダウンローダー・ミドルウェアと *response_downloaded* シグナルのハンドラーには含まれません。

meta

Response.request オブジェクトの *Request.meta* 属性 (つまり `self.request.meta`) へのショートカット。

Response.request 属性とは異なり、 *Response.meta* 属性はリダイレクトと再試行に沿って伝播されるため、元の *Request.meta* がスパイダーから送信されます。

参考:

Request.meta 属性

flags

このレスポンスのフラグを含むリスト。フラグは、レスポンスのタグ付けに使用されるラベルです。例: `'cached'` 、`'redirected'` など。これらは、エンジンがログ記録に使用する Response (*__str__* メソッド) の文字列表現に表示されます。

copy()

このレスポンスのコピーである新しいレスポンスを返します。

replace([*url*, *status*, *headers*, *body*, *request*, *flags*, *cls*])

指定されたキーワード引数によって新しい値が指定されたメンバーを除き、同じメンバーを持つレスポ

ンスオブジェクトを返します。属性 *Response.meta* はデフォルトでコピーされます。

urljoin(*url*)
指定の *url* (たぶん相対 URL) と レスポンスの *url* (Response.url) を組み合わせて、絶対 URL を構築します。

これは urlparse.urljoin のラッパーであり、以下の呼び出しを行うための単なるエイリアスです:

```
urlparse.urljoin(response.url, url)
```

3.9.5 Response のサブクラス

使用可能な組み込み Response のサブクラスのリストは以下のとおりです。 Response クラスをサブクラス化して、独自の機能を実装することもできます。

TextResponse オブジェクト

class scrapy.http.**TextResponse**(*url*[, *encoding*[, ...]])
TextResponse オブジェクトは、エンコード機能を、ベースの *Response* クラスに追加します。これは、画像、音声、メディアファイルなどのバイナリデータにのみ使用することを目的としています。

TextResponse オブジェクトは、ベースの *Response* オブジェクトに加えて、新しいコンストラクター引数をサポートします。残りの機能は *Response* クラスと同じであり、ここでは説明しません。

パラメータ **encoding** (*string*) – このレスポンスに使用するエンコーディングを含む文字列です。ユニコード・ボディで *TextResponse* オブジェクトを作成する場合、このエンコードを使用してエンコードされます (body 属性は常に文字列であることに注意してください)。 encoding が None (デフォルト値) の場合、代わりにレスポンス・ヘッダーとボディからエンコードを検索します。

TextResponse オブジェクトは、標準の *Response* に加えて、次の属性をサポートします:

text
ユニコードとしてのレスポンス・ボディ

response.body.decode(response.encoding) と同じですが、最初の呼び出し後に結果がキャッシュされるため、余分なオーバーヘッドなしで response.text に複数回アクセスできます。

注釈: unicode(response.body) はレスポンス・ボディをユニコードに変換する正しい方法ではありません。レスポンス・エンコーディングの代わりにシステムのデフォルト・エンコーディング (通常は ascii) を使用することになります。

---

encoding
このレスポンスのエンコードを含む文字列。 エンコードは、次のメカニズムを順番に試して解決されます:

1. コンストラクタ `encoding` 引数に渡されたエンコーディング
2. Content-Type HTTP ヘッダーで宣言されたエンコーディング。 このエンコードが有効でない (つまり不明の) 場合、無視され、次の解決メカニズムが試行されます。
3. レスポンス・ボディで宣言されたエンコーディング。 TextResponse クラスは、このための特別な機能を提供しません。 ただし、 *HtmlResponse* と *XmlResponse* クラスはサポートします。
4. レスポンス・ボディを見て推測するエンコーディング。 これはより壊れやすい方法ですが、最後に試す方法でもあります。

selector
レスポンスをターゲットとして使用する `Selector` インスタンス。セレクターは最初のアクセスで遅延的 (lazily) にインスタンス化されます。

TextResponse オブジェクトは標準の *Response* に加えて以下のメソッドをサポートします:

xpath(*query*)
`TextResponse.selector.xpath(query)` へのショートカット:

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

css(*query*)
`TextResponse.selector.css(query)` へのショートカット:

```
response.css('p')
```

body_as_unicode()
text と同じですが、メソッドとして使用できます。 このメソッドは、後方互換性のために残されています。 `response.text` を優先してください。

HtmlResponse オブジェクト

class scrapy.http.**HtmlResponse**(*url*[, ...])
HtmlResponse クラスは *TextResponse* のサブクラスで、HTML の meta http-equiv 属性を調べることでエンコーディングの自動検出サポートを追加します。 *TextResponse.encoding* 参照。

XmlResponse オブジェクト

class scrapy.http.**XmlResponse**(*url*[, ...])
XmlResponse クラスは *TextResponse* のサブクラスで、XML 宣言行を調べることでエンコーディン

グの自動検出サポートを追加します。 *TextResponse.encoding* 参照。

3.10 リンク抽出器 (extractor)

リンク抽出器 (link extractor) は、最終的に追跡される Web ページ (*scrapy.http.Response* オブジェクト) からリンクを抽出することを唯一の目的とするオブジェクトです。

Scrapy には `scrapy.linkextractors.LinkExtractor` がありますが、シンプルなインターフェースを実装することで、ニーズに合わせて独自のカスタム・リンク抽出器を作成できます。

すべてのリンク抽出器が持つ唯一のパブリック・メソッドは `extract_links` で、これは *Response* オブジェクトを受け取り、 `scrapy.link.Link` オブジェクトのリストを返します。リンク抽出器は一度インスタンス化され、それらの `extract_links` メソッドが異なるレスポンスで数回呼び出されて、追跡するリンクを抽出します。

リンク抽出器は、(Scrapy に用意されている) *CrawlSpider* クラスで一連のルールを介して使用されますが、*CrawlSpider* サブクラスを作成しなくても、スパイダーでも使用できます。

3.10.1 組み込みリンク抽出器リファレンス

Scrapy に同梱されているリンク抽出器クラスは、 *scrapy.linkextractors* モジュールで提供されます。

デフォルトのリンク抽出器は `LinkExtractor` で、これは *LxmlLinkExtractor* と同じです:

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

以前の Scrapy バージョンには他のリンク抽出器クラスがありましたが、現在は廃止されています。

LxmlLinkExtractor

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

LxmlLinkExtractor は、便利なフィルタリングオプションを備えた、おすすめのリンク抽出器です。lxml の

堅牢な HTMLParser を使用して実装されています。

パラメータ

- **allow** (*a regular expression (or list of)*) – (絶対)URL が抽出されるために一致する必要がある単一の正規表現 (または正規表現のリスト)。指定しない場合 (または空の場合) は、すべてのリンクに一致します。
- **deny** (*a regular expression (or list of)*) – (絶対)URL が除外される (抽出されない) ために一致する必要がある単一の正規表現 (または正規表現のリスト)。 `allow` パラメータよりも優先されます。指定されていない (または空の) 場合、リンクは除外されません。
- **allow_domains** (*str or list*) – リンクを抽出するために考慮されるドメインを含む単一の値または文字列のリスト
- **deny_domains** (*str or list*) – リンクを抽出するために **考慮されない** ドメインを含む単一の値または文字列のリスト
- **deny_extensions** (*list*) – リンクを抽出するときに無視される拡張子を含む単一の値または文字列のリスト。 指定しない場合、 scrapy.linkextractors パッケージで定義された `IGNORED_EXTENSIONS` リストがデフォルトになります。
- **restrict_xpaths** (*str or list*) – これは、リンクを抽出するレスポンス内の領域を定義する XPath(または XPath のリスト) です。指定すると、それらの XPath によって選択されたテキストのみがリンクをスキャンされます。以下の例を参照してください。
- **restrict_css** (*str or list*) – リンクを抽出するレスポンス内の領域を定義する CSS セレクター (またはセレクターのリスト)。 `restrict_xpaths` と同じ動作をします。
- **restrict_text** (*a regular expression (or list of)*) – リンクのテキストが抽出されるために一致する必要がある単一の正規表現 (または正規表現のリスト)。指定しない場合 (または空の場合) は、すべてのリンクに一致します。正規表現のリストが指定されている場合、リンクが少なくとも 1 つと一致するとリンクが抽出されます。
- **tags** (*str or list*) – リンクを抽出するときに考慮するタグまたはタグのリスト。 デフォルトは `('a', 'area')` です。
- **attrs** (*list*) – 抽出するリンクを探すときに考慮する必要がある属性または属性のリスト (`tags` パラメータで指定されたタグのみ)。デフォルトは `('href',)`
- **canonicalize** (*boolean*) – 抽出された各 URL を正規化します (w3lib.url.canonicalize_url を使用)。 デフォルトは `False` です。canonicalize_url は重複チェックを目的としていることに注意してください。サーバー側で表示される URL を変更できるため、正規化された URL と生の URL を使用したリクエストのレスポンスが異なる場合があります。LinkExtractor を使用してリンクをたどっている場合、デフォルトの `canonicalize=False` のままにしておいた方がより堅牢です。

- **unique** (*boolean*) – 抽出されたリンクに重複フィルタリングを適用するかどうか。
- **process_value** (*callable*) – タグから抽出された各値とスキャンされた属性を受け取り、値を変更して新しい値を返すか、リンクを完全に無視するために None を返すことができる関数。 指定しない場合、 process_value のデフォルトは lambda x: x になります。

  たとえば、このコードからリンクを抽出するには:

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

  あなたは process_value で次の関数を使用できます:

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

- **strip** (*boolean*) – 抽出された属性から空白を削除するかどうか。HTML5 標準によれば、先頭と末尾の空白は、<a> や <area> の href 属性や他の多くの要素、<img> や <iframe> の src 属性、などから削除する必要があります。そのため LinkExtractor はデフォルトでスペース文字を削除します。オフにするには strip=False を設定します (たとえば、先頭または末尾の空白を許可する要素または属性から URL を抽出する場合)。

3.11 設定

Scrapy 設定を使用すると、コア、拡張機能、パイプライン、スパイダー自体を含むすべての Scrapy コンポーネントの動作をカスタマイズできます。

設定のインフラストラクチャは、コードが構成値を取得するために使用できるキーと値のマッピングのグローバル名前空間を提供します。設定は、以下で説明するさまざまなメカニズムを使用して設定できます。

設定は、(多くの場合) 現在アクティブな Scrapy プロジェクトを選択するためのメカニズムでもあります。

利用可能な組み込み設定のリストについては、 **組み込みの設定リファレンス** を参照してください。

3.11.1 設定の指定

あなたが Scrapy を使用するときは、あなたは使用している設定を伝える必要があります。これを行うには、環境変数 SCRAPY_SETTINGS_MODULE を使用します。

SCRAPY_SETTINGS_MODULE の値は、Python パス構文である必要があります。例えば myproject.settings です。設定モジュールは Python のインポート検索パス (import search path) にある必要があることに注意してください。

3.11.2 設定の入力

設定は、それぞれ異なる優先順位を持つさまざまなメカニズムを使用して入力できます。 優先順位の降順でそれらのリストを示します:

1. コマンド・ライン・オプション (最優先)
2. スパイダーごとの設定
3. プロジェクト設定モジュール
4. コマンドごとのデフォルト設定
5. デフォルトのグローバル設定 (最も優先度が低い)

これらの設定ソースの入力は内部的に処理されますが、API 呼び出しを使用して手動で処理することができます。参考として *API の設定* トピックを参照してください。

これらのメカニズムについては、以下で詳しく説明します。

1. コマンド・ライン・オプション

コマンドラインで提供される引数は、他のオプションより優先され、最も優先される引数です。 ‘ -s‘‘ (または `--set`) コマンドラインオプションを使用して、1 つ (または複数) の設定を明示的にオーバーライドできます。

例:

```
scrapy crawl myspider -s LOG_FILE=scrapy.log
```

2. スパイダーごとの設定

スパイダー (**スパイダー** 参照) は、プロジェクト設定を優先して上書きする独自の設定を定義できます。 そのためには *custom_settings* 属性を設定します:

```
class MySpider(scrapy.Spider):
    name = 'myspider'

    custom_settings = {
        'SOME_SETTING': 'some value',
    }
```

3. プロジェクト設定モジュール

プロジェクト設定モジュールは、Scrapy プロジェクトの標準構成ファイルであり、ほとんどのカスタム設定がそこに入力されます。 標準の Scrapy プロジェクトの場合、これは、プロジェクト用に作成された `settings.py` ファイルの設定を追加または変更することを意味します。

4. コマンドごとのデフォルト設定

各 *Scrapy* **ツール** コマンドには、グローバルなデフォルト設定を上書きする独自のデフォルト設定を含めることができます。これらのカスタム・コマンド設定は、コマンド・クラスの `default_settings` 属性で指定されます。

5. デフォルトのグローバル設定

グローバルなデフォルトは `scrapy.settings.default_settings` モジュールにあり、 **組み込みの設定リファレンス** で文書化されています。

3.11.3 設定にアクセスする方法

スパイダーでは、設定は `self.settings` から利用できます:

```
class MySpider(scrapy.Spider):
    name = 'myspider'
    start_urls = ['http://example.com']

    def parse(self, response):
        print("Existing settings: %s" % self.settings.attributes.keys())
```

注釈: `settings` 属性は、スパイダーが初期化された後にベース Spider クラスで設定されます。初期化の前に設定を使用する場合 (たとえば、スパイダーの `__init__()` メソッドで)、 *`from_crawler()`* メソッドをオーバーライドする必要があります。

設定には、拡張機能、ミドルウェア、アイテム・パイプラインの `from_crawler` メソッドに渡されるクローラーの *`scrapy.crawler.Crawler.settings`* 属性からアクセスできます:

```
class MyExtension(object):
    def __init__(self, log_is_enabled=False):
        if log_is_enabled:
            print("log is enabled!")

    @classmethod
    def from_crawler(cls, crawler):
        settings = crawler.settings
        return cls(settings.getbool('LOG_ENABLED'))
```

設定オブジェクトは辞書のように使用できます (例: `settings['LOG_ENABLED']`)。ただし、通常、タイプ・エラーを回避するために必要な形式で設定を抽出し、 `Settings` API で提供されるメソッドの 1 つを使用することをお勧めします。

3.11.4 名前を設定する理由

設定名には通常、構成するコンポーネントの接頭辞が付きます。 たとえば、架空の robots.txt 拡張子の適切な設定名は、 ROBOTSTXT_ENABLED 、 ROBOTSTXT_OBEY 、 ROBOTSTXT_CACHEDIR などになります。

3.11.5 組み込みの設定リファレンス

以下に、利用可能なすべてのスクレイピー設定のリストをアルファベット順に、デフォルト値と適用範囲とともに示します。

使用可能な場所では、スコープは、特定のコンポーネントに関連付けられている場合、設定が使用されている場所を示します。 その場合、そのコンポーネントのモジュールは通常、拡張機能、ミドルウェア、またはパイプラインが表示されます。 また、設定を有効にするには、コンポーネントを有効にする必要があります。

AWS_ACCESS_KEY_ID

デフォルト: None

S3 **フィードストレージバックエンド** など、 Amazon Web services へのアクセスを必要とするコードで使用される AWS アクセスキー。

AWS_SECRET_ACCESS_KEY

デフォルト: None

S3 **フィード・ストレージ・バックエンド** など、 Amazon Web services へのアクセスを必要とするコードで使用される AWS シークレット・キー

AWS_ENDPOINT_URL

デフォルト: None

Minio や s3.scality など、S3 のようなストレージに使用されるエンドポイント URL。 botocore ライブラリでのみサポートされています。

AWS_USE_SSL

デフォルト: None

S3 または S3 のようなストレージとの通信のために SSL 接続を無効にする場合は、このオプションを使用します。デフォルトでは、SSL が使用されます。 botocore ライブラリでのみサポートされています。

AWS_VERIFY

デフォルト: None

Scrapy と S3 または S3 のようなストレージ間の SSL 接続を検証 (verify) します。デフォルトでは、SSL 検証が行われます。 botocore ライブラリでのみサポートされています。

AWS_REGION_NAME

デフォルト: None

AWS クライアントに関連付けられているリージョンの名前。 botocore ライブラリでのみサポートされています。

BOT_NAME

デフォルト: 'scrapybot'

この Scrapy プロジェクトによって実装されるボットの名前 (プロジェクト名とも呼ばれます)。 これは、デフォルトで User-Agent を構築するために使用され、ログでも使用されます。

startproject コマンドでプロジェクトを作成すると、プロジェクト名が自動的に入力されます。

CONCURRENT_ITEMS

デフォルト: 100

アイテム・プロセッサ (**アイテム・パイプライン** とも呼ばれます) で並列処理する (レスポンスごとの) 同時アイテムの最大数。

CONCURRENT_REQUESTS

デフォルト: 16

Scrapy ダウンローダーが実行する並列 (すなわち同時) リクエストの最大数。

CONCURRENT_REQUESTS_PER_DOMAIN

デフォルト: 8

任意の単一ドメインに対して実行される並列 (すなわち同時) リクエストの最大数。

AutoThrottle **拡張機能** と、その *AUTOTHROTTLE_TARGET_CONCURRENCY* オプションを参照して下さい。

CONCURRENT_REQUESTS_PER_IP

デフォルト: 0

単一の IP に対して実行される並行 (すなわち同時) リクエストの最大数。 ゼロ以外の場合、*CONCURRENT_REQUESTS_PER_DOMAIN* 設定は無視され、代わりにこの設定が使用されます。 つまり、並列実行制限はドメインごとではなく IP ごとに適用されます。

この設定は、*DOWNLOAD_DELAY* と *AutoThrottle* **拡張機能** にも影響します。*CONCURRENT_REQUESTS_PER_IP* がゼロ以外の場合、ダウンロード遅延はドメインごとではなく IP ごとに適用されます。

DEFAULT_ITEM_CLASS

デフォルト: 'scrapy.item.Item'

Scrapy **シェル** 内のアイテムのインスタンス化に使用されるデフォルト・クラス。

DEFAULT_REQUEST_HEADERS

デフォルト:

```
{
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language': 'en',
}
```

Scrapy HTTP リクエストに使用されるデフォルトのヘッダー。それらは *DefaultHeadersMiddleware* に取り込まれます。

DEPTH_LIMIT

デフォルト: 0

スコープ: scrapy.spidermiddlewares.depth.DepthMiddleware

任意のサイトでクロールできる最大深度。 ゼロの場合、制限は課されません。

DEPTH_PRIORITY

デフォルト: 0

スコープ: scrapy.spidermiddlewares.depth.DepthMiddleware

深さに基づいて *Request* の priority を調整するために使用される整数。

リクエストの優先度は次のように調整されます:

```
request.priority = request.priority - ( depth * DEPTH_PRIORITY )
```

深さが増加すると、 DEPTH_PRIORITY の正の値はリクエストの優先度 (BFO) を下げ、負の値はリクエストの優先度 (DFO) を上げます。 *Scrapy* **は幅** *(breadth)* **優先または深さ** *(depth)* **優先でクロールしますか？** も参照してください。

注釈: この設定は、他の優先度設定 *REDIRECT_PRIORITY_ADJUST* と *RETRY_PRIORITY_ADJUST* と比較して、 **逆の方法で** 優先度を調整します。

DEPTH_STATS_VERBOSE

デフォルト: False

スコープ: scrapy.spidermiddlewares.depth.DepthMiddleware

詳細な統計情報を収集するかどうか。これが有効になっている場合、各深さのリクエスト数が統計に収集されます。

DNSCACHE_ENABLED

デフォルト: True

DNS イン・メモリ・キャッシュを有効にするかどうか。

DNSCACHE_SIZE

デフォルト: 10000

DNS イン・メモリ・キャッシュ・サイズ。

DNS_TIMEOUT

デフォルト: 60

DNS クエリの処理のタイムアウト (秒)。float 値がサポートされています。

DOWNLOADER

デフォルト: 'scrapy.core.downloader.Downloader'

クロールに使用するダウンローダー。

DOWNLOADER_HTTPCLIENTFACTORY

デフォルト: 'scrapy.core.downloader.webclient.ScrapyHTTPClientFactory'

(HTTP10DownloadHandler の場合、) HTTP/1.0 接続に使用する Twisted protocol.ClientFactory クラスを定義します。

注釈: 最近では HTTP/1.0 はめったに使用されないため、Twisted <11.1 を使用する場合、または HTTP/1.0 を使用して http(s) スキームの *DOWNLOAD_HANDLERS_BASE* をオーバーライドする場合、つまり 'scrapy.core.downloader.handlers.http.HTTP10DownloadHandler' を使用する場合を除き、この設定を無視しても安全です。

DOWNLOADER_CLIENTCONTEXTFACTORY

デフォルト: 'scrapy.core.downloader.contextfactory.ScrapyClientContextFactory'

使用する ContextFactory へのクラスパスを表します。

ここで、ContextFactory は SSL/TLS コンテキストの Twisted 用語であり、使用する TLS/SSL プロトコルのバージョン、証明書の検証 (verification) を行うか、クライアント側の認証を有効にするかなどを定義します。

注釈: Scrapy デフォルト・コンテキスト・ファクトリは **リモート・サーバー証明書の検証を実行しません** 。これは通常、Web スクレイピングに適しています。

リモート・サーバー証明書の検証を有効にする必要がある場合、Scrapy には設定可能な別のコンテキスト・ファクトリ・クラス 'scrapy.core.downloader.contextfactory.BrowserLikeContextFactory' があり、プラットフォームの証明書を使用してリモート・エンドポイントを検証します。 **これは、Twisted>=14.0 を使用する場合にのみ利用可能です** 。

カスタム ContextFactory を使用する場合、その __init__ メソッドが method パラメーター (これは OpenSSL.SSL メソッド・マッピング *DOWNLOADER_CLIENT_TLS_METHOD* です) と tls_verbose_logging'' **パラメーター** (``bool) と tls_ciphers パラメーター (*DOWNLOADER_CLIENT_TLS_CIPHERS* 参照) を受け入れる事を確認して下さい。

DOWNLOADER_CLIENT_TLS_CIPHERS

デフォルト: 'DEFAULT'

この設定を使用して、デフォルトの HTTP/1.1 ダウンローダーが使用する TLS/SSL 暗号 cipher) をカスタマイズします。

設定には OpenSSL 暗号リスト形式 (OpenSSL cipher list format) の文字列が含まれている必要があります。これらの暗号はクライアント暗号として使用されます。 特定の HTTPS Web サイトにアクセスするには、この設定の変更が必要になる場合があります。たとえば、弱い DH パラメーターを持つ Web サイトに `'DEFAULT:!DH'` を使用しするか、または、Web サイトが要求しない場合は `DEFAULT` 含まれない特定の暗号を有効にする必要があります。

DOWNLOADER_CLIENT_TLS_METHOD

デフォルト: `'TLS'`

この設定を使用して、デフォルトの HTTP/1.1 ダウンローダーが使用する TLS/SSL メソッドをカスタマイズします。

この設定は、次の文字列値のいずれかでなければなりません:

- `'TLS'`: これは OpenSSL の `TLS_method()` (別名 `SSLv23_method()`) にマップします。これにより、プラットフォームでサポートされる最高のものからプロトコル・ネゴシエーションが可能になります。 **これがデフォルトかつ推奨です**
- `'TLSv1.0'`: この値は、HTTPS 接続が TLS バージョン 1.0 を使用するように強制します。Scrapy<1.1 の動作が必要な場合はこれを設定します
- `'TLSv1.1'`: TLS バージョン 1.1 の使用を強制します
- `'TLSv1.2'`: TLS バージョン 1.2 の使用を強制します
- `'SSLv3'`: SSL バージョン 3 の使用を強制します (**非推奨**)

注釈: 私達は PyOpenSSL>=0.13 かつ Twisted>=0.1 3 以上 (可能な場合は Twisted>=14.0) を使用することをお勧めします。

DOWNLOADER_CLIENT_TLS_VERBOSE_LOGGING

デフォルト: `False`

これを `True` に設定すると、HTTPS 接続を確立した後、TLS 接続パラメーターに関する DEBUG レベルのメッセージが有効になります。記録される情報の種類は、OpenSSL および pyOpenSSL のバージョンによって異なります。

この設定は、デフォルトの *DOWNLOADER_CLIENTCONTEXTFACTORY* にのみ使用されます。

DOWNLOADER_MIDDLEWARES

デフォルト:: `{}`

プロジェクトで有効になっているダウンローダー・ミドルウェアとその順序を含む辞書。詳細については、 **ダウンローダーミドルウェアをアクティブにする** を参照してください。

DOWNLOADER_MIDDLEWARES_BASE

デフォルト:

```
{
    'scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware': 100,
    'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware': 300,
    'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware': 350,
    'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware': 400,
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': 500,
    'scrapy.downloadermiddlewares.retry.RetryMiddleware': 550,
    'scrapy.downloadermiddlewares.ajaxcrawl.AjaxCrawlMiddleware': 560,
    'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware': 580,
    'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 590,
    'scrapy.downloadermiddlewares.redirect.RedirectMiddleware': 600,
    'scrapy.downloadermiddlewares.cookies.CookiesMiddleware': 700,
    'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware': 750,
    'scrapy.downloadermiddlewares.stats.DownloaderStats': 850,
    'scrapy.downloadermiddlewares.httpcache.HttpCacheMiddleware': 900,
}
```

Scrapy でデフォルトで有効になっているダウンローダー・ミドルウェアを含む辞書。 低次はエンジンに近く、高次はダウンローダーに近いです。プロジェクト内でこの設定を変更しないでください。代わりに *DOWNLOADER_MIDDLEWARES* を変更してください。詳細については、 **ダウンローダーミドルウェアをアクティブにする** を参照してください。

DOWNLOADER_STATS

デフォルト: `True`

ダウンローダー統計収集を有効にするかどうか。

DOWNLOAD_DELAY

デフォルト: `0`

ダウンローダーが同じ Web サイトから連続したページをダウンロードするまで待機する時間 (秒)。 これを使用してクロール速度を調整し、サーバーへの過度のヒットを回避できます。10 進数がサポートされています。 例:

```
DOWNLOAD_DELAY = 0.25    # 250 ms of delay
```

この設定は、 *RANDOMIZE_DOWNLOAD_DELAY* 設定 (デフォルトで有効) の影響も受けます。デフォルトで

は、Scrapy はリクエスト間で一定の時間を待機する訳ではなく、(0.5 * *DOWNLOAD_DELAY*) から (1.5 * *DOWNLOAD_DELAY*) の間のランダムな間隔を使用します。

CONCURRENT_REQUESTS_PER_IP がゼロ以外の場合、遅延はドメインごとではなく IP アドレスごとに適用されます。

あなたは download_delay スパイダー属性を設定することで、スパイダーごとにこの設定を変更することもできます。

DOWNLOAD_HANDLERS

デフォルト: {}

プロジェクトで有効にされたリクエスト・ダウンローダー・ハンドラーを含む辞書。 形式の例については、*DOWNLOAD_HANDLERS_BASE* を参照してください。

DOWNLOAD_HANDLERS_BASE

デフォルト:

```
{
    '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',
    'ftp': 'scrapy.core.downloader.handlers.ftp.FTPDownloadHandler',
}
```

Scrapy でデフォルトで有効になっているリクエスト・ダウンロード・ハンドラーを含む辞書。 プロジェクトのこの設定を変更するのではなく、代わりに *DOWNLOAD_HANDLERS* を変更してください。

DOWNLOAD_HANDLERS で URI スキームに None を割り当てることで、これらのダウンロード・ハンドラーを無効にできます。たとえば、組み込み FTP ハンドラーを (置換なしで) 無効にするには、これを settings.py に配置します:

```
DOWNLOAD_HANDLERS = {
    'ftp': None,
}
```

DOWNLOAD_TIMEOUT

デフォルト: 180

ダウンローダーがタイムアウトするまで待機する時間 (秒)。

注釈: このタイムアウトは、 download_timeout スパイダー属性を使用してスパイダーごとに設定でき、*download_timeout* Request.meta キーを使用してリクエストごとに設定できます。

DOWNLOAD_MAXSIZE

デフォルト: 1073741824 (1024MB)

ダウンローダーがダウンロードする最大レスポンス・サイズ (バイト単位)。

無効にしたい場合は 0 をセットします。

注釈: このサイズは、スパイダー属性 download_maxsize を使用してスパイダーごとに設定し、リクエストごとに *download_maxsize* Request.meta キーを使用して設定できます。

この機能には Twisted >= 11.1. が必要です。

DOWNLOAD_WARNSIZE

デフォルト: 33554432 (32MB)

ダウンローダーが警告し始めるレスポンス・サイズ (バイト単位)。

無効にしたい場合は 0 をセットします。

注釈: download_warnsize スパイダー属性を使用してスパイダーごとにこのサイズを設定し、download_warnsize Request.meta キーを使用してリクエストごとに設定できます。

この機能には Twisted >= 11.1. が必要です。

DOWNLOAD_FAIL_ON_DATALOSS

デフォルト: True

壊れた応答で失敗するかどうか、つまり、宣言された Content-Length がサーバーによって送信されたコンテンツと一致しないか、チャンクされた応答が適切に終了しませんでした。 True の場合、これらのレスポンスは ResponseFailed([_DataLoss]) エラーを発生させます。 False の場合、これらのレスポンスはパス・スルーされ、フラグ dataloss がレスポンスに追加されます。すなわち、response.flags の 'dataloss' は True です。

オプションで、これは *download_fail_on_dataloss* Request.meta キーを False に使用することで、リクエストごとに設定できます。

注釈: サーバーの設定ミスからネットワークエラー、データ破損まで、いくつかの状況下で、レスポンスの破損、またはデータ損失エラーが発生する場合があります。 部分的なコンテンツや不完全なコンテンツが含まれている可能性があることを考慮して、壊れたレスポンスを処理することが理にかなっているかどうかを判断するのはユーザーの責任です。本設定が True に設定されていて、かつ *RETRY_ENABLED* が True に設定されている場合、ResponseFailed([_DataLoss]) の失敗は通常どおり再試行されます。

DUPEFILTER_CLASS

デフォルト: 'scrapy.dupefilters.RFPDupeFilter'

重複したリクエストを検出およびフィルタリングするために使用されるクラス。

デフォルト (RFPDupeFilter) は scrapy.utils.request.request_fingerprint 関数を使用してリクエストのフィンガー・プリントに基づいてフィルターします。重複のチェック方法を変更するには、RFPDupeFilter をサブクラス化し、その request_fingerprint メソッドをオーバーライドします。このメソッドは、scrapy *Request* オブジェクトを受け入れ、そのフィンガー・プリント (文字列) を返す必要があります。

DUPEFILTER_CLASS を 'scrapy.dupefilters.BaseDupeFilter' に設定することで、重複したリクエストのフィルタリングを無効にできます。ただし、クロール・ループに入る可能性があるため、これには十分注意してください。通常、フィルタリングしない特定の *Request* で dont_filter パラメーターを True に設定することをお勧めします。

DUPEFILTER_DEBUG

デフォルト: False

デフォルトでは、 RFPDupeFilter は最初の重複リクエストのみを記録します。 *DUPEFILTER_DEBUG* を True に設定すると、重複するすべてのリクエストがログに記録されます。

EDITOR

デフォルト: vi (Unix システムの場合)、または IDLE エディター (Windows の場合)

edit コマンドでスパイダーを編集するために使用するエディター。さらに、 EDITOR 環境変数が設定されている場合、 *edit* コマンドはデフォルト設定よりもそれを優先します。

EXTENSIONS

デフォルト:: { }

プロジェクトで有効になっている拡張機能とその順序を含む辞書。

EXTENSIONS_BASE

デフォルト:

```
{
    'scrapy.extensions.corestats.CoreStats': 0,
    'scrapy.extensions.telnet.TelnetConsole': 0,
    'scrapy.extensions.memusage.MemoryUsage': 0,
    'scrapy.extensions.memdebug.MemoryDebugger': 0,
    'scrapy.extensions.closespider.CloseSpider': 0,
    'scrapy.extensions.feedexport.FeedExporter': 0,
    'scrapy.extensions.logstats.LogStats': 0,
    'scrapy.extensions.spiderstate.SpiderState': 0,
    'scrapy.extensions.throttle.AutoThrottle': 0,
}
```

Scrapy でデフォルトで使用可能な拡張機能とその順序を含む辞書。この設定には、すべての安定した組み込み拡張機能が含まれています。それらのいくつかは設定によって有効にする必要があることに留意してください。

詳細については、 **拡張機能ユーザーガイド** および **利用可能な拡張機能のリスト** を参照してください。

FEED_TEMPDIR

Feed Temp dir では、 *FTP* **フィード・ストレージ** と *Amazon S3* でアップロードする前に、クローラーの一時ファイルを保存するカスタム・フォルダーを設定できます。

FTP_PASSIVE_MODE

デフォルト: `True`

FTP 転送を開始するときにパッシブモードを使用するかどうか。

FTP_PASSWORD

デフォルト: `"guest"`

`Request` メタに `"ftp_password"` がない場合に FTP 接続に使用するパスワード。

注釈: RFC 1635 を意訳すると、匿名 FTP にはパスワード "guest" または自分の電子メールアドレスを使用するの

が一般的ですが、一部の FTP サーバーは、ユーザーの電子メールアドレスを明示的に要求し、 "guest" パスワードでのログインを許可しません。

FTP_USER

デフォルト: `"anonymous"`

`Request` メタに `"ftp_user"` がない場合に FTP 接続に使用するユーザー名。

ITEM_PIPELINES

デフォルト: `{}`

使用するアイテム・パイプラインとその順序を含む辞書。 順序の値は任意ですが、0〜1000 の範囲で定義するのが一般的です。 低いオーダーは高いオーダーの前に処理されます。

例:

```
ITEM_PIPELINES = {
    'mybot.pipelines.validate.ValidateMyItem': 300,
    'mybot.pipelines.validate.StoreMyItem': 800,
}
```

ITEM_PIPELINES_BASE

デフォルト: `{}`

Scrapy でデフォルトで有効になっているパイプラインを含む辞書。プロジェクトでこの設定を変更することは決してせず、代わりに *ITEM_PIPELINES* を変更してください。

LOG_ENABLED

デフォルト: `True`

ロギングを有効にするかどうか。

LOG_ENCODING

デフォルト: `'utf-8'`

ロギングに使用するエンコード。

LOG_FILE

デフォルト: None

ログ出力に使用するファイル名。 None の場合、標準エラーが使用されます。

LOG_FORMAT

デフォルト: '%(asctime)s [%(name)s] %(levelname)s: %(message)s'

ログ・メッセージをフォーマットするための文字列。 利用可能なプレース・ホルダーの全リストについては、Python logging documentation を参照してください。

LOG_DATEFORMAT

デフォルト: '%Y-%m-%d %H:%M:%S'

日付/時刻をフォーマットするための文字列、 *LOG_FORMAT* の %(asctime)s プレース・ホルダーの展開。 利用可能なディレクティブのリストについては、 Python datetime documentation を参照してください。

LOG_FORMATTER

デフォルト: scrapy.logformatter.LogFormatter

さまざまなアクションのログ・メッセージのフォーマット に使用するクラス。

LOG_LEVEL

デフォルト: 'DEBUG'

記録する最小レベル。 利用可能なレベルは、CRITICAL、ERROR、WARNING、INFO、DEBUG です。詳細については、**ロギング** *(logging)* を参照してください。

LOG_STDOUT

デフォルト: False

True の場合、処理のすべての標準出力 (およびエラー) がログにリダイレクトされます。 たとえば、print('hello') の場合、Scrapy ログに表示されます。

LOG_SHORT_NAMES

デフォルト: False

True の場合、ログにはルート・パスのみが含まれます。 False に設定されている場合、ログ出力を担当するコンポーネントが表示されます

LOGSTATS_INTERVAL

デフォルト: 60.0

LogStats による統計の各ログ出力間の間隔 (秒単位)。

MEMDEBUG_ENABLED

デフォルト: False

メモリデバッグを有効にするかどうか。

MEMDEBUG_NOTIFY

デフォルト: []

メモリ・デバッグが有効になっている場合、この設定が空でない場合、指定されたアドレスにメモリレポートが送信されます。そうでない場合、レポートはログに書き込まれます。

例:

```
MEMDEBUG_NOTIFY = ['user@example.com']
```

MEMUSAGE_ENABLED

デフォルト: True

スコープ: scrapy.extensions.memusage

メモリ使用量拡張機能を有効にするかどうか。 この拡張機能は、プロセスが使用するピークメモリを追跡します (統計に書き込みます)。また、オプションで、メモリ制限を超えたときに Scrapy プロセスをシャットダウンし (*MEMUSAGE_LIMIT_MB* を参照)、それが発生したときに電子メールで通知することができます (*MEMUSAGE_NOTIFY_MAIL* を参照)。

メモリ使用量の拡張機能 参照。

MEMUSAGE_LIMIT_MB

デフォルト: 0

スコープ: scrapy.extensions.memusage

(MEMUSAGE_ENABLED が True の場合、)Scrapy をシャットダウンする前に許可するメモリの最大量 (メガバイト単位)。ゼロの場合、チェックは実行されません。

メモリ使用量の拡張機能 参照。

MEMUSAGE_CHECK_INTERVAL_SECONDS

バージョン 1.1 で追加.

デフォルト: `60.0`

スコープ: `scrapy.extensions.memusage`

メモリ使用量拡張 は、現在のメモリ使用量と、 *MEMUSAGE_LIMIT_MB* および *MEMUSAGE_WARNING_MB* で設定された制限を一定の時間間隔でチェックします。

これにより、これらの間隔の長さが秒単位で設定されます。

メモリ使用量の拡張機能 参照。

MEMUSAGE_NOTIFY_MAIL

デフォルト: `False`

スコープ: `scrapy.extensions.memusage`

メモリ制限に達した場合に通知する電子メールのリスト。

例:

```
MEMUSAGE_NOTIFY_MAIL = ['user@example.com']
```

メモリ使用量の拡張機能 参照。

MEMUSAGE_WARNING_MB

デフォルト: `0`

スコープ: `scrapy.extensions.memusage`

通知する警告メールを送信する前に許可するメモリの最大量 (メガバイト単位)。ゼロの場合、警告は生成されません。

NEWSPIDER_MODULE

デフォルト: `''`

genspider コマンドを使用して新しいスパイダーを作成するモジュール。

例:

```
NEWSPIDER_MODULE = 'mybot.spiders_dev'
```

RANDOMIZE_DOWNLOAD_DELAY

デフォルト: `True`

有効にすると、Scrapy はランダムな時間 (0.5 * *DOWNLOAD_DELAY*) から (1.5 * *DOWNLOAD_DELAY*) の間待機し、同じ Web サイトからリクエストを取得します。

このランダム化により、リクエストを分析し、リクエスト間の時間の統計的に有意な類似性を探しているサイトによってクローラーが検出される (そしてその後ブロックされる) 機会が減少します。

ランダム化ポリシーは、 wget の `--random-wait` オプションで使用されるものと同じです。

DOWNLOAD_DELAY がゼロ (デフォルト) の場合、このオプションは効果がありません。

REACTOR_THREADPOOL_MAXSIZE

デフォルト: `10`

Twisted リアクター・スレッド・プール・サイズの最大制限。 これは、さまざまな Scrapy コンポーネントで使用される一般的な多目的スレッド・プールです。スレッド DNS リゾルバー、BlockingFeedStorage、S3FilesStore などがあります。ブロッキング IO が不十分な問題が発生している場合は、この値を増やします。

REDIRECT_MAX_TIMES

デフォルト: `20`

リクエストをリダイレクトできる最大回数を定義します。この最大値を超えると、リクエストのレスポンスがそのまま返されます。 私たちは Firefox の同じタスクのデフォルト値を使用しました。

REDIRECT_PRIORITY_ADJUST

デフォルト: `+2`

スコープ: `scrapy.downloadermiddlewares.redirect.RedirectMiddleware`

元のリクエストに対するリダイレクト・リクエストの優先度を調整する:

- **正の優先度調整 (デフォルト) は、より高い優先度を意味します** 。
- 負の優先度調整は、より低い優先度を意味します。

RETRY_PRIORITY_ADJUST

デフォルト: -1

スコープ: scrapy.downloadermiddlewares.retry.RetryMiddleware

元のリクエストに対する再試行要求の優先度を調整する:

- 正の優先度調整はより高い優先度を意味します。
- **負の優先度調整 (デフォルト) は、より低いを意味します** 。

ROBOTSTXT_OBEY

デフォルト: False

スコープ: scrapy.downloadermiddlewares.robotstxt

有効にすると、Scrapy は robots.txt ポリシーを尊重します。 詳細については、 *RobotsTxtMiddleware* を参照してください。

注釈: 歴史的な理由からデフォルト値は False ですが、このオプションは scrapy startproject コマンドによって生成された settings.py ファイルでデフォルトで有効になっています。

ROBOTSTXT_PARSER

デフォルト: 'scrapy.robotstxt.PythonRobotParser'

robots.txt ファイルの解析に使用するパーサー・バックエンド。詳細については、 *RobotsTxtMiddleware* を参照してください。

ROBOTSTXT_USER_AGENT

デフォルト: None

robots.txt ファイルでの照合に使用するユーザー・エージェント文字列。 None の場合、リクエストで送信する User-Agent ヘッダーまたは *USER_AGENT* 設定は、(この順序で、)robots.txt ファイルで使用するユーザー・エージェントを決定するために使用されます。

SCHEDULER

デフォルト: 'scrapy.core.scheduler.Scheduler'

クロールに使用するスケジューラー。

SCHEDULER_DEBUG

デフォルト: False

True に設定すると、リクエスト・スケジューラに関するデバッグ情報が記録されます。 現在、リクエストをディスクにシリアル化できない場合にログに記録されます (1 回のみ)。 統計カウンター (scheduler/unserializable) は、これが発生した回数を追跡します。

ログのエントリの例:

```
1956-01-31 00:00:00+0800 [scrapy.core.scheduler] ERROR: Unable to serialize request:
<GET http://example.com> - reason: cannot serialize <Request at 0x9a7c7ec>
(type Request)> - no more unserializable requests will be logged
(see 'scheduler/unserializable' stats counter)
```

SCHEDULER_DISK_QUEUE

デフォルト: 'scrapy.squeues.PickleLifoDiskQueue'

スケジューラが使用するディスク・キューのタイプ。 他の利用可能なタイプは、 scrapy.squeues.PickleFifoDiskQueue 、 scrapy.squeues.MarshalFifoDiskQueue 、 scrapy.squeues.MarshalLifoDiskQueue です。

SCHEDULER_MEMORY_QUEUE

デフォルト: 'scrapy.squeues.LifoMemoryQueue'

スケジューラが使用するメモリ内キューのタイプ。その他の利用可能なタイプは、 scrapy.squeues.FifoMemoryQueue です。

SCHEDULER_PRIORITY_QUEUE

デフォルト: 'scrapy.pqueues.ScrapyPriorityQueue'

スケジューラが使用する優先度キューのタイプ。 別の利用可能なタイプは scrapy.pqueues.DownloaderAwarePriorityQueue です。 多数の異なるドメインを並行してクロールする場合、scrapy.pqueues.DownloaderAwarePriorityQueue は scrapy.pqueues.ScrapyPriorityQueue よりも適切に機能します。しかし、現在 scrapy.pqueues.DownloaderAwarePriorityQueue は *CONCURRENT_REQUESTS_PER_IP* と一緒には機能しません。

SPIDER_CONTRACTS

デフォルト:: {}

プロジェクトで有効にされたスパイダー・コントラクトを含む辞書。スパイダーのテストに使用されます。 詳細については、 **スパイダー コントラクト** を参照してください。

SPIDER_CONTRACTS_BASE

デフォルト:

```
{
    'scrapy.contracts.default.UrlContract' : 1,
    'scrapy.contracts.default.ReturnsContract': 2,
    'scrapy.contracts.default.ScrapesContract': 3,
}
```

Scrapy でデフォルトで有効になっている Scrapy コントラクトを含む辞書。プロジェクトでこの設定を変更することは決してせず、代わりに *SPIDER_CONTRACTS* を変更してください。詳細については、 **スパイダー コントラクト** を参照してください。

SPIDER_CONTRACTS でクラス・パスに None を割り当てることで、これらのコントラクトを無効にできます。たとえば、組み込みの ScrapesContract を無効にするには、これを settings.py に配置します:

```
SPIDER_CONTRACTS = {
    'scrapy.contracts.default.ScrapesContract': None,
}
```

SPIDER_LOADER_CLASS

デフォルト: 'scrapy.spiderloader.SpiderLoader'

SpiderLoader API を実装する必要があるスパイダーのロードに使用されるクラス。

SPIDER_LOADER_WARN_ONLY

バージョン 1.3.3 で追加.

デフォルト: False

デフォルトでは、Scrapy が *SPIDER_MODULES* からスパイダー・クラスをインポートしようとすると、ImportError 例外があると大声で怒られます。けれども、 SPIDER_LOADER_WARN_ONLY = True を設定することで、この例外を黙らせて単純な警告に変えることができます。

注釈: いくつかの *scrapy* **コマンド** は、実際にはスパイダー・クラスをロードする必要がないため、この設定を True に設定して実行します（つまり、警告のみを発行し、失敗しません）: *scrapy runspider* 、 *scrapy settings* 、 *scrapy startproject* 、 *scrapy version*

SPIDER_MIDDLEWARES

デフォルト:: { }

プロジェクトで有効になっているスパイダー・ミドルウェアとその順序を含む辞書。 詳細については、 **スパイダー・ミドルウェアをアクティブにする** を参照してください。

SPIDER_MIDDLEWARES_BASE

デフォルト:

```
{
    'scrapy.spidermiddlewares.httperror.HttpErrorMiddleware': 50,
    'scrapy.spidermiddlewares.offsite.OffsiteMiddleware': 500,
    'scrapy.spidermiddlewares.referer.RefererMiddleware': 700,
    'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware': 800,
    'scrapy.spidermiddlewares.depth.DepthMiddleware': 900,
}
```

Scrapy でデフォルトで有効になっているスパイダー・ミドルウェアとその順序を含む辞書。 低次はエンジンに近く、高次はスパイダーに近い。 詳細については、 **スパイダー・ミドルウェアをアクティブにする** を参照してください。

SPIDER_MODULES

デフォルト: `[]`

Scrapy がスパイダーを探すモジュールのリスト。

例:

```
SPIDER_MODULES = ['mybot.spiders_prod', 'mybot.spiders_dev']
```

STATS_CLASS

デフォルト: `'scrapy.statscollectors.MemoryStatsCollector'`

統計収集器 *API* を実装する必要がある統計情報の収集に使用するクラス。

STATS_DUMP

デフォルト: `True`

スパイダーが終了したら *Scrapy stats* を (Scrapy ログに) ダンプします。

詳細は **統計をとる** 参照。

STATSMAILER_RCPTS

デフォルト: [] (空リスト)

スパイダーがスクレイピングを完了した後、スクレイピーの統計を送信します。 詳細については、*StatsMailer* を参照してください。

TELNETCONSOLE_ENABLED

デフォルト: True

telnet **コンソール** を有効にするかどうかを指定するブール値 (当該拡張機能も有効になっている場合)。

TELNETCONSOLE_PORT

デフォルト: [6023, 6073]

telnet コンソールに使用するポート範囲。 None または 0 に設定すると、動的に割り当てられたポートが使用されます。 詳細については、 *Telnet* **コンソール** を参照してください。

TEMPLATES_DIR

デフォルト: Scrapy モジュール内の templates ディレクトリ

startproject コマンドで新しいプロジェクトを作成し、 *genspider* コマンドで新しいスパイダーを作成するときにテンプレートを探すディレクトリ。

プロジェクト名は、 project サブディレクトリ内のカスタム・ファイルまたはディレクトリの名前と競合してはいけません。

URLLENGTH_LIMIT

デフォルト: 2083

スコープ: spidermiddlewares.urllength

クロールされた URL を許可する最大 URL 長。 この設定のデフォルト値の詳細については、 https://boutell.com/newfaq/misc/urllength.html を参照してください。

USER_AGENT

デフォルト: "Scrapy/VERSION (+https://scrapy.org)"

オーバーライドされない限り、クロール時に使用するデフォルトの User-Agent。このユーザー・エージェントは、*ROBOTSTXT_USER_AGENT* 設定が None であり、リクエストに指定された User-Agent ヘッダーが指定されてい

ない場合、 *RobotsTxtMiddleware* によっても使用されます。

他の場所で文書化された設定:

以下の設定は他の場所で文書化されています。それぞれの特定の場合をチェックして、それらを有効にして使用する方法を確認してください。

- *AJAXCRAWL_ENABLED*
- *AUTOTHROTTLE_DEBUG*
- *AUTOTHROTTLE_ENABLED*
- *AUTOTHROTTLE_MAX_DELAY*
- *AUTOTHROTTLE_START_DELAY*
- *AUTOTHROTTLE_TARGET_CONCURRENCY*
- *AWS_ACCESS_KEY_ID*
- *AWS_ENDPOINT_URL*
- *AWS_REGION_NAME*
- *AWS_SECRET_ACCESS_KEY*
- *AWS_USE_SSL*
- *AWS_VERIFY*
- *BOT_NAME*
- *CLOSESPIDER_ERRORCOUNT*
- *CLOSESPIDER_ITEMCOUNT*
- *CLOSESPIDER_PAGECOUNT*
- *CLOSESPIDER_TIMEOUT*
- *COMMANDS_MODULE*
- *COMPRESSION_ENABLED*
- *CONCURRENT_ITEMS*
- *CONCURRENT_REQUESTS*
- *CONCURRENT_REQUESTS_PER_DOMAIN*
- *CONCURRENT_REQUESTS_PER_IP*

- *COOKIES_DEBUG*
- *COOKIES_ENABLED*
- *DEFAULT_ITEM_CLASS*
- *DEFAULT_REQUEST_HEADERS*
- *DEPTH_LIMIT*
- *DEPTH_PRIORITY*
- *DEPTH_STATS_VERBOSE*
- *DNSCACHE_ENABLED*
- *DNSCACHE_SIZE*
- *DNS_TIMEOUT*
- *DOWNLOADER*
- *DOWNLOADER_CLIENTCONTEXTFACTORY*
- *DOWNLOADER_CLIENT_TLS_CIPHERS*
- *DOWNLOADER_CLIENT_TLS_METHOD*
- *DOWNLOADER_CLIENT_TLS_VERBOSE_LOGGING*
- *DOWNLOADER_HTTPCLIENTFACTORY*
- *DOWNLOADER_MIDDLEWARES*
- *DOWNLOADER_MIDDLEWARES_BASE*
- *DOWNLOADER_STATS*
- *DOWNLOAD_DELAY*
- *DOWNLOAD_FAIL_ON_DATALOSS*
- *DOWNLOAD_HANDLERS*
- *DOWNLOAD_HANDLERS_BASE*
- *DOWNLOAD_MAXSIZE*
- *DOWNLOAD_TIMEOUT*
- *DOWNLOAD_WARNSIZE*
- *DUPEFILTER_CLASS*

- *DUPEFILTER_DEBUG*
- *EDITOR*
- *EXTENSIONS*
- *EXTENSIONS_BASE*
- *FEED_EXPORTERS*
- *FEED_EXPORTERS_BASE*
- *FEED_EXPORT_ENCODING*
- *FEED_EXPORT_FIELDS*
- *FEED_EXPORT_INDENT*
- *FEED_FORMAT*
- *FEED_STORAGES*
- *FEED_STORAGES_BASE*
- *FEED_STORAGE_FTP_ACTIVE*
- *FEED_STORAGE_S3_ACL*
- *FEED_STORE_EMPTY*
- *FEED_TEMPDIR*
- *FEED_URI*
- *FILES_EXPIRES*
- *FILES_RESULT_FIELD*
- *FILES_STORE*
- *FILES_STORE_GCS_ACL*
- *FILES_STORE_S3_ACL*
- *FILES_URLS_FIELD*
- *FTP_PASSIVE_MODE*
- *FTP_PASSWORD*
- *FTP_USER*
- *GCS_PROJECT_ID*

- *HTTPCACHE_ALWAYS_STORE*
- *HTTPCACHE_DBM_MODULE*
- *HTTPCACHE_DIR*
- *HTTPCACHE_ENABLED*
- *HTTPCACHE_EXPIRATION_SECS*
- *HTTPCACHE_GZIP*
- *HTTPCACHE_IGNORE_HTTP_CODES*
- *HTTPCACHE_IGNORE_MISSING*
- *HTTPCACHE_IGNORE_RESPONSE_CACHE_CONTROLS*
- *HTTPCACHE_IGNORE_SCHEMES*
- *HTTPCACHE_POLICY*
- *HTTPCACHE_STORAGE*
- *HTTPERROR_ALLOWED_CODES*
- *HTTPERROR_ALLOW_ALL*
- *HTTPPROXY_AUTH_ENCODING*
- *HTTPPROXY_ENABLED*
- *IMAGES_EXPIRES*
- *IMAGES_MIN_HEIGHT*
- *IMAGES_MIN_WIDTH*
- *IMAGES_RESULT_FIELD*
- *IMAGES_STORE*
- *IMAGES_STORE_GCS_ACL*
- *IMAGES_STORE_S3_ACL*
- *IMAGES_THUMBS*
- *IMAGES_URLS_FIELD*
- *ITEM_PIPELINES*
- *ITEM_PIPELINES_BASE*

- *LOGSTATS_INTERVAL*
- *LOG_DATEFORMAT*
- *LOG_ENABLED*
- *LOG_ENCODING*
- *LOG_FILE*
- *LOG_FORMAT*
- *LOG_FORMATTER*
- *LOG_LEVEL*
- *LOG_SHORT_NAMES*
- *LOG_STDOUT*
- *MAIL_FROM*
- *MAIL_HOST*
- *MAIL_PASS*
- *MAIL_PORT*
- *MAIL_SSL*
- *MAIL_TLS*
- *MAIL_USER*
- *MEDIA_ALLOW_REDIRECTS*
- *MEMDEBUG_ENABLED*
- *MEMDEBUG_NOTIFY*
- *MEMUSAGE_CHECK_INTERVAL_SECONDS*
- *MEMUSAGE_ENABLED*
- *MEMUSAGE_LIMIT_MB*
- *MEMUSAGE_NOTIFY_MAIL*
- *MEMUSAGE_WARNING_MB*
- *METAREFRESH_ENABLED*
- *METAREFRESH_IGNORE_TAGS*

- *METAREFRESH_MAXDELAY*
- *NEWSPIDER_MODULE*
- *RANDOMIZE_DOWNLOAD_DELAY*
- *REACTOR_THREADPOOL_MAXSIZE*
- *REDIRECT_ENABLED*
- *REDIRECT_MAX_TIMES*
- *REDIRECT_MAX_TIMES*
- *REDIRECT_PRIORITY_ADJUST*
- *REFERER_ENABLED*
- *REFERRER_POLICY*
- *RETRY_ENABLED*
- *RETRY_HTTP_CODES*
- *RETRY_PRIORITY_ADJUST*
- *RETRY_TIMES*
- *ROBOTSTXT_OBEY*
- *ROBOTSTXT_PARSER*
- *ROBOTSTXT_USER_AGENT*
- *SCHEDULER*
- *SCHEDULER_DEBUG*
- *SCHEDULER_DISK_QUEUE*
- *SCHEDULER_MEMORY_QUEUE*
- *SCHEDULER_PRIORITY_QUEUE*
- *SPIDER_CONTRACTS*
- *SPIDER_CONTRACTS_BASE*
- *SPIDER_LOADER_CLASS*
- *SPIDER_LOADER_WARN_ONLY*
- *SPIDER_MIDDLEWARES*

- *SPIDER_MIDDLEWARES_BASE*
- *SPIDER_MODULES*
- *STATSMAILER_RCPTS*
- *STATS_CLASS*
- *STATS_DUMP*
- *TELNETCONSOLE_ENABLED*
- *TELNETCONSOLE_HOST*
- *TELNETCONSOLE_PASSWORD*
- *TELNETCONSOLE_PORT*
- *TELNETCONSOLE_PORT*
- *TELNETCONSOLE_USERNAME*
- *TEMPLATES_DIR*
- *URLLENGTH_LIMIT*
- *USER_AGENT*

3.12 例外 (Exceptions)

3.12.1 組み込み例外リファレンス

Scrapy に含まれるすべての例外とその使用法のリストを次に示します。

DropItem

exception scrapy.exceptions.**DropItem**

アイテムの処理を停止するためにアイテム・パイプライン・ステージによって発生させる必要がある例外。 詳細については **アイテム・パイプライン** を参照してください。

CloseSpider

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

この例外は、スパイダーのクローズまたは停止を要求するスパイダーコールバックから発生する可能性があります。サポートされている引数は以下です:

パラメータ **reason** (*str*) – クローズの理由

例えば:

```
def parse_page(self, response):
    if 'Bandwidth exceeded' in response.body:
        raise CloseSpider('bandwidth_exceeded')
```

DontCloseSpider

exception scrapy.exceptions.**DontCloseSpider**

この例外は、*spider_idle* シグナルハンドラーで発生させて、スパイダーが閉じないようにすることができます。

IgnoreRequest

exception scrapy.exceptions.**IgnoreRequest**

この例外は、スケジューラまたはダウンローダー・ミドルウェアによって発生し、要求を無視する必要があることを示します。

NotConfigured

exception scrapy.exceptions.**NotConfigured**

一部のコンポーネントは、この例外を発生させて、無効のままにすることを示すことができます。以下のコンポーネントが含まれます:

- 拡張機能
- アイテム・パイプライン
- ダウンローダー・ミドルウェア
- スパイダー・ミドルウェア

コンポーネントの `__init__` メソッドで例外を発生させる必要があります。

NotSupported

exception scrapy.exceptions.**NotSupported**

この例外は、サポートされていない機能を示すために発生します。

コマンドラインツール Scrapy プロジェクトの管理に使用するコマンドライン・ツールについて学習します。

スパイダー Web サイトをクロールするルールを作成します。

セレクター XPath を使用して Web ページからデータを抽出します。

Scrapy **シェル** 対話環境で抽出コードをテストします。

アイテム あなたがスクレイピングしたいと欲するデータを定義します。

アイテム・ローダー 抽出したデータをあなたのアイテムに入れます。

アイテム・パイプライン スクレイピングしたデータを後処理して保存します。

フィード・エクスポート さまざまな形式を使用して、さまざまストレージに、スクレイピングされたデータを出力します。

リクエストとレスポンス HTTP リクエストとレスポンスを表すために使用されるクラスを理解します。

リンク抽出器 *(extractor)* ページからたどるリンクを抽出する便利なクラス。

設定 Scrapy をどのように設定するか学びます。 *available settings* 参照。

例外 *(Exceptions)* Scrapy で利用可能な全ての例外とその意味

第 4 章

組み込み済サービス群

4.1 ロギング (logging)

注釈: `scrapy.log` は、Python 標準ロギングの明示的な呼び出しを支援する機能とともに非推奨になりました。新しいロギングシステムの詳細については、以下をご覧ください。

Scrapy は、イベント・ロギングに Python の組み込みロギング・システム (Python's builtin logging system) を使用します。 始めるための簡単な例をいくつか紹介しますが、より高度なユース・ケースについては、Python の組み込みロギング・システムのドキュメントを徹底的に読むことを強くお勧めします。

ロギングはそのままで機能し、 **ロギング設定** にリストされている Scrapy 設定である程度設定できます。

Scrapy は `scrapy.utils.log.configure_logging()` を呼び出していくつかの妥当なデフォルトを設定し、コマンドを実行するときに **ロギング設定** でそれらの設定を処理するため、**スクリプトから *Scrapy* を実行する** で説明されているスクリプトから Scrapy を実行している場合は、手動で呼び出すことをお勧めします。

4.1.1 ログ・レベル

Python の組み込みロギングは、特定のログメッセージの重大度を示す 5 つの異なるレベルを定義します。 以下は標準のもので、降順でリストされています:

1. `logging.CRITICAL` - 致命的なエラーの場合 (最高の重要度)
2. `logging.ERROR` - 通常のエラーの場合
3. `logging.WARNING` - 警告メッセージの場合
4. `logging.INFO` - 情報メッセージ用
5. `logging.DEBUG` - デバッグメッセージ用 (最も低い重要度)

4.1.2 メッセージをログ出しする方法

logging.WARNING レベルを使用してメッセージをログ出しする方法の簡単な例を次に示します:

```
import logging
logging.warning("This is a warning")
```

標準の 5 つのレベルのいずれかでログメッセージを発行するためのショートカットがあり、引数として指定されたレベルを取る一般的な logging.log メソッドもあります。必要に応じて、さっきの例を次のように書き換えることができます:

```
import logging
logging.log(logging.WARNING, "This is a warning")
```

さらに、あなたはメッセージをカプセル化するためのさまざまなロガーを作成できます。(たとえば、一般的な方法は、モジュールごとに異なるロガーを作成することです)。 これらのロガーは独立して構成でき、階層構造が可能です。

前の例では、舞台裏でルート・ロガーを使用します。これは、特に指定がない限り、すべてのメッセージが伝播されるトップ・レベル・ロガーです。 logging ヘルパーの使用は、ルート・ロガーを明示的に取得するための単なるショートカットであるため、これは最後のコード片と同等です。

```
import logging
logger = logging.getLogger()
logger.warning("This is a warning")
```

logging.getLogger 関数で名前を取得するだけで、異なるロガーを使用できます:

```
import logging
logger = logging.getLogger('mycustomlogger')
logger.warning("This is a warning")
```

最後に、 __name__ 変数を使用して、作業中のモジュールのカスタム・ロガーを確保できます。これには、現在のモジュールのパスが入力されます:

```
import logging
logger = logging.getLogger(__name__)
logger.warning("This is a warning")
```

参考:

モジュール・ロギング HowTo　基本ロギング・チュートリアル (https://docs.python.org/2/howto/logging.html)

モジュール・ロギング Loggers　ロガーに関する詳細なドキュメント (https://docs.python.org/2/library/logging.html#logger-objects)

4.1.3 スパイダーからのロギング

Scrapy は、各スパイダー・インスタンス内で *logger* を提供します。

```
import scrapy

class MySpider(scrapy.Spider):

    name = 'myspider'
    start_urls = ['https://scrapinghub.com']

    def parse(self, response):
        self.logger.info('Parse function called on %s', response.url)
```

そのロガーはスパイダーの名前を使用して作成されますが、任意のカスタム Python ロガーを使用できます。 例えば以下です:

```
import logging
import scrapy

logger = logging.getLogger('mycustomlogger')

class MySpider(scrapy.Spider):

    name = 'myspider'
    start_urls = ['https://scrapinghub.com']

    def parse(self, response):
        logger.info('Parse function called on %s', response.url)
```

4.1.4 ロギング構成 (configuration)

ロガー自身は、ロガーを介して送信されたメッセージの表示方法を管理しません。 このタスクでは、さまざまなハンドラーを任意のロガー・インスタンスにアタッチして、それらのメッセージを標準出力、ファイル、電子メールなどの適切な宛先にリダイレクトします。

デフォルトでは、Scrapy は以下の設定に基づいて、ルート・ロガーのハンドラーを設定および構成します。

ロギング設定

これらの設定は、ロギングの構成 (configuration) に使用できます:

- *LOG_FILE*
- *LOG_ENABLED*

- *LOG_ENCODING*
- *LOG_LEVEL*
- *LOG_FORMAT*
- *LOG_DATEFORMAT*
- *LOG_STDOUT*
- *LOG_SHORT_NAMES*

最初のいくつかの設定は、ログメッセージの宛先を定義します。 *LOG_FILE* が設定されている場合、ルート・ロガーを介して送信されたメッセージは *LOG_ENCODING* エンコーディングで *LOG_FILE* という名前のファイルにリダイレクトされます。設定が解除され、 *LOG_ENABLED* が `True` の場合、ログメッセージは標準エラーに表示されます。 そして、 *LOG_ENABLED* が `False` の場合、目に見えるログ出力はありません。

LOG_LEVEL は表示する重大度の最小レベルを決定し、指定より重大度の低いメッセージは除外されます。 **ログ・レベル** にリストされている可能なレベルを範囲としています。

LOG_FORMAT と *LOG_DATEFORMAT* は、すべてのメッセージのレイアウトとして使用されるフォーマット文字列を指定します。これらの文字列には、 それぞれ、ロギングのログ・レコード属性文書 (logging's logrecord attributes docs) や、日時の strftime および strptime ディレクティブ (datetime's strftime and strptime directives) にリストされているプレース・ホルダーを含めることができます。

LOG_SHORT_NAMES が設定されている場合、ログはログを印刷する Scrapy コンポーネントを表示しません。 デフォルトでは設定されていないため、ログにはそのログ出力の原因となる Scrapy コンポーネントが含まれています。

コマンド・ライン・オプション

すべてのコマンドで使用できるコマンドライン引数があり、これを使用してロギングに関する Scrapy 設定の一部をオーバーライドできます。

- `--logfile FILE` *LOG_FILE* をオーバーライドする
- `--loglevel/-L LEVEL` *LOG_LEVEL* をオーバーライドする
- `--nolog` *LOG_ENABLED* を `False` に設定

参考:

logging.handlers **モジュール** 利用可能なハンドラーに関する詳細なドキュメント

カスタム・ログ書式

`LogFormatter` クラスを拡張し、 *LOG_FORMATTER* が新しいクラスを指すようにすることで、さまざまなアクションに対してカスタム・ログ形式を設定できます。

高度なカスタマイズ

Scrapy は stdlib ロギング・モジュールを使用するため、stdlib ロギングのすべての機能を使用してロギングをカスタマイズできます。

たとえば、多くの「HTTP 404」や「HTTP 500」のレスポンスを返す Web サイトをスクレイピングしていて、このようなすべてのメッセージを非表示にしたいとします:

```
2016-12-16 22:00:06 [scrapy.spidermiddlewares.httperror] INFO: Ignoring
response <500 http://quotes.toscrape.com/page/1-34/>: HTTP status code
is not handled or not allowed
```

最初に注意することはロガー名です。角括弧内に書きます。`[scrapy.spidermiddlewares.httperror]`。単に `[scrapy]` を取得した場合、*LOG_SHORT_NAMES* はおそらく True に設定されています。False に設定して、クロールを再実行します。

次に、メッセージに INFO レベルがあることがわかります。非表示にするには、INFO よりも高い `scrapy.spidermiddlewares.httperror` のログ・レベルを設定する必要があります。 INFO の次のレベルは WARNING です。 スパイダー `__init__` メソッド内でその設定を行うことができます:

```
import logging
import scrapy


class MySpider(scrapy.Spider):
    # ...
    def __init__(self, *args, **kwargs):
        logger = logging.getLogger('scrapy.spidermiddlewares.httperror')
        logger.setLevel(logging.WARNING)
        super().__init__(*args, **kwargs)
```

あなたがこのスパイダーを再度実行すると、`scrapy.spidermiddlewares.httperror` ロガーからの INFO メッセージはなくなります。

4.1.5 scrapy.utils.log モジュール

4.2 統計をとる

Scrapy は、値が多くの場合カウンターである、キー/値の形式で統計を収集するための便利な機能を提供します。この機能は統計収集器 (Stats Collector) と呼ばれ、以下の **一般的な統計収集器を使う** 節の例にあるように、**クローラー** *API* の *stats* 属性を通じてアクセスできます。

ただし、統計収集器 (Stats Collector) は常に使用可能なため、統計収集が有効かどうかに関係なく、モジュールにいつでもインポートして、API を使用 (新しい統計キーをインクリメントまたは設定) できます。無効にしても、API は機能しますが、何も収集しません。これは、統計収集器の使用を簡素化することを目的としています。スパ

イダー、Scrapy 拡張機能、または統計収集器を使用しているコードの統計を取るために、1 行以上のコードを費やす必要はありません。

統計収集器 (Stats Collector) のもう 1 つの特色は、(有効な場合) とても効率的で、無効な場合は極めて効率的 (ほとんど気付かない) だということです。

統計収集器は、開いているスパイダーごとに統計テーブルを保持します。このテーブルは、スパイダーを開くと自動的に開き、スパイダーを閉じると閉じます。

4.2.1 一般的な統計収集器を使う

stats 属性を介して統計収集器 (stats collector) にアクセスします。統計にアクセスする拡張機能の例を次に示します:

```
class ExtensionThatAccessStats(object):

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

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

統計値設定:

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

統計値加算:

```
stats.inc_value('custom_count')
```

以前より大きい場合のみ統計値設定:

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

以前より小さい場合のみ統計値設定:

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

統計値取得:

```
>>> stats.get_value('custom_count')
1
```

全統計取得:

```
>>> stats.get_stats()
{'custom_count': 1, 'start_time': datetime.datetime(2009, 7, 14, 21, 47, 28, 977139)}
```

4.2.2 利用可能な統計収集器

基本的な StatsCollector に加えて、基本的な統計収集器を拡張する Scrapy で利用可能な他の統計収集器があります。 *STATS_CLASS* 設定を使用して、使用する統計収集器を選択できます。 使用されるデフォルトの統計収集器は MemoryStatsCollector です。

MemoryStatsCollector

class scrapy.statscollectors.**MemoryStatsCollector**

スパイダーが閉じられた後、(各スパイダーの）最後のスクレイピング実行の統計をメモリに保持する単純な統計収集器。 統計は、 *spider_stats* 属性を介してアクセスできます。これは、スパイダー・ドメイン名をキーとする辞書です。

これは、Scrapy で使用されるデフォルトの統計収集器です。

spider_stats

各スパイダーの最後のスクレイピング実行の統計を含む (スパイダー名をキーとする) 辞書の辞書

DummyStatsCollector

class scrapy.statscollectors.**DummyStatsCollector**

(何もしないので) 非常に効率的ですが、何もしない統計収集器。この統計収集器は、 *STATS_CLASS* 設定を介して設定でき、パフォーマンスを改善するために統計収集を無効にします。 ただし、統計収集のパフォーマンス・ペナルティは、通常、ページの解析などの他の Scrapy ワーク・ロードと比較してわずかです。

4.3 電子メールの送信

Python は smtplib ライブラリを介して電子メールを比較的簡単に送信できますが、Scrapy は電子メールを送信するための独自の機能を提供します。これは非常に使いやすく、クローラーの非ブロッキング IO の干渉を避けるため、 Twisted non-blocking IO を使用して実装されています。また、添付ファイルを送信するためのシンプルな API を提供し、いくつかの **電子メール設定** で非常に簡単に構成 (configure) できます。

4.3.1 簡単な例

メール送信者をインスタンス化するには 2 つの方法があります。あなたは標準コンストラクタを使用してインスタンス化できます:

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

または、 **電子メール設定** を尊重する Scrapy 設定オブジェクトを渡してインスタンス化できます。

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

そして、以下を使用して (添付ファイルなしで) 電子メールを送信する方法があります:

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

4.3.2 MailSender クラス・リファレンス

MailSender は、フレームワークの残りの部分と同様に、 Twisted non-blocking IO を使用するため、Scrapy から電子メールを送信するために使用する優先クラスです。

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

パラメータ

- **smtphost** (*str or bytes*) – 電子メールの送信に使用する SMTP ホスト。省略すると、*MAIL_HOST* 設定が使用されます。
- **mailfrom** (*str*) – メールの送信に使用されるアドレス (`From:` ヘッダー内)。 省略すると、*MAIL_FROM* 設定が使用されます。
- **smtpuser** – SMTP ユーザー。省略すると、 *MAIL_USER* 設定が使用されます。指定しない場合、SMTP 認証は実行されません。
- **smtppass** (*str or bytes*) – 認証用の SMTP パスワード
- **smtpport** (*int*) – 接続のための SMTP ポート番号
- **smtptls** (*boolean*) – SMTP STARTTLS の使用を強制する
- **smtpssl** (*boolean*) – 安全な SSL 接続の使用を強制する

classmethod from_settings(*settings*)

これらの *Scrapy* 電子メール設定 を尊重する Scrapy 設定オブジェクトを使用してインスタンス化します。

パラメータ **settings** (`scrapy.settings.Settings` object) – 電子メールの受信者

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

指定された受信者にメールを送信します。

パラメータ

- **to** (*str or list of str*) – 電子メールの受信者
- **subject** (*str*) – 電子メールの件名
- **cc** (*str or list of str*) – カーボン・コピー (CC) への電子メール
- **body** (*str*) – 電子メール本文
- **attachs** (*iterable*) – タプル `(attach_name, mimetype, file_object)` の反復可能オブジェクト (iterable)。 ここで、 `attach_name` は、電子メールの添付ファイルに表示される名前の文字列で、 `mimetype` は添付ファイルの MIME タイプであり、`file_object` は添付ファイルの内容を含む読み取り可能なファイルオブジェクトです
- **mimetype** (*str*) – 電子メールの MIME タイプ
- **charset** (*str*) – 電子メールのコンテンツに使用する文字エンコード

4.3.3 メール設定

これらの設定は、 *MailSender* クラスのデフォルトのコンストラクター値を定義し、コードを記述せずにプロジェクトで電子メール通知を構成 (configure) するために使用できます (*MailSender* を使用する拡張機能およびコード用))。

MAIL_FROM

デフォルト: `'scrapy@localhost'`

電子メールの送信に使用する送信者の電子メール (`From:` ヘッダー)。

MAIL_HOST

デフォルト: `'localhost'`

電子メールの送信に使用する SMTP ホスト。

MAIL_PORT

デフォルト: `25`

電子メールの送信に使用する SMTP ポート。

MAIL_USER

デフォルト: None

SMTP 認証に使用するユーザー。無効にすると、SMTP 認証は実行されません。

MAIL_PASS

デフォルト: None

MAIL_USER とともに、SMTP 認証に使用するパスワード。

MAIL_TLS

デフォルト: False

STARTTLS を使用を強制します。STARTTLS は、既存の安全でない接続を取得し、SSL/TLS を使用して安全な接続にアップグレードする方法です。

MAIL_SSL

デフォルト: False

SSL 暗号化接続を使用した接続を強制する

4.4 Telnet コンソール

Scrapy には、Scrapy 実行中のプロセスを検査および制御するための組み込みの telnet コンソールが付属しています。telnet コンソールは、Scrapy プロセス内で実行される通常の python シェルであるため、文字通り何でもできます。

telnet コンソールは **組み込みの** *Scrapy* **拡張機能** であり、デフォルトで有効になっていますが、必要に応じて無効にすることもできます。拡張機能自体の詳細については、 *Telnet* **コンソール拡張機能** を参照してください。

> **警告:** telnet はトランスポート層セキュリティを提供しないため、パブリック・ネットワーク経由で telnet コンソールを使用することは安全ではありません。 ユーザー名/パスワード認証を使用しても、それは変わりません。
>
> 意図している使用方法は、実行中の Scrapy スパイダーにローカル (スパイダー・プロセスと telnet クライアントが同じマシン上にある) または安全な接続 (VPN、SSH トンネル) に接続することです。安全でない接続では telnet コンソールを使用しないようにするか、 *TELNETCONSOLE_ENABLED* オプションを使用して完全に無効にしてください。

4.4.1 telnet コンソールにアクセスする方法

telnet コンソールは *TELNETCONSOLE_PORT* 設定で定義された TCP ポートでリッスンします。デフォルトは `6023` です。コンソールにアクセスするには次のように入力する必要があります:

```
telnet localhost 6023
Trying localhost...
Connected to localhost.
Escape character is '^]'.
Username:
Password:
>>>
```

デフォルトでは、ユーザー名は `scrapy` であり、パスワードは自動生成されます。自動生成されたパスワードは、以下の例のように Scrapy ログで確認できます:

```
2018-10-16 14:35:21 [scrapy.extensions.telnet] INFO: Telnet Password: 16f92501e8a59326
```

デフォルトのユーザー名とパスワードは、設定 `TELNET_CONSOLE_USERNAME` と *TELNETCONSOLE_PASSWORD* で上書きできます。

警告: telnet は安全なトランスポートを使用していないため、ユーザー名とパスワードは限定的な保護しか提供しません。デフォルトでは、ユーザー名とパスワードが設定されていてもトラフィックは暗号化されません。

Windows、およびほとんどの Linux ディストリビューションにデフォルトでインストールされる telnet プログラムが必要です。

4.4.2 telnet コンソールで使用可能な変数

telnet コンソールは、Scrapy プロセス内で実行される通常の Python シェルのように、新しいモジュールのインポートなど、あらゆる操作を行うことができます。

けれども、Telnet コンソールには、便宜上いくつかのデフォルト変数が定義されています:

ショートカット	説明
crawler	Scrapy クローラー (*scrapy.crawler.Crawler* オブジェクト)
engine	Crawler.engine 属性
spider	現在アクティブなスパイダー
slot	エンジン・スロット (engine slot)
extensions	拡張機能マネージャー (Crawler.extensions 属性)
stats	統計収集器 (stats collector)(Crawler.stats 属性)
settings	Scrapy 設定オブジェクト (Crawler.settings 属性)
est	Scrapy エンジンのステータスレポートを出力
prefs	メモリ・デバッグ用 (**メモリ・リークのデバッグ** 参照)
p	pprint.pprint 関数へのショートカット
hpy	メモリ・デバッグ用 (**メモリ・リークのデバッグ** 参照)

4.4.3 Telnet コンソール使用例

telnet コンソールで実行できるタスクの例を次に示します:

Scrapy エンジンのステータスを表示

あなたは Scrapy エンジンの `est()` メソッドを使用して、telnet コンソールを使用してその状態をすばやく表示できます。

```
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
```

Scrapy エンジンを一時停止、再開、停止する

一時停止するためには:

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

(一時停止したのを) 再開するためには:

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

停止 (再開不可) するためには:

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

4.4.4 Telnet コンソール・シグナル

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

telnet コンソールが開く直前に送信されます。この信号に接続して、telnet ローカル名前空間で使用できる変数を追加、削除、または更新できます。そのためには、ハンドラーの `telnet_vars` 辞書を更新する必要があります。

パラメータ **telnet_vars** (*dict*) – telnet 変数の辞書

4.4.5 Telnet 設定

これらは、Telnet コンソールの振る舞いを制御する設定です:

TELNETCONSOLE_PORT

デフォルト: `[6023, 6073]`

telnet コンソールに使用するポート範囲。 `None` または `0` に設定すると、動的に割り当てられたポートが使用されます。

TELNETCONSOLE_HOST

デフォルト: `'127.0.0.1'`

telnet コンソールがリッスンするネットワーク・インターフェイス

TELNETCONSOLE_USERNAME

デフォルト: `'scrapy'`

telnet コンソールに使用されるユーザー名

TELNETCONSOLE_PASSWORD

デフォルト: `None`

telnet コンソールに使用されるパスワード。デフォルトの動作では自動生成されます

4.5 Web サービス

webservice は別のプロジェクトに移動しました。

以下でホストされています:

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

ロギング *(logging)*　Scrapy で Python 組み込みのログ機能を使う方法を習います。

統計をとる　あなたのスクレイピング・クローラーの統計を収集します。

電子メールの送信　特定のイベントが発生したときに電子メールを送信します。

Telnet **コンソール**　組み込みの Python コンソールを使って、実行している最中のクローラーを詳しく調べます。

Web **サービス**　Web サービスを使用してクローラーを監視および制御します。

第 5 章

特定の問題の解決

5.1 F.A.Q.(よくある質問と回答)

5.1.1 Scrapy は BeautifulSoup や lxml と比較してどうですか？

BeautifulSoup と lxml は、HTML と XML を解析するためのライブラリです。 Scrapy は、Web サイトをクロールし、そこからデータを抽出する Web スパイダーを作成するためのアプリケーションフレームワークです。

Scrapy はデータを抽出するための組み込みメカニズムを提供します (**セレクター** とよばれます) が、より快適に作業できる場合は、代わりに BeautifulSoup (または lxml) を簡単に使用できます。結局のところ、それらは任意の Python コードからインポートして使用できるライブラリを利用しているだけです。

いいかえると、 BeautifulSoup (または lxml) と Scrapy を比較することは、 jinja2 と Django を比較するようなものです。

5.1.2 Scrapy で BeautifulSoup を使用できますか？

はい、できます。 **上記** のように、 BeautifulSoup は Scrapy コールバックで HTML レスポンスをパースするために使用できます。 レスポンスのボディを `BeautifulSoup` オブジェクトに送り、必要なデータを抽出するだけです。

HTML パーサーとして `lxml` を使用して、BeautifulSoup API を使用するスパイダーの例を次に示します:

```
from bs4 import BeautifulSoup
import scrapy


class ExampleSpider(scrapy.Spider):
    name = "example"
    allowed_domains = ["example.com"]
    start_urls = (
```

(次のページに続く)

(前のページからの続き)

```
        'http://www.example.com/',
    )

    def parse(self, response):
        # use lxml to get decent HTML parsing speed
        soup = BeautifulSoup(response.text, 'lxml')
        yield {
            "url": response.url,
            "title": soup.h1.string
        }
```

注釈: `BeautifulSoup` はいくつかの HTML/XML パーサーをサポートしています。利用可能なものについては BeautifulSoup の公式ドキュメント (BeautifulSoup's official documentation) を参照してください。

5.1.3 Scrapy はどの Python バージョンをサポートしていますか？

Scrapy は CPython(デフォルトの Python 実装) での Python 2.7 および Python 3.5+ と、PyPy(PyPy 5.9 以降) でサポートされています。Python 2.6 のサポートは Scrapy 0.20 以降は削除されました。Python 3 のサポートは Scrapy 1.1 で追加されました。 PyPy サポートは Scrapy 1.4 で追加され、PyPy3 サポートは Scrapy 1.5 で追加されました。

注釈: Windows で Python 3 をサポートするには、**インストールガイドで概説** しているように、Anaconda/Miniconda を使用することをお勧めします。

5.1.4 Scrapy は Django から hogehoge を盗んだ？

たぶん。だけど、私たちはそういう言い方はしないな。 Django は素晴らしいオープンソースプロジェクトであり、従うべき例であると考えているため、Scrapy の着想を得るのに利用したんだよ。

車輪の再発明する必要はないという信念です。この信念は、オープンソースおよびフリーソフトウェアの基礎の 1 つであることに加えて、ソフトウェアだけでなく、ドキュメント、手順、ポリシーなどにも適用されます。したがって、各問題を自分で進めるのではなく、それらのプロジェクトからアイデアをコピーすることを選択しますそれが既に各問題を適切に解決しているので、私たちは解決する必要がある実際の問題に焦点を当てる事ができます。

私たちは Scrapy が他のプロジェクトのインスピレーションとして役立つことを誇りに思います。 じゃんじゃん盗め！

5.1.5 Scrapy は HTTP プロキシ経由で動作しますか？

はい。 HTTP プロキシのサポートは、HTTP プロキシ・ダウンローダー・ミドルウェアを通じて提供されます (Scrapy 0.8 以降)。 *HttpProxyMiddleware* を参照してください。

5.1.6 異なるページの属性を持つアイテムをスクレイピングするにはどうすればよいですか？

追加のデータをコールバック関数に渡す 参照。

5.1.7 Scrapy がクラッシュします。「ImportError: No module named win32api」

あなたは「このツイストバグ」(this Twisted bug) のため、 pywin32 をインストールする必要があります。

5.1.8 スパイダーでユーザーログインをシミュレートするにはどうすればよいですか？

FormRequest.from_response() **を使用してユーザーログインをシミュレートする** 参照。

5.1.9 Scrapy は幅 (breadth) 優先または深さ (depth) 優先でクロールしますか？

デフォルトでは、Scrapy は保留中のリクエストを保存するために LIFO キューを使用します。これは基本的に、DFO 順序 (DFO order) でクロールすることを意味します。 ほとんどの場合、この順序の方が便利です。

あなたが本当に BFO 順 (BFO order) でクロールしたい場合は、次の設定を行うことで実行できます:

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

保留中のリクエストが *CONCURRENT_REQUESTS* または *CONCURRENT_REQUESTS_PER_DOMAIN* または *CONCURRENT_REQUESTS_PER_DOMAIN* の設定値を下回っている間、これらのリクエストは同時に送信されます。その結果、クロールの最初のいくつかのリクエストが目的の順序に従うことはほとんどありません。 これらの設定を 1 に下げると、目的の順序が強制されますが、クロール全体が大幅に遅くなります。

5.1.10 Scrapy クローラーにメモリリークがあります。 何か私にできる事がありますか？

メモリ・リークのデバッグ 参照。

また、Python には *Scrapy* **ではリークしてないのにリークしてる** *orz* で説明されている組み込みのメモリリークの問題があります。

5.1.11 Scrapy が消費するメモリを減らすにはどうすればよいですか？

1 つ前の質問を見て下さい。

5.1.12 スパイダーは基本 HTTP 認証を使用できますか？

はい。 *HttpAuthMiddleware* 参照。

5.1.13 Scrapy が母国語ではなく英語でページをダウンロードするのはなぜですか？

DEFAULT_REQUEST_HEADERS 設定をオーバーライドして、デフォルトの Accept-Language リクエスト・ヘッダーを変更してみてください。

5.1.14 Scrapy プロジェクトの例はどこにありますか？

例 参照。

5.1.15 プロジェクトを作成せずにスパイダーを実行できますか？

はい。 *runspider* コマンドを使用できます。たとえば、 `my_spider.py` ファイルにスパイダーが記述されている場合は、次のコマンドで実行できます:

```
scrapy runspider my_spider.py
```

詳細については *runspider* を参照してください。

5.1.16 "Filtered offsite request"(フィルターされたオフサイト要求) メッセージが表示されます。 どうすれば修正できますか？

これらのメッセージ (DEBUG レベルでログに記録される) は、必ずしも問題があることを意味するわけではないため、修正する必要はありません。

これらのメッセージは、オフサイト・スパイダー・ミドルウェアによって送出されます。オフサイト・スパイダー・ミドルウェアは、スパイダーの対象外のドメインへのリクエストをフィルター処理することを目的とするスパイダー・ミドルウェア (デフォルトで有効) です。

詳細は *OffsiteMiddleware* を参照して下さい。

5.1.17 Scrapy クローラーを運用環境に展開する推奨方法は何ですか？

スパイダーのデプロイ 参照。

5.1.18 大規模なエクスポートに JSON を使用できますか？

出力の大きさに依存します。 *JsonItemExporter* 文書の **警告** を参照してください。

5.1.19 シグナルハンドラーから (Twisted) 遅延 (deferred) を返すことができますか？

ハンドラーからの遅延 (deferred) を返すことをサポートするシグナルもあれば、サポートしないシグナルもあります。 **組み込みシグナル・リファレンス** を参照して、どれがどれか確認してください。

5.1.20 レスポンス・ステータス・コード 999 の意味は何ですか？

999 は、リクエストを抑制するために Yahoo サイトで使用されるカスタム・レスポンス・ステータス・コードです。スパイダーで 2 (またはそれ以上) のダウンロード遅延を使用して、クロール速度を遅くしてみてください:

```
class MySpider(CrawlSpider):

    name = 'myspider'

    download_delay = 2

    # [ ... rest of the spider code ... ]
```

または、 *DOWNLOAD_DELAY* 設定でプロジェクトのグローバル・ダウンロード遅延を設定します。

5.1.21 スパイダーから `pdb.set_trace()` を呼び出してデバッグできますか？

はい。ただし、スパイダーによって処理されているレスポンスをすばやく分析 (および変更) できる Scrapy シェルを使用することもできます。これは、通常の `pdb.set_trace()` よりも非常に便利です。

詳細は **スパイダーからシェルを呼び出してレスポンスを検査する** を参照して下さい。

5.1.22 スクレイピングしたすべてのアイテムを JSON/CSV/XML ファイルにダンプする最も簡単な方法は？

JSON ファイルにダンプするには:

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

CSV ファイルにダンプするには:

```
scrapy crawl myspider -o items.csv
```

XML ファイルにダンプするには:

```
scrapy crawl myspider -o items.xml
```

詳細は **フィード・エクスポート** を参照して下さい。

5.1.23 いくつかのフォームで使用されているこの巨大な __VIEWSTATE パラメーターは何ですか？

__VIEWSTATE パラメーターは、ASP.NET/VB.NET で構築されたサイトで使用されます。動作の詳細については、http://search.cpan.org/~ecarroll/HTML-TreeBuilderX-ASP_NET-0.09/lib/HTML/TreeBuilderX/ASP_NET.pm を参照してください。また、これらのサイトの 1 つをスクレイピングする example spider もあります。

5.1.24 大きな XML/CSV データ・フィードを解析する最良の方法は何ですか？

XPath セレクターを使用して大きなフィードを解析すると、フィード全体の DOM をメモリに構築する必要があるため問題が発生する可能性があります。これは非常に遅く、大量のメモリを消費する可能性があります。

メモリ内のフィード全体を一度に解析することを避けるために、 scrapy.utils.iterators モジュールの関数 xmliter と csviter を使用できます。 実際、これはフィード・スパイダー (**スパイダー** 参照) が内部で使用しているものです。

5.1.25 Scrapy はクッキーを自動的に管理しますか？

はい、Scrapy はサーバーから送信されたクッキーを受信して追跡し、通常の Web ブラウザーが行うように、後続のリクエストでそれらを送り返します。

詳細は **リクエストとレスポンス** と *CookiesMiddleware* を参照下さい。

5.1.26 Scrapy との間で送受信されている Cookie を確認するにはどうすればよいですか？

COOKIES_DEBUG 設定を有効にします。

5.1.27 スパイダーに自分自身を止めるように指示するにはどうすればよいですか？

コールバックから *CloseSpider* 例外を発生させます。 詳細については、 *CloseSpider* を参照してください。

5.1.28 Scrapy ボットがバン (BAN) されるのを防ぐにはどうすればよいですか？

バン *(*拒否*)* されるのを避ける 参照。

5.1.29 スパイダーを設定するには、スパイダーの引数または設定を使用する必要がありますか？

スパイダー引数 と **設定** の両方を使用して、スパイダーを設定できます。どちらか一方を使用することを義務付ける厳密なルールはありませんが、設定は一度設定するとあまり変化しないパラメーターに適しています。一方、スパイダーの引数はスパイダーの実行ごとに頻繁に変更されることを意図しており、スパイダーを実行するには (たとえば、スパイダーの開始 URL を設定するためなど、) どうせ必要になります。

例で説明するために、データをスクレイピングするためにサイトにログインする必要があるスパイダーがあり、(毎回異なる) サイトの特定のセクションからのみデータをスクレイピングしたいとします。 その場合、ログインする資格情報は設定になり、スクレイピングするセクションの URL はスパイダー引数になります。

5.1.30 XML ドキュメントをスクレイピングしていますが、XPath セレクターはアイテムを返しません

名前空間を削除する必要がある場合があります。 **名前空間 *(namespace)* の削除** 参照。

5.1.31 アイテム・パイプラインでアイテムを複数のアイテムに分割する方法は？

アイテム・パイプライン は、入力アイテムごとに複数のアイテムを生成できません。 代わりに **スパイダー・ミドルウェア** を作成し、この目的で *process_spider_output()* メソッドを使用します。例えば以下です:

```
from copy import deepcopy

from scrapy.item import BaseItem


class MultiplyItemsMiddleware:

    def process_spider_output(self, response, result, spider):
        for item in result:
            if isinstance(item, (BaseItem, dict)):
```

(次のページに続く)

(前のページからの続き)

```
                for _ in range(item['multiply_by']):
                    yield deepcopy(item)
```

5.2 スパイダーのデバッグ

この文書では、スパイダーをデバッグするための最も一般的な手法について説明します。 以下の Scrapy スパイダーについて考えます:

```
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):
        # <processing code not shown>
        # collect `item_urls`
        for item_url in item_urls:
            yield scrapy.Request(item_url, self.parse_item)

    def parse_item(self, response):
        # <processing code not shown>
        item = MyItem()
        # populate `item` fields
        # and extract item_details_url
        yield scrapy.Request(item_details_url, self.parse_details, cb_kwargs={'item':␣
↪item})

    def parse_details(self, response, item):
        # populate more `item` fields
        return item
```

基本的に、これは 2 ページのアイテム (start_urls) をパースする単純なスパイダーです。アイテムには追加情報のある詳細ページもあるため、 *Request* の *cb_kwargs*‘ 機能を使用して、部分的に入力されたアイテムを渡します。

5.2.1 parse コマンド

スパイダーの出力を確認する最も基本的な方法は、 *parse* コマンドを使用することです。メソッド・レベルでスパイダーのさまざまな部分の動作を確認できます。柔軟で使いやすいという利点がありますが、メソッド内のコー

ドをデバッグすることはできません。

特定の URL からスクレイピングされたアイテムを表示するには:

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

--verbose または -v オプションを使用すると、各深度レベルでステータスを確認できます:

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

単一の start_url からスクレイプされたアイテムのチェックも、以下を使用して簡単に実現できます:

```
$ scrapy parse --spider=myspider -d 3 'http://example.com/page1'
```

5.2.2 Scrapy シェル

parse コマンドはスパイダーの動作を確認するのに非常に役立ちますが、受信したレスポンスと出力を表示する以外の、コールバック内で何が起こるかを確認することにはほとんど役に立ちません。 では、 parse_details が時々アイテムを受け取らない状況をデバッグするには？

shell は、そういうあなたにピッタリのツールです (**スパイダーからシェルを呼び出してレスポンスを検査する** 参照):

```
from scrapy.shell import inspect_response

def parse_details(self, response, item=None):
    if item:
        # populate more `item` fields
        return item
    else:
        inspect_response(response, self)
```

スパイダーからシェルを呼び出してレスポンスを検査する も参照。

5.2.3 ブラウザを開く

あなたが特定のレスポンスがブラウザでどのように見えるかを確認したいだけの場合は、 `open_in_browser` 関数を使用できます。使用方法の例を次に示します:

```
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` は、その時点で Scrapy が受け取ったレスポンスでブラウザーを開き、画像とスタイルが適切に表示されるように base tag を調整します。

5.2.4 ロギング

ロギングは、スパイダーの実行に関する情報を取得するためのもう 1 つの便利なオプションです。それほど便利ではありませんが、ログが再び必要になった場合に、将来のすべての実行でログを使用できるという利点があります:

```
def parse_details(self, response, item=None):
    if item:
        # populate more `item` fields
        return item
    else:
        self.logger.warning('No item received for %s', response.url)
```

詳細については **ロギング** *(logging)* 節をチェックして下さい。

5.3 スパイダー コントラクト

バージョン 0.15 で追加.

注釈: これは新機能 (Scrapy 0.15 で導入) であり、細かい機能や API の更新が行われる場合があります。更新を知る為に **リリースノート** をチェックしてください。

スパイダーのテストは特にウンコで、単体テストを書くのは楽チンだけど、テスト作業はマンドクサ。Scrapy はコントラクト (訳注:contract; 取り決め) によってスパイダーを統一的にテストする方法を提供します。

これにより、サンプル URL をハードコーディングしてスパイダーの各コールバックをテストし、コールバックが応答を処理する方法のさまざまな制約を確認できます。各コントラクトは docstring に含め、各コントラクトの先頭には @ が付けられています。次の例をご覧ください:

```
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
    """
```

このコールバックは、3 つの組み込みコントラクトを使用してテストされます。:

class scrapy.contracts.default.**UrlContract**

このコントラクト (@url) は、このスパイダーの他のコントラクト条件をチェックするときに使用されるサンプル URL を設定します。 このコントラクトは必須です。 チェックを実行する場合、このコントラクトがないコールバックはすべて無視されます:

```
@url url
```

class scrapy.contracts.default.**CallbackKeywordArgumentsContract**

このコントラクト (@cb_kwargs) は、サンプルリクエストの *cb_kwargs* 属性を設定します。 有効な JSON 辞書である必要があります。:

```
@cb_kwargs {"arg1": "value1", "arg2": "value2", ...}
```

class scrapy.contracts.default.**ReturnsContract**

このコントラクト (@returns) は、スパイダーによって返されるアイテムとリクエストの下限と上限を設定します。 上限はオプションです。:

```
@returns item(s)|request(s) [min [max]]
```

class scrapy.contracts.default.**ScrapesContract**

このコントラクト (@scrapes) は、コールバックによって返されたすべてのアイテムに指定されたフィールドがあることを確認します。:

```
@scrapes field_1 field_2 ...
```

check コマンドを使用して、コントラクトチェックを実行します。

5.3.1 カスタム コントラクト

うぬはチカラが欲しくないか？ 組み込み Scrapy コントラクトよりも多くのチカラを。その場合は、*SPIDER_CONTRACTS* 設定を使用して、プロジェクトに独自のコントラクトを作成してロードできます:

```
SPIDER_CONTRACTS = {
    'myproject.contracts.ResponseCheck': 10,
    'myproject.contracts.ItemValidate': 10,
}
```

各コントラクトは *Contract* から継承する必要があり、3 つのメソッドをオーバーライドできます。:

class scrapy.contracts.**Contract**(*method*, **args*)

パラメータ

- **method** (*function*) – コントラクトが関連付けられているコールバック関数
- **args** (*list*) – docstring に渡される引数のリスト (空白区切り)

adjust_request_args(*args*)

これは、リクエストオブジェクトのデフォルト引数を含む引数として dict を受け取ります。 *Request* はデフォルトで使用されますが、これは request_cls 属性で変更できます。 チェーン内の複数のコントラクトにこの属性が定義されている場合、最後のコントラクトが使用されます。

同じまたは変更されたバージョンを返す必要があります。

pre_process(*response*)

これにより、コールバックに渡される前に、サンプルリクエストから受信したレスポンスのさまざまなチェックをフックすることを許します。

post_process(*output*)

これにより、コールバックの出力を処理できます。 イテレータは、このフックに渡される前に変換されてリスト化されます。

期待どおりで無い場合、 *pre_process* または *post_process* から ContractFail 例外が送出されます。

以下に、受信した応答のカスタムヘッダーの存在を確認するデモ・コントラクトを示します。:

```
from scrapy.contracts import Contract
from scrapy.exceptions import ContractFail

```

(次のページに続く)

(前のページからの続き)

```
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')
```

5.3.2 scrapy check 実行の検出

scrapy check が実行されているとき、 SCRAPY_CHECK 環境変数を true 文字列に設定します。 scrapy check が使用されている場合、 os.environ を使用して、スパイダーや設定に変更を加えることができます。:

```
import os
import scrapy

class ExampleSpider(scrapy.Spider):
    name = 'example'

    def __init__(self):
        if os.environ.get('SCRAPY_CHECK'):
            pass  # Do some scraper adjustments when a check is running
```

5.4 よくある例

このセクションでは、Scrapy を使用する際によくある例について説明します。 これらはいくつもトピックに渡るものであり、他の特定のトピックにはあまり該当しません。

5.4.1 スクリプトから Scrapy を実行する

あなたは scrapy crawl を介して Scrapy を実行する一般的な方法の代わりに、 *API* を使用してスクリプトから Scrapy を実行できます。

Scrapy は Twisted 非同期ネットワークライブラリの上に構築されているため、Twisted リアクター内で実行する必要があることに注意してください。

あなたがスパイダーを実行するために使用できる最初のユーティリティは scrapy.crawler.CrawlerProcess です。 このクラスは、Twisted リアクターを開始し、ロギングを構成し、シャットダウ

ン・ハンドラーを設定します。 このクラスは、すべての Scrapy コマンドで使用されるクラスです。

単一のスパイダーを実行する方法を示す例を以下に示します。

```
import scrapy
from scrapy.crawler import CrawlerProcess

class MySpider(scrapy.Spider):
    # Your spider definition
    ...

process = CrawlerProcess(settings={
    'FEED_FORMAT': 'json',
    'FEED_URI': 'items.json'
})

process.crawl(MySpider)
process.start() # the script will block here until the crawling is finished
```

CrawlerProcess の辞書内の設定を定義します。 CrawlerProcess の文書を確認して、使用方法の詳細を把握してください。

あなたが Scrapy プロジェクト内にいる場合は、プロジェクト内にこれらのコンポーネントをインポートするために使用できる追加のヘルパーがいくつかあります。名前を CrawlerProcess に渡してスパイダーを自動的にインポートし、 get_project_settings を使用してプロジェクト設定で Settings インスタンスを取得できます。

以下は、例として testspiders プロジェクトを使用して、それを行う方法の実際の例です。

```
from scrapy.crawler import CrawlerProcess
from scrapy.utils.project import get_project_settings

process = CrawlerProcess(get_project_settings())

# 'followall' is the name of one of the spiders of the project.
process.crawl('followall', domain='scrapinghub.com')
process.start() # the script will block here until the crawling is finished
```

クロール・プロセスをより詳細に制御する別の Scrapy ユーティリティがあります。それは scrapy.crawler.CrawlerRunner です。このクラスは、複数のクローラーを実行するための単純なヘルパーをカプセル化する薄いラッパーですが、既存のリアクターを開始したり、それに干渉したりすることはありません。

このクラスを使用すると、スパイダーをスケジュールした後にリアクターを明示的に実行する必要があります。アプリケーションがすでに Twisted を使用しており、同じリアクターで Scrapy を実行する場合は、CrawlerProcess の代わりに CrawlerRunner を使用することをお勧めします。

スパイダーが終了した後、自分で Twisted リアクターをシャットダウンする必要があることに注意してください。これは、 CrawlerRunner.crawl メソッドによって返される遅延オブジェクトにコールバックを追加すること

で実現できます。

以下に使用例と、 MySpider の実行が終了した後に手動でリアクターを停止するコールバックを示します。

```
from twisted.internet import reactor
import scrapy
from scrapy.crawler import CrawlerRunner
from scrapy.utils.log import configure_logging

class MySpider(scrapy.Spider):
    # Your spider definition
    ...

configure_logging({'LOG_FORMAT': '%(levelname)s: %(message)s'})
runner = CrawlerRunner()

d = runner.crawl(MySpider)
d.addBoth(lambda _: reactor.stop())
reactor.run() # the script will block here until the crawling is finished
```

参考:

Twisted Reactor Overview.

5.4.2 同じプロセスで複数のスパイダーを実行する

デフォルトでは、あなたが `scrapy crawl` を実行すると、Scrapy はプロセスごとに 1 つのスパイダーを実行します。ただし、Scrapy は、**内部** *API* を使用して、プロセスごとに複数のスパイダーを実行することをサポートしています。

複数のスパイダーを同時に実行する例を次に示します:

```
import scrapy
from scrapy.crawler import CrawlerProcess

class MySpider1(scrapy.Spider):
    # Your first spider definition
    ...

class MySpider2(scrapy.Spider):
    # Your second spider definition
    ...

process = CrawlerProcess()
process.crawl(MySpider1)
process.crawl(MySpider2)
process.start() # the script will block here until all crawling jobs are finished
```

CrawlerRunner を使用した同じ例:

```
import scrapy
from twisted.internet import reactor
from scrapy.crawler import CrawlerRunner
from scrapy.utils.log import configure_logging

class MySpider1(scrapy.Spider):
    # Your first spider definition
    ...

class MySpider2(scrapy.Spider):
    # Your second spider definition
    ...

configure_logging()
runner = CrawlerRunner()
runner.crawl(MySpider1)
runner.crawl(MySpider2)
d = runner.join()
d.addBoth(lambda _: reactor.stop())

reactor.run() # the script will block here until all crawling jobs are finished
```

同じ例ですが、遅延オブジェクトを連鎖させてスパイダーを順番に実行します:

```
from twisted.internet import reactor, defer
from scrapy.crawler import CrawlerRunner
from scrapy.utils.log import configure_logging

class MySpider1(scrapy.Spider):
    # Your first spider definition
    ...

class MySpider2(scrapy.Spider):
    # Your second spider definition
    ...

configure_logging()
runner = CrawlerRunner()

@defer.inlineCallbacks
def crawl():
    yield runner.crawl(MySpider1)
    yield runner.crawl(MySpider2)
    reactor.stop()

crawl()
reactor.run() # the script will block here until the last crawl call is finished
```

参考:

スクリプトから *Scrapy* を実行する

5.4.3 分散クロール

Scrapy は、分散 (マルチサーバー) 方式でクロールを実行するための組み込み機能を提供しません。ただし、クロールを配布する方法はいくつかあり、それらは配布方法によって異なります。

多数のスパイダーがある場合、負荷を分散する明白な方法は、多くの Scrapyd インスタンスをセットアップし、それらの間でスパイダー実行を分散することです。

代わりに、多くのマシンで単一の (大きな) スパイダーを実行したい場合、通常行うことは、クロールする URL をパーティション分割して、各スパイダーに送信します。 具体例を次に示します:

最初に、クロールする URL のリストを準備し、それらを個別のファイル/URL に入れます:

```
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
```

次に、3 つの異なる Scrapyd サーバーで実行されるスパイダーを起動します。スパイダーはクロールするパーティションの番号を含む (スパイダー) 引数 `part` を受け取ります:

```
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
```

5.4.4 バン (拒否) されるのを避ける

一部の Web サイトでは、高度なレベルのさまざまなボットによるクロールを防止するための特定の手段を実装しています。これらの対策を回避することは難しい場合があり、特別なインフラストラクチャが必要になる場合があります。疑問がある場合は、(有償の) 商用サポート (commercial support) に連絡することを検討してください。

これらの種類のサイトを扱う際に留意するべきいくつかのヒントは以下にあります:

- ブラウザから取得したよく使われているユーザーエージェントのプールを使ってユーザーエージェントをローテーションします (google でそれらのリストを取得します)
- クッキーを無効にします (*COOKIES_ENABLED* 参照)
- ダウンロード遅延 (2 以上) を使用します。 *DOWNLOAD_DELAY* 設定を参照してください。

- 可能であれば、サイトに直接アクセスするのではなく、Google キャッシュ (Google cache) を使用してページを取得します
- ローテート IP のプールを使用します。 たとえば、無料の Tor プロジェクト (Tor project) または ProxyMesh のような有料サービスです。オープンソースの代替手段は、 scrapoxy です。これは、独自のプロキシをアタッチできるスーパープロキシです。
- 内部的に禁止を回避する高度に分散されたダウンローダーを使用することで、クリーンなページのパースに集中できます。そのようなダウンローダーの一例は Crawlera です

それでもボットが禁止されるのを防ぐことができない場合は、(有償の) 商用サポート (commercial support) に連絡することを検討してください。

5.5 広範なクロール

Scrapy のデフォルトは、特定のサイトをクロールするために最適化されています。 多くの場合、これらのサイトは単一の Scrapy スパイダーで処理されますが、これは必要または必須ではありません (たとえば、スロー (throw) される特定のサイトを処理する汎用スパイダーがあります)。

この「フォーカスクロール」に加えて、多数の（潜在的に無制限の）ドメインをカバーする別の一般的なタイプのクロールがあり、ドメインが完全にクロールされたときまたは、実行するリクエストがなくなったときに停止するのではなく、時間またはその他の任意の制約によってのみ制限されます。これらは広範なクロール (broad crawls) と呼ばれ、検索エンジンで採用されている典型的なクローラーです。

これらは、広範なクロールでよく見られる一般的なプロパティです:

- 特定のサイトセットではなく、多くのドメイン（多くの場合、無制限）をクロールします。
- ドメインをクロールする必要はありません。実行するのは非現実的（または不可能）であるため、代わりにクロールを時間またはクロールされるページ数で制限します。
- (多くの抽出ルールを持つ非常に複雑なスパイダーとは対照的に、) 多くの場合、データは別の段階で後処理されることが多いため、ロジックが単純です。
- 多くのドメインを並列してクロールします。これにより、特定のサイトの制約によって制限されないため、クロール速度が向上します (各サイトはポライトネスを尊重するためにゆっくりクロールされますが、多くのサイトが並行してクロールされます)

前述のように、Scrapy のデフォルト設定は、広範なクロールではなく、集中的なクロール用に最適化されています。ただし、非同期アーキテクチャのため、Scrapy は高速で広範なクロールを実行するのに非常に適しています。ここでは、Scrapy を使用して幅広いクロールを行う際に留意する必要があるいくつかの事項を要約し、効率的な幅広いクロールを実現するために調整する Scrapy 設定の具体的な提案を行います。

5.5.1 正しい SCHEDULER_PRIORITY_QUEUE の使用

Scrapy のデフォルトのスケジューラ優先度キューは 'scrapy.pqueues.ScrapyPriorityQueue' です。単一ドメインクロール中に最適に機能します。多くの異なるドメインを並行してクロールするとうまく機能しません

推奨される優先度キューを適用するには:

```
SCHEDULER_PRIORITY_QUEUE = 'scrapy.pqueues.DownloaderAwarePriorityQueue'
```

5.5.2 並行性を高める

同時実行性は、並行して処理されるリクエストの数です。グローバル制限 (*CONCURRENT_REQUESTS*) と、ドメイン (*CONCURRENT_REQUESTS_PER_DOMAIN*) または IP(*CONCURRENT_REQUESTS_PER_IP*) ごとに設定できる追加の制限があります。

注釈: スケジューラの優先度キューの **広範なクロールにお勧めのキュー** は *CONCURRENT_REQUESTS_PER_IP* をサポートしていません。

Scrapy のデフォルトのグローバル同時実行制限は、多くの異なるドメインを並行してクロールするのには適していないため、増やすことをお勧めします。どれだけ増やすかは、クローラーが使用できる CPU とメモリの量によって異なります。

良い出発点は 100 です:

```
CONCURRENT_REQUESTS = 100
```

しかし、それを見つけるための最良の方法は、いくつかの試行を行い、Scrapy プロセスが CPU の限界に達する同時性を特定することです。最適なパフォーマンスを得るには、CPU 使用率が 80〜90 %の同時実行性を選択する必要があります。

並行性を高めると、メモリ使用量も増えます。メモリ使用量が懸念される場合は、それに応じてグローバル同時実行制限を下げる必要があります。

5.5.3 Twisted IO スレッド・プールの最大サイズを増やす

現在、Scrapy は、スレッド・プールの使用をブロックする方法で DNS 解決を行います。 同時実行レベルが高いと、クロールが遅くなったり、DNS リゾルバーのタイムアウトに失敗したりすることさえあります。可能な解決策は DNS クエリを処理するスレッドの数を増やす事です。そうすれば、DNS キューはより速く処理され、接続の確立とクロール全体が高速化されます。

スレッド・プールの最大サイズを増やすには:

```
REACTOR_THREADPOOL_MAXSIZE = 20
```

5.5.4 あなた独自の DNS をセットアップする

複数のクロール・プロセスと単一の中央 DNS がある場合、DNS サーバーに対する DoS 攻撃のように動作し、ネットワーク全体の速度を低下させたり、マシンをブロックすることさえあります。この設定を回避するには、ローカルキャッシュを備えた独自の DNS サーバーと、OpenDNS や Verizon などの大規模な DNS のアップ・ストリームを使用します。

5.5.5 ログレベルを下げる

広範なクロールを行う場合、多くの場合、取得するクロール・レートと検出されたエラーのみに関心があります。これらの統計は、 INFO ログ・レベルを使用しているときに Scrapy によって報告されます。 CPU 資源 (およびログ・ストレージ要件) を節約するために、本番環境で大規模な広範なクロールを実行するときに DEBUG ログ・レベルを使用しないでください。(広範囲の) クローラーの開発時では DEBUG レベルを使用しても問題ありません。

ログレベルを設定するには:

```
LOG_LEVEL = 'INFO'
```

5.5.6 クッキーを無効にする

本当に 必要がない限り、クッキーを無効にします。クッキーは、広範なクロールを実行するときに必要ない場合が多く (検索エンジンク・ローラーはそれらを無視します)、CPU サイクルを節約し、Scrapy クローラーのメモリ・フット・プリントを削減することでパフォーマンスを向上させます。

クッキーを無効にするには:

```
COOKIES_ENABLED = False
```

5.5.7 再試行を無効にする

失敗した HTTP リクエストを再試行すると、特にサイトが原因でレスポンスが非常に遅い (または失敗する) 場合、クロールが大幅に遅くなる可能性があります。そのため、タイム・アウト・エラーが何度も再試行され、不必要にクローラー容量が他のドメインで再利用できなくなります。

再試行を無効にするには:

```
RETRY_ENABLED = False
```

5.5.8 ダウンロードのタイムアウトを短縮

非常に遅い接続からクロールする場合を除き (広範なクロールの場合はそうではありません)、ダウンロード・タイムアウトを減らして、スタックしたリクエストを迅速に破棄し、次のリクエストを処理するための容量を解放します。

ダウンロードのタイムアウトを短縮するには:

```
DOWNLOAD_TIMEOUT = 15
```

5.5.9 リダイレクトを無効にする

リダイレクトの追跡に興味がない限り、リダイレクトを無効にすることを検討してください。広範なクロールを実行する場合、リダイレクトを保存し、後のクロールでサイトを再訪するときにリダイレクトを解決するのが一般的です。これは、クロール・バッチごとにリクエストの数を一定に保つのにも役立ちます。そうしないと、リダイレクト・ループが原因で、クローラーが特定のドメインで多くのリソースを占有する可能性があります。

リダイレクトを無効にするには:

```
REDIRECT_ENABLED = False
```

5.5.10 「Ajax クロール可能なページ」のクロールを有効にします

一部のページ (2013 年の経験データに基づいて最大 1 %) は、自身を ajax クロール可能 (ajax crawlable) と宣言しています。 これは、通常 AJAX 経由でのみ利用可能なコンテンツのプレーン HTML バージョンを提供することを意味します。ページは 2 つの方法でそれを示すことができます:

1) URL で `#!` を使用する - これがデフォルトの方法です;

2) 特殊なメタ・タグを使用する - この方法は、"main"、"index"ウェブサイトページで使用されます。

Scrapy は (1) を自動的に処理します。そして、 (2) を処理するには、 *AjaxCrawlMiddleware* を有効にします:

```
AJAXCRAWL_ENABLED = True
```

広範なクロールを行う場合、多くの「インデックス」Web ページをクロールするのが一般的です。 AjaxCrawlMiddleware は、それらを正しくクロールするのに役立ちます。 パフォーマンスのオーバーヘッドがあるため、デフォルトではオフになっています。また、集中的なクロール (focused crawls) を有効にすることはあまり意味がありません。

5.5.11 BFO 順でクロールする

Scrapy **クロールのデフォルトは** *DFO* **順**.

ただし、広範なクロールでは、ページのクロールはページの処理よりも高速になる傾向があります。その結果、未処理の初期リクエストは最終的な深さに達するまでメモリ内に留まり、メモリ使用量が大幅に増加する可能性があります。

メモリを節約するために、代わりに、 *BFO* **順でクロール** する。

5.5.12 メモリリークに気を配る

BFO **の順序でのクロール** および **並行処理の低下** に加えて、広範なクロールのメモリ使用量が多い場合、 あなたは **メモリリークのデバッグ** をするべきです。

5.6 Web ブラウザの開発ツールを使ってスクレイピングする

ここでは、ブラウザの開発ツールを使用してスクレイピング・プロセスを簡単にする方法に関する一般的なガイドを示します。 今日、ほとんどすべてのブラウザには Developer Tools が組み込まれています。このガイドでは Firefox を使用しますが、概念は他のブラウザにも適用できます。

このガイドでは、 quotes.toscrape.com をスクレイピングすることにより、ブラウザーの開発ツールから使用する基本的なツールを紹介します。

5.6.1 ライブ・ブラウザ DOM の検査に関する注意事項

開発ツールはライブ・ブラウザ DOM で動作するため、ページ・ソースの検査時に実際に表示されるのは元の HTML ではなく、ブラウザのクリーンアップを適用して Javascript コードを実行した後の変更された HTML です。特に、Firefox は `<tbody>` 要素をテーブルに追加することで知られています。一方、Scrapy は元のページの HTML を変更しないため、XPath 式で `<tbody>` を使用すると、データを抽出できません。

したがって、あなたは次のことに注意してください:

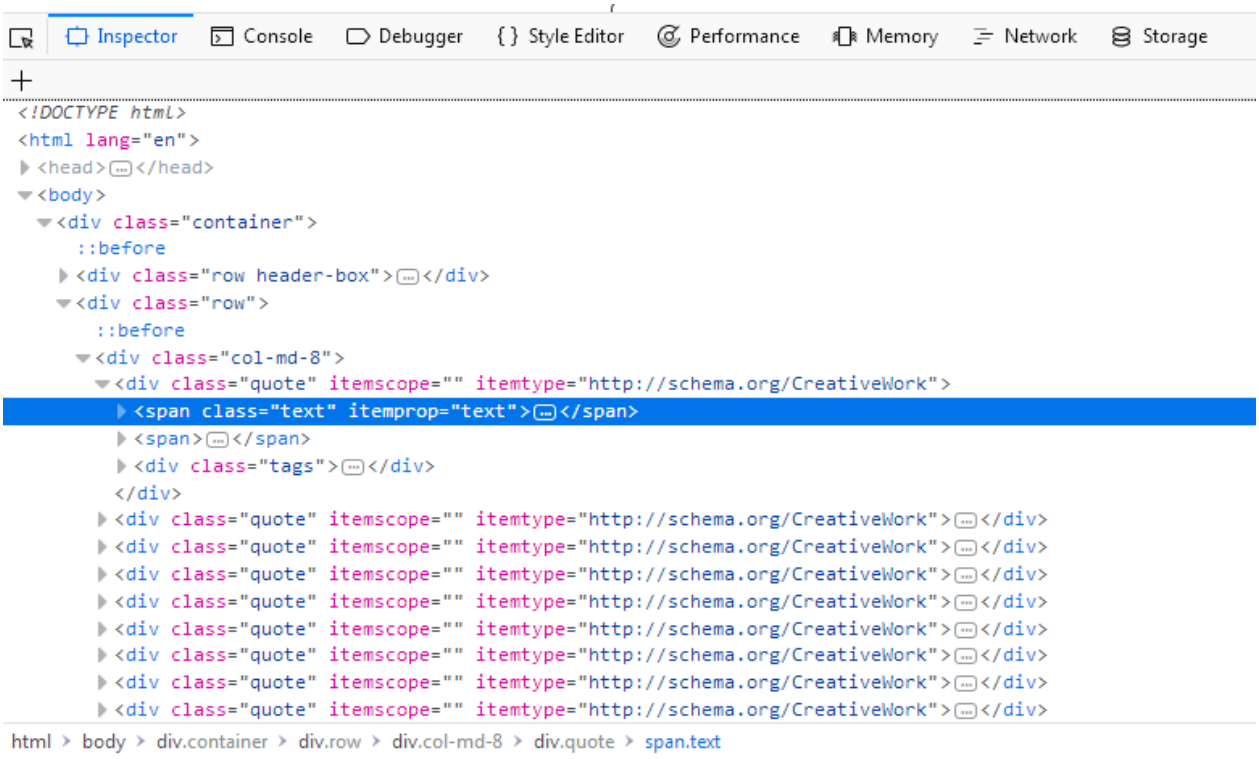
- Scrapy で使用される XPath を探す DOM の検査中に JavaScript を無効にします (開発ツールの設定で「JavaScript を無効化」をクリックします)
- フルパスの XPath 式を使用せず、(`id`、 `class`、 `width` などのような) 属性または `contains(@href, 'image')` のような識別機能に基づいた相対的で賢いパスを使用してください。
- あなたが何をしているのか本当に理解していない限り、あなたの XPath 式に `<tbody>` 要素を含めないでください。

5.6.2 ウェブサイトの検査

開発ツールの最も便利な機能はインスペクター (*Inspector*) 機能です。これにより、Web ページの基になる HTML コードを検査できます。インスペクターをデモンストレーションするために、 quotes.toscrape.com -site を見てみましょう。

このサイトには、特定のタグを含むさまざまな著者からの合計 10 個の引用と、トップ 10 のタグがあります。 著者、タグなどに関するメタ情報なしで、このページのすべての引用を抽出したいとしましょう。

ページのソースコード全体を表示する代わりに、引用を右クリックして「要素の検査 (Q)」を選択するだけで、インスペクターが開きます。その中に次のようなものが見えるはずです:



私たちにとって興味深い部分はこれです:

```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
  <span class="text" itemprop="text">(...)</span>
  <span>(...)</span>
  <div class="tags">(...)</div>
</div>
```

スクリーンショットで強調表示された span タグのすぐ上の最初の div にカーソルを合わせると、Web ページの対応するセクションも強調表示されます。 これで私たちはセクションを得ましたが、引用テキストはどこにも見つかりません。

インスペクタの利点は、Web ページのセクションとタグを自動的に展開および折りたたむことであり、読みやすさ

が大幅に向上します。タグの前にある矢印をクリックするか、タグを直接ダブルクリックして、タグを展開したり折りたたんだりできます。span タグを class= "text" で展開すると、クリックした引用テキストが表示されます。インスペクタを使用すると、選択した要素に XPath をコピーできます。試してみましょう。 span タグを右クリックして **コピー** →XPath を選択し、Scrapy シェルに貼り付けます:

```
$ scrapy shell "http://quotes.toscrape.com/"
(...)
>>> response.xpath('/html/body/div/div[2]/div[1]/div[1]/span[1]/text()').getall()
['"The world as we have created it is a process of our thinking. It cannot be changed␣
↪without changing our thinking."]
```

最後に text() を追加すると、この基本セレクターで最初の引用を抽出できます。しかし、この XPath はあまり賢くありません。それは、ソースコードの html から始まる望ましいパスをたどるだけです。それでは、この XPath を少し改良できるかどうか見てみましょう:

私たちがインスペクターを再びチェックすると、展開された div タグの下に、それぞれが最初の属性と同じ属性を持つ 9 つの同一の div タグがあることがわかります。それらのいずれかを展開すると、最初の引用と同じ構造が表示されます。2 つの span タグと 1 つの div タグです。 div タグ内の class="text" で各 span タグを展開し、各引用を見ることができます:

```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
  <span class="text" itemprop="text">
    “The world as we have created it is a process of our thinking. It cannot be␣
↪changed without changing our thinking.”
  </span>
  <span>(...)</span>
  <div class="tags">(...)</div>
</div>
```

この知識があれば、私たちの XPath を改良できます。たどるパスの代わりに、 has-class-extension を使用して class="text" を持つすべての span タグを選択するだけです:

```
 >>> response.xpath('//span[has-class("text")]/text()').getall()
['"The world as we have created it is a process of our thinking. It cannot be changed␣
↪without changing our thinking.",
 '“It is our choices, Harry, that show what we truly are, far more than our abilities.”
',
 '“There are only two ways to live your life. One is as though nothing is a miracle.␣
↪The other is as though everything is a miracle."',
 (...)]
```

そして、1 つの単純で賢い XPath を使用して、ページからすべての引用を抽出できます。 最初の XPath にループを構築して最後の div の数を増やすこともできましたが、これは不必要に複雑であり、単に has-class("text") で XPath を構築することで、すべての引用を 1 行で抽出できました。

インスペクターには、ソースコードの検索や選択した要素への直接スクロールなど、他の便利な機能がたくさんあ

ります。以下にユースケースを示しましょう:

あなたはページの Next ボタンを見つけたいとします。インスペクターの右上にある検索バーに Next と入力します。2 つの結果が得られます。 最初は class="text" を持つ li タグで、2 つ目は a タグのテキストです。 a タグを右クリックして、 **この要素の位置にスクロール** (S) を選択します。ここから **リンク抽出器** を簡単に作成して、ページネーションを追跡できます。 このようなシンプルなサイトでは、要素を視覚的に見つける必要はないかもしれませんが、複雑なサイトでは **この要素の位置にスクロール** (S) 機能は非常に便利です。

検索バーは、CSS セレクターの検索とテストにも使用できることに注意してください。 たとえば、 span.text を検索して、すべての引用テキストを見つけることができます。全文検索の代わりに、これはページ内で class="text" を持つ span タグを正確に検索します。

5.6.3 ネットワーク・ツール

スクレイピング中に、ページの一部が複数のリクエストを介して動的にロードされる動的 Web ページに遭遇する場合があります。これは非常に難しい場合がありますが、開発ツールの「ネットワーク」ツールはこのタスクを非常に容易にします。ネットワーク・ツールをデモンストレーションするために、ページ quotes.toscrape.com/scroll を見てみましょう。

このページは基本的な quotes.toscrape.com -page に非常に似ていますが、上記の Next ボタンの代わりに、ページを下にスクロールすると自動的に新しい引用を読み込みます。先に進んでさまざまな XPath を直接試すこともできますが、代わりに Scrapy シェルから別の非常に便利なコマンドをチェックします:

```
$ scrapy shell "quotes.toscrape.com/scroll"
(...)
>>> view(response)
```

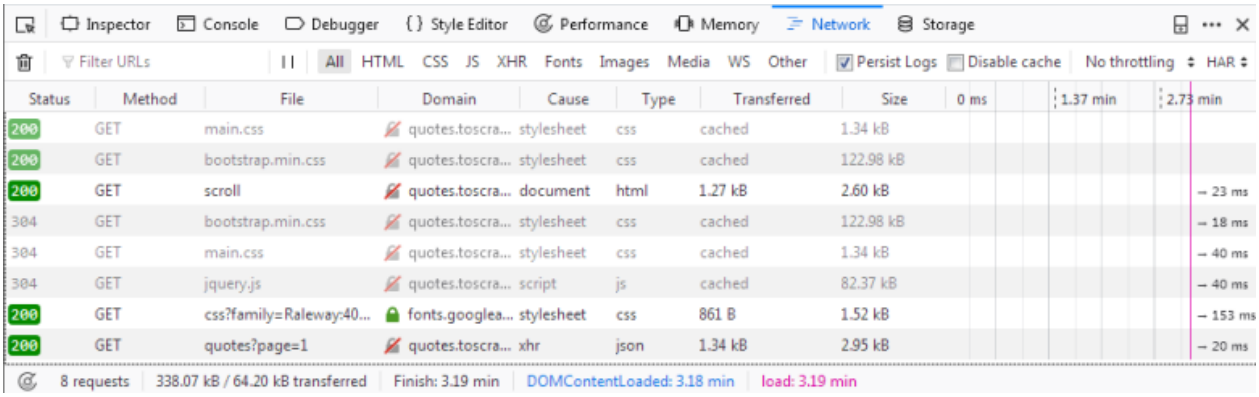
ブラウザーウィンドウは Web ページで開きますが、1 つの重要な違いがあります。引用の代わりに、Loading... という語が付いた緑がかったバーが表示されるだけです。

view(response) コマンドにより、シェルまたは後でスパイダーがサーバーから受信するレスポンスを表示で

きます。ここでは、タイトル、ログイン・ボタン、フッターを含むいくつかの基本的なテンプレートが読み込まれていますが、引用が欠落しています。これは、引用が `quotes.toscrape/scroll` とは異なるリクエストからロードされていることを示しています。

「ネットワーク」タブをクリックすると、おそらく 2 つのエントリしか表示されません。最初に行うことは、「永続ログ」をクリックして永続ログを有効にすることです。このオプションを無効にすると、別のページに移動するたびにログが自動的にクリアされます。ログをクリアするタイミングを制御できるため、このオプションを有効にするのは良い怠慢です。

ここでページをリロードすると、ログに 6 つの新しいリクエストが表示されます。

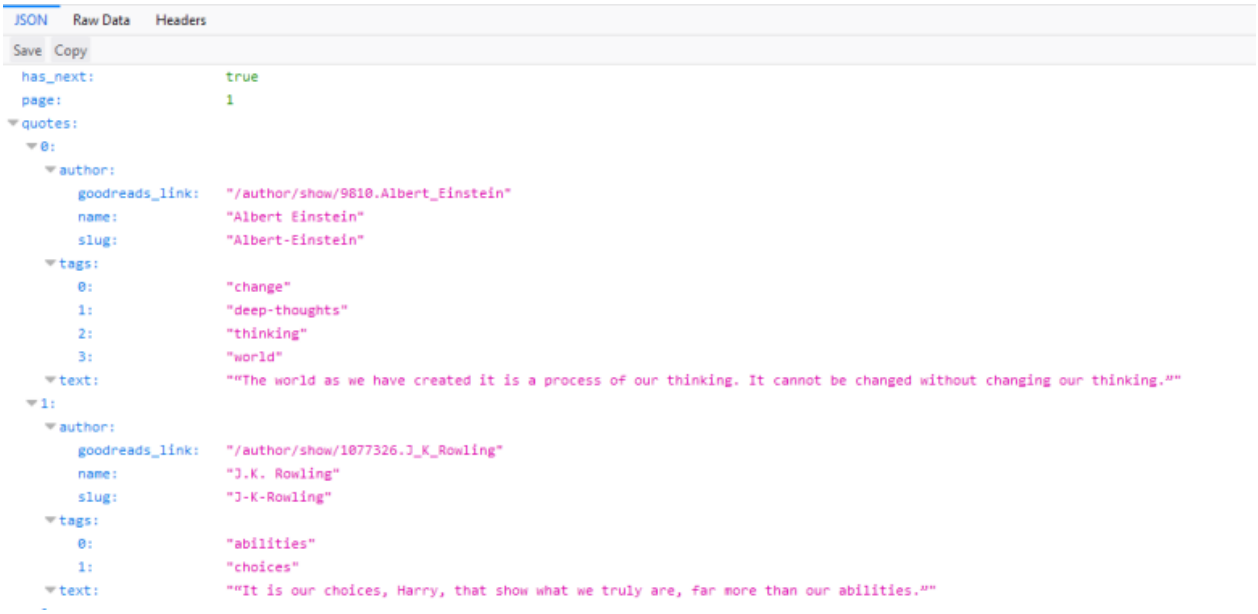


ここでは、ページのリロード時に行われたすべてのリクエストが表示され、各リクエストとそのレスポンスを検査できます。それでは、私たちの引用がどこから来ているのかを見てみましょう:

最初に `scroll` という名前のリクエストをクリックします。 開発ツール画面の右側で、リクエストを検査できます。ヘッダー・タブには、URL、メソッド、IP アドレスなどのリクエスト・ヘッダーに関する詳細があります。 他のタブは無視し、応答タブをクリックします。

プレビュー・ペインに表示されるのはレンダリングされた HTML コードです。これは、シェルで `view(response)` を呼び出したときに見たものです。 したがって、ログ内のリクエストの `type` は `html` です。 他のリクエストには `css` や `js` などのタイプがありますが、興味深いのは、タイプが `json` の `quotes?page=1` と呼ばれるリクエストです。

このリクエストをクリックすると、リクエスト URL が `http://quotes.toscrape.com/api/quotes?page=1` であり、レスポンスが引用を含む JSON オブジェクトであることがわかります。 また、リクエストを右クリックして「新しいタブで開く」(`Open in new tab`) を開いて、概要を確認することもできます。



このレスポンスにより、JSON オブジェクトを簡単にパースし、各ページにサイト上のすべての引用を取得するようリクエストすることができます。

```
import scrapy
import json


class QuoteSpider(scrapy.Spider):
    name = 'quote'
    allowed_domains = ['quotes.toscrape.com']
    page = 1
    start_urls = ['http://quotes.toscrape.com/api/quotes?page=1']

    def parse(self, response):
        data = json.loads(response.text)
        for quote in data["quotes"]:
            yield {"quote": quote["text"]}
        if data["has_next"]:
            self.page += 1
            url = "http://quotes.toscrape.com/api/quotes?page={}".format(self.page)
            yield scrapy.Request(url=url, callback=self.parse)
```

このスパイダーは quotes-API の最初のページから始まります。各レスポンスで、 `response.text` をパースし、 `data` に割り当てます。 これにより、Python 辞書のように JSON オブジェクトを操作できます。 `quotes` を繰り返し処理し、 `quote["text"]` を出力します。 便利な `has_next` 要素が `true` の場合 (ブラウザーに quotes.toscrape.com/api/quotes?page=10 または 10 より大きいページ番号をロードしてみてください)、 `page` 属性と新しいリクエストを生成 (`yield`) し、インクリメントしたページ番号を `url` に挿入します。

より複雑な Web サイトでは、リクエストを簡単に再現するのが難しい場合があります。リクエストを機能させるにはヘッダーまたはクッキーを追加する必要があるからです。これらの場合、ネットワーク・ツールで各リクエス

トを右クリックし、 `from_curl()` メソッドを使用して同等のリクエストを生成することにより、リクエストを cURL 形式でエクスポートできます。

```
from scrapy import Request

request = Request.from_curl(
    "curl 'http://quotes.toscrape.com/api/quotes?page=1' -H 'User-Agent: Mozil"
    "la/5.0 (X11; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0' -H 'Acce"
    "pt: */*' -H 'Accept-Language: ca,en-US;q=0.7,en;q=0.3' --compressed -H 'X"
    "-Requested-With: XMLHttpRequest' -H 'Proxy-Authorization: Basic QFRLLTAzM"
    "zEwZTAxLTk5MWUtNDFiNC1iZWRmLTJjNGI4M2ZiNDBmNDpAVEstMDMzMTBlMDEtOTkxZS00MW"
    "I0LWJlZGYtMmM0YjgzZmI0MGY0' -H 'Connection: keep-alive' -H 'Referer: http"
    "://quotes.toscrape.com/scroll' -H 'Cache-Control: max-age=0'")
```

あるいは、そのリクエストを再作成するために必要な引数を知りたい場合、 `scrapy.utils.curl.curl_to_request_kwargs()` 関数を使用して同等の引数を持つ辞書を取得できます。

ご覧のとおり、ネットワーク・ツールでいくつかの検査を行うことで、ページのスクロール機能の動的リクエストを簡単に複製できました。 動的ページのクロールは非常に困難な場合があり、ページは非常に複雑になる可能性がありますが、最終的には正しいリクエストを識別し、それをスパイダーで複製することになります。

5.7 動的に読み込まれたコンテンツの選択

一部の Web ページは、あなたがそれらのページを Web ブラウザーに読み込み後に目的のデータを表示します。 ただし、Scrapy を使用してそれらをダウンロードする場合、 *selectors* を使用して目的のデータに到達することはできません。

この場合、推奨されるアプローチは **データ・ソースを探し、** そこからデータを抽出することです。

データ・ソースを探すのに失敗し、それでも Web ブラウザから *DOM* を介して目的のデータにアクセスできる場合は、 *JavaScript* **の事前レンダリング** を参照してください。

5.7.1 データ・ソースを探す

目的のデータを抽出するには、最初にソースの場所を見つける必要があります。

データが画像や PDF ドキュメントなどの非テキストベースの形式である場合、あなたの Web ブラウザの **ネットワークツール** を使用して、対応するリクエストを見つけ、 **それを再現します。**

Web ブラウザーで目的のデータをテキストとして選択できる場合、データは埋め込み JavaScript コードで定義されるか、テキストベースの形式で外部リソースからロードされます。

その場合、wgrep などのツールを使用して、そのリソースの URL を見つけることができます。

データが元の URL 自体からのものであることが判明した場合、 *Web* **ページのソースコードを調べて、** データの場所を特定する必要があります。

データが別の URL からのものである場合は、 **対応するリクエストを再現する必要があります。**

5.7.2 Web ページのソースコードの調査

しばしば、(*DOM* ではなく)Web ページのソースコードを調べて、必要なデータがどこにあるかを判断する必要がある場合があります。

Scrapy が見れる Web ページのコンテンツをダウンロードするために Scrapy の `fetch` コマンドを使用しください。:

```
scrapy fetch --nolog https://example.com > response.html
```

目的のデータが <script/> 要素内の埋め込み JavaScript コードにある場合は、 *JavaScript* **コードのパース** を参照してください。

目的のデータが見つからない場合は、まず、Scrapy だけで見つからないのではないことを確認します。curl や wget などの HTTP クライアントでも Web ページをダウンロードしてみて、取得したレスポンスで情報が見つかるかどうかを確認します。

他の HTTP クライアントで目的のデータがあるレスポンスを受け取った場合、その HTTP クライアントと同じになるようにあなたの Scrapy の `Request` を変更します。 たとえば、同じユーザーエージェント文字列 (`USER_AGENT`) または同じ `headers` にします。

目的のデータなしのレスポンスが返される場合は、リクエストをウェブブラウザのリクエストにより近いものにするための手順を実行する必要があります。 **リクエストの再現** を参照してください。

5.7.3 リクエストの再現

しばしば、Web ブラウザが実行する方法でリクエストを再現する必要がある場合があります。

Web ブラウザーの **ネットワーク・ツール** を使用して、Web ブラウザーが目的のリクエストをどのように実行するかを確認し、Scrapy でそのリクエストを再現してください。

同じ HTTP メソッドと URL で `Request` を生成すれば十分かもしれません。ただし、そのリクエストの本文、ヘッダー、フォームパラメーター (`FormRequest` 参照) を再現する必要がある場合もあります。

すべての主要なブラウザはリクエストを cURL 形式でエクスポートできるため、Scrapy は `from_curl()` メソッドを組み込んで、cURL コマンドから同等の `Request` を生成します。 詳細については、ネットワークツール節内の *request from curl* を参照してください。

あなたが期待するレスポンスを取得したら、あなたは **目的のデータをそこから抽出する事** ができます。

Scrapy でリクエストを再現できます。 けれども、必要なすべてのリクエストを再現することは、開発時には効率的でないように見える場合があります。 もしあなたがそのような場合に遭遇し、そして、クロール速度があなたにとって大きな関心事ではない場合は、代わりに *JavaScript* **の事前レンダリング** を検討することもできます。

予想されるレスポンスが時々は得られるが、常にではない場合、問題はおそらくリクエストではなく、ターゲットサーバーにあります。 ターゲットサーバーはバグがあるか、過負荷であるか、または **禁止** リクエストの一部です。

5.7.4 さまざまなレスポンス形式の処理

あなたが目的のデータを含むレスポンスを取得した後、そこから目的のデータを抽出する方法は、レスポンスのタイプによって異なります。

- レスポンスが HTML または XML の場合、通常どおり **セレクター** を使用します。
- 応答が JSON の場合、 json.loads を使用して `response.text` から目的のデータをロードします。:

```
data = json.loads(response.text)
```

  目的のデータが JSON データに埋め込まれた HTML または XML コード内にある場合、その HTML または XML コードを `Selector` にロードして、それから、いつものように **セレクター** を使います。

```
selector = Selector(data['html'])
```

- 応答が JavaScript、または目的のデータを含む `<script/>` 要素を持つ HTML の場合、 *JavaScript* **コードのパース** を参照してください。
- レスポンスが CSS の場合、正規表現 (regular expression) を使用して `response.text` から目的のデータを抽出します。
- レスポンスが画像または画像に基づく別の形式 (PDF など)）の場合、レスポンスを **レスポンス・ボディ** からバイトとして読み取り、OCR ソリューションを使用してテキストとして目的のデータを抽出します。

  たとえば、 pytesseract を使用できます。 PDF から表を読むには、 tabula-py がより良い選択かもしれません。

- レスポンスが SVG、または目的のデータを含む埋め込み SVG を含む HTML の場合、SVG は XML に基づいているため、 **セレクター** を使用して目的のデータを抽出できます。

  そうでない場合は、SVG コードをラスターイメージに変換し、 **ラスターイメージ処理** を行う必要があります。

5.7.5 JavaScript コードのパース

目的のデータが JavaScript でハードコーディングされている場合、最初に JavaScript コードを取得する必要があります。:

- JavaScript コードが JavaScript ファイルにある場合は、単に *response.text* から読み取ります。
- JavaScript コードが HTML ページの <script/> 要素内にある場合、 **セレクター** を使用して、その <script/> 要素内のテキストを抽出します。

あなたが JavaScript コードを含む文字列を取得したら、そこから目的のデータを抽出できます。:

- 正規表現 (regular expression) を使用して JSON 形式で目的のデータを抽出し、 json.loads でパースできる場合があります。

  たとえば、JavaScript コードに `var data = {"field": "value"};` のような個別の行が含まれている場合、次のようにそのデータを抽出できます。:

```
>>> pattern = r'\bvar\s+data\s*=\s*(\{.*?\})\s*;\s*\n'
>>> json_data = response.css('script::text').re_first(pattern)
>>> json.loads(json_data)
{'field': 'value'}
```

- それ以外の場合は、js2xml を使用して JavaScript コードを XML 文書に変換し、 **セレクター** を使用して解析できます。

  たとえば、JavaScript コードに `var data = {field: "value"};` が含まれている場合、次のようにしてそのデータを抽出できます。:

```
>>> import js2xml
>>> import lxml.etree
>>> from parsel import Selector
>>> javascript = response.css('script::text').get()
>>> xml = lxml.etree.tostring(js2xml.parse(javascript), encoding='unicode')
>>> selector = Selector(text=xml)
>>> selector.css('var[name="data"]').get()
'<var name="data"><object><property name="field"><string>value</string></property>
↪</object></var>'
```

5.7.6 JavaScript の事前レンダリング

追加のリクエストからデータを取得するウェブページでは、目的のデータを含むリクエストを再現することをお勧めします。 多くの場合、その努力は、最小のパース時間とネットワーク転送で構造化された完全なデータという結果で報われます。

けれども、特定のリクエストを再現するのが非常に難しい場合があります。 または、Web ブラウザーで表示される Web ページのスクリーンショットなど、リクエストで提供できないものが必要な場合があります。

これらの場合、シームレスな統合のために、Splash JavaScript レンダリングサービス (https://github.com/scrapinghub/splash) と scrapy-splash (https://github.com/scrapinghub/splash) を使用します。

スプラッシュは HTML として Web ページの *DOM* を返すため、 **セレクター** でパースできます。 configuration または scripting を介して大きな柔軟性を提供します。

以前に記述されたスクリプトを使用する代わりに、Python コードからオンザフライで DOM と対話する、または複数の Web ブラウザーウィンドウを処理するなど、Splash が提供する以上のものが必要な場合は、 代わりに **ヘッドレス ブラウザ** を使用する必要があります。

5.7.7 ヘッドレスブラウザ (headless browser) の使用

ヘッドレスブラウザー (headless browser) は、自動化のための API を提供する特別な Web ブラウザーです。

Scrapy でヘッドレスブラウザを使用する最も簡単な方法は、シームレスな統合のために scrapy-selenium (https://github.com/clemfromspace/scrapy-selenium) とともに Selenium (https://www.seleniumhq.org/) を使用することです。(scrapy-selenium 0.0.7 README 邦訳 https://gist.github.com/kuma35/37d0c6c80af7d5d0e1d01edce30027f1#file-readme-jp-md)

5.8 メモリ・リークのデバッグ

Scrapy では、リクエスト、レスポンス、アイテムなどのオブジェクトのライフタイムは有限です。それらは作成され、しばらく使用され、最終的に破棄されます。

これらすべてのオブジェクトの中で、リクエストはおそらく最も長いライフタイムを持つものです。リクエストを処理するまでスケジューラのキューで待機しているためです。 詳細については、 **アーキテクチャ概観** を参照してください。

これらの Scrapy オブジェクトには (かなり長い) 寿命があるため、それらを適切に解放せずにメモリに蓄積してしまい、「メモリ・リーク」と呼ばれるものを引き起こすリスクが常にあります。

メモリリークのデバッグを支援するために、Scrapy は *trackref* というオブジェクト参照を追跡するための組み込みメカニズムを提供します。また、 *Guppy* というサードパーティ・ライブラリを使用すると、より高度なメモリ・デバッグが可能になります (詳細は以下を参照)。両方のメカニズムは、*Telnet* **コンソール** から使用する必要があります。

5.8.1 メモリ・リークの一般的な原因

Scrapy 開発者がリクエストで参照されるオブジェクトを渡し (たとえば、 *cb_kwargs* または *meta* 属性またはリクエスト・コールバック関数の使用)、そして、事実上、それらの参照されたオブジェクトの寿命をリクエストの寿命に合わせます。これは、Scrapy プロジェクトでのメモリ・リークの最も一般的な原因であり、初心者にとってデバッグが非常に難しいものです。

大きなプロジェクトでは、スパイダーは通常、異なる人々によって作成され、それらのスパイダーの一部は漏出 (leak) する可能性があり、したがって、同時に実行されると他の (ちゃんと書かれた）スパイダーに次々に影響を与え、クロール・プロセス全体に影響を与えます。

(以前に割り当てられた) リソースを適切に解放していない場合、作成したカスタムミドルウェア、パイプライン、または拡張機能からもリークが発生する可能性があります。たとえば、*spider_opened* でリソースを割り当てても、 *spider_closed* でリソースを解放しないと、**プロセスごとに複数のスパイダー** を実行している場合に問題が発生する可能性があります。

リクエストが多すぎるのか？

デフォルトでは、Scrapy はリクエスト・キューをメモリに保持します。 *Request* オブジェクトと Request 属性で参照されるすべてのオブジェクト（例 *cb_kwargs* と *meta*)。必ずしもリークではありませんが、これには大量のメモリが必要になる場合があります。 **永続ジョブ・キュー** を有効にすると、メモリ使用量を制御できます。

5.8.2 `trackref` を使用したメモリ・リークのデバッグ

`trackref` は、メモリ・リークの最も一般的なケースをデバッグするために Scrapy が提供するモジュールです。基本的に、すべての、生存中の、リクエスト、レスポンス、アイテム、セレクタ・オブジェクトへの参照を追跡します。

telnet コンソールに入り、 *print_live_refs()* のエイリアスである `prefs()` 関数を使用して、(上記のクラスの) オブジェクトが現在何個生きているかを検査することができます。

```
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
```

ご覧のとおり、このレポートには各クラスの最も古いオブジェクトの「年齢」も表示されます。プロセスごとに複数のスパイダーを実行している場合、最も古い要求または応答を調べることで、どのスパイダーがリークしている

かを把握できます。 *get_oldest()* 関数を使用して (telnet コンソールから) 各クラスの最も古いオブジェクトを取得できます。

どのオブジェクトが追跡されますか？

trackrefs によって追跡されるオブジェクトはすべてこれらのクラス (およびそのすべてのサブクラス) からのものです:

- *scrapy.http.Request*
- *scrapy.http.Response*
- *scrapy.item.Item*
- scrapy.selector.Selector
- *scrapy.spiders.Spider*

実例

メモリ・リークの仮定ケースの具体例を見てみましょう。 以下のようなスパイダーがあるとします:

```
return Request("http://www.somenastyspider.com/product.php?pid=%d" % product_id,
               callback=self.parse, cb_kwargs={'referer': response})
```

この例は、レスポンスの有効期間をリクエストの有効期間と効果的に結び付けるレスポンス参照をリクエスト内に渡しているため、間違いなくメモリ・リークが発生します。

trackref ツールを使用して、原因を (もちろん、事前に知らないものとして) 発見する方法を見てみましょう。

クローラーが数分間実行され、メモリー使用量が大幅に増加したことに気付いたら、telnet コンソールに入り、生存中の参照 (Live References) を確認できます:

```
>>> prefs()
Live References

SomenastySpider                     1   oldest: 15s ago
HtmlResponse                     3890   oldest: 265s ago
Selector                            2   oldest: 0s ago
Request                          3878   oldest: 250s ago
```

レスポンスはリクエストと比較して比較的短い寿命であるはずなので、非常に多くの生存中のレスポンスが存在するという事実 (そしてそれらが非常に古いという事実) は間違いなく疑わしいです。レスポンスの数はリクエストの数と似ているため、何らかの形で結び付けられているように見えます。これで、スパイダーのコードを調べて、リークを生成している厄介な当該コード (リクエスト内でレスポンス参照を渡している) を発見できます。

生存中オブジェクトに関する追加情報が役立つ場合があります。

```
>>> from scrapy.utils.trackref import get_oldest
>>> r = get_oldest('HtmlResponse')
>>> r.url
'http://www.somenastyspider.com/product.php?pid=123'
```

最も古いオブジェクトを取得する代わりに、すべてのオブジェクトを反復処理する場合は、 *scrapy.utils.trackref.iter_all()* 関数を使用できます:

```
>>> 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',
...
```

スパイダーが多すぎるのか？

プロジェクトで並行して実行されるスパイダーが多すぎる場合、 `prefs()` の出力は読みにくい場合があります。このため、その関数には、特定のクラス (およびそのすべてのサブクラス) を無視するために使用できる `ignore` 引数があります。たとえば、以下はスパイダーへの生存中参照を表示しません:

```
>>> from scrapy.spiders import Spider
>>> prefs(ignore=Spider)
```

scrapy.utils.trackref モジュール

trackref モジュールで利用可能な関数は以下のとおりです。

class scrapy.utils.trackref.**object_ref**

trackref モジュールを使用して生存中のインスタンスを追跡する場合は、(オブジェクトの代わりに) このクラスから継承します。

scrapy.utils.trackref.**print_live_refs**(*class_name*, *ignore=NoneType*)

クラス名でグループ化された生存中参照のレポートを出力します。

パラメータ **ignore** (*class or classes tuple*) – 指定された場合、指定されたクラス (またはクラスのタプル) からのすべてのオブジェクトは無視されます。

scrapy.utils.trackref.**get_oldest**(*class_name*)

指定されたクラス名で生きている最も古いオブジェクトを返すか、見つからない場合は None を返します。最初に *print_live_refs()* を使用して、クラス名ごとに追跡されているすべての生存中のオブジェクトのリストを取得します。

scrapy.utils.trackref.**iter_all**(*class_name*)

指定されたクラス名で生きているすべてのオブジェクトのイテレータを返します。見つからない場合は

None を返します。最初に *print_live_refs()* を使用して、クラス名ごとに追跡されているすべての生存中のオブジェクトのリストを取得します。

5.8.3 Guppy を使ってメモリ・リークをデバッグする

‘‘ trackref‘‘は、メモリ・リークを追跡するための非常に便利なメカニズムを提供しますが、メモリリークを引き起こす可能性が高いオブジェクト (リクエスト、レスポンス、アイテム、セレクタ) のみを追跡します。 ただし、他の (多かれ少なかれ不明瞭な) オブジェクトからメモリ・リークが発生する場合があります。そうした場合で、trackref を使用してリークを見つけることができない場合、更に別の方法があります。それは Guppy library です。そして、Python3 を使用している場合は、 *muppy* **を使ってメモリ・リークをデバッグする** を参照してください。

pip を使用する場合、次のコマンドで Guppy をインストールできます:

```
pip install guppy
```

telnet コンソールには、グッピー・ヒープ・オブジェクトにアクセスするための組み込みのショートカット (hpy) も付属しています。 Guppy を使用して、ヒープで使用可能なすべての Python オブジェクトを表示する例を次に示します:

```
>>> 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.>
```

ほとんどのスペースが辞書 (dict) によって使用されていることがわかります。次に、それらの辞書がどの属性から参照されているかを見たい場合は、以下のようにします:

```
>>> 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
```

(次のページに続く)

(前のページからの続き)

```
     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.>
```

ご覧のとおり、Guppy モジュールは非常に強力ですが、Python 内部についての深い知識も必要です。 グッピーの詳細については、 Guppy documentation を参照してください。

5.8.4 muppy を使ってメモリ・リークをデバッグする

Python 3 を使用している場合、 Pympler から muppy を使用できます。

`pip` を使用する場合、次のコマンドで muppy をインストールできます:

```
pip install Pympler
```

muppy を使用してヒープ内で使用可能なすべての Python オブジェクトを表示する例を次に示します:

```
>>> from pympler import muppy
>>> all_objects = muppy.get_objects()
>>> len(all_objects)
28667
>>> from pympler import summary
>>> suml = summary.summarize(all_objects)
>>> summary.print_(suml)
                                 types |   # objects |   total size
====================================== | =========== | ============
                          <class 'str |        9822 |      1.10 MB
                         <class 'dict |        1658 |    856.62 KB
                         <class 'type |         436 |    443.60 KB
                         <class 'code |        2974 |    419.56 KB
           <class '_io.BufferedWriter |           2 |    256.34 KB
                          <class 'set |         420 |    159.88 KB
           <class '_io.BufferedReader |           1 |    128.17 KB
           <class 'wrapper_descriptor |        1130 |     88.28 KB
                       <class 'tuple |        1304 |     86.57 KB
                      <class 'weakref |        1013 |     79.14 KB
  <class 'builtin_function_or_method |         958 |     67.36 KB
            <class 'method_descriptor |         865 |     60.82 KB
                  <class 'abc.ABCMeta |          62 |     59.96 KB
                         <class 'list |         446 |     58.52 KB
                          <class 'int |        1425 |     43.20 KB
```

muppy の詳細については、 muppy documentation を参照してください。

5.8.5 Scrapy ではリークしてないのにリークしてる orz

Scrapy プロセスのメモリ使用量は増加するだけで、決して減少しないことに気付く場合があります。 残念ながら、Scrapy もプロジェクトもメモリをリークしていなくても、これは起こり得ます。これは、Python の (あまりよくない) 既知の問題が原因であり、場合によっては解放されたメモリをオペレーティングシステムに返さないことがあります。 この問題の詳細については以下を参照して下さい。

- Python Memory Management
- Python Memory Management Part 2
- Python Memory Management Part 3

this paper で詳しく説明されている、エヴァン・ジョーンズによって提案された改善点は、Python 2.5 に統合されましたが、これは問題を軽減するだけで、完全には修正しません。 当該箇所を引用すると:

> **残念ながら、このパッチは、オブジェクトが割り当てられていない場合にのみアリーナを解放できます。これは、断片化が大きな問題であることを意味します。アプリケーションには、すべてのアリーナに散らばった数メガバイトの空きメモリがありますが、どれも解放できません。これは、すべてのメモリ・アロケータで発生する問題です。これを解決する唯一の方法は、メモリ内のオブジェクトを移動できる圧縮ガベージコレクタに移動することです。これには、*Python* インタープリターの大幅な変更が必要になります。**

メモリ消費を適切に保つために、ジョブをいくつかの小さなジョブに分割するか、 **永続ジョブ・キュー** を有効にして、スパイダーを時々停止/開始できます。

5.9 ファイルと画像のダウンロードおよび処理

Scrapy は、特定のアイテムに添付されたファイルをダウンロードするための再利用可能な **アイテム・パイプライン** を提供します (たとえば、製品をスクレイピングし、画像をローカルにダウンロードする場合)。 これらのパイプラインは少しの機能と構造を共有します (媒体パイプラインと呼びます) が、通常はファイル・パイプラインまたは画像パイプラインを使用します。

両方のパイプラインは以下の機能を実装しています:

- 最近ダウンロードした媒体の再ダウンロードを避ける
- メディアの保存場所の指定 (ファイル・システム・ディレクトリ、Amazon S3 バケット、Google Cloud Storage バケット)

画像パイプラインには、画像を処理するためのいくつかの追加機能があります:

- ダウンロードしたすべての画像を共通の形式 (JPG) とモード (RGB) に変換する

- サムネイル生成
- 画像の幅/高さをチェックして、最小の制約を満たしていることを確認します

パイプラインは、現在ダウンロードがスケジュールされている媒体 URL の内部キューを保持し、到着したレスポンスのうち、同じ媒体を含むレスポンスをそのキューに接続します。これにより、複数のアイテムで共有されている場合に同じ媒体を複数回ダウンロードすることがなくなります。

5.9.1 ファイル・パイプラインの使用

`FilesPipeline` を使用する場合の典型的なワークフローは次のようになります:

1. Spider では、アイテムをスクレイプし、目的の URL を `file_urls` フィールドに入れます。
2. アイテムはスパイダーから返され、アイテム・パイプラインに送られます。
3. アイテムが `FilesPipeline` に到達すると、`file_urls` フィールドの URL は標準の Scrapy スケジューラーとダウンローダー (スケジューラーとダウンローダーのミドルウェアが再利用されることを意味します) を使用してダウンロード用にスケジュールされます。しかし、他のページの処理より高い優先度で、他のページがスクレイピングされる前にダウンロード用の処理を行います。ファイルのダウンロードが完了する (または何らかの理由で失敗する)) まで、アイテムはその特定のパイプライン・ステージでロックされたままになります。
4. ファイルがダウンロードされると、別のフィールド (`files`) に結果が入力されます。このフィールドには、ダウンロードしたパス、スクレイプされた元の URL(`file_urls` フィールドから取得) や、ファイルのチェックサムのような、ダウンロードしたファイルに関する情報を含む辞書のリストが含まれます。`files` フィールドのリストにあるファイルは、 元の `file_urls` フィールドと同じ順序を保持します。ファイルのダウンロードに失敗した場合、エラーがログに記録され、ファイルは `files` フィールドに存在しません。

5.9.2 画像パイプラインの使用

`ImagesPipeline` を使用することは、使用されるデフォルトのフィールド名が異なることを除き `FilesPipeline` を使用することとよく似ています。あなたがアイテムの画像 URL に `image_urls` を使用すると、ダウンロードした画像に関する情報が `images` フィールドに入力されます。

画像ファイルに *`ImagesPipeline`* を使用する利点は、サムネイルの生成やサイズに基づいた画像のフィルタリングなどの追加機能を設定できることです。

画像パイプラインは、画像を JPEG/RGB 形式にサムネイル化および正規化するために Pillow を使用するため、使用するにはこのライブラリをインストールする必要があります。 Python Imaging Library (PIL) もほとんどの場合に動作するはずですが、セットアップによっては問題を引き起こすことが知られているため、PIL の代わりに Pillow を使用することをお勧めします。

5.9.3 あなたの媒体パイプラインを有効にする

媒体パイプラインを有効にするには、まずプロジェクトに *ITEM_PIPELINES* 設定を追加する必要があります。

画像パイプラインを使うには:

```
ITEM_PIPELINES = {'scrapy.pipelines.images.ImagesPipeline': 1}
```

ファイル・パイプラインを使うには:

```
ITEM_PIPELINES = {'scrapy.pipelines.files.FilesPipeline': 1}
```

注釈: ファイルと画像のパイプラインの両方を同時に使用することもできます。

次に、ダウンロードした画像の保存に使用される有効な値でターゲット・ストレージ設定を構成 (configure) します。 そうしないと、パイプラインは *ITEM_PIPELINES* 設定に含めても無効のままになります。

ファイル・パイプラインの場合、 *FILES_STORE* 設定を設定します:

```
FILES_STORE = '/path/to/valid/dir'
```

画像パイプラインの場合、 *IMAGES_STORE* 設定を設定します:

```
IMAGES_STORE = '/path/to/valid/dir'
```

5.9.4 サポートされるストレージ

現在、ファイル・システムは公式にサポートされている唯一のストレージですが、 Amazon S3 と Google Cloud Storage にファイルを保存することもサポートされています。

ファイル・システム・ストレージ

ファイルは、ファイル名の URL から生成する SHA1 hash を使用して保存されます。

たとえば、次の画像 URL:

```
http://www.example.com/image.jpg
```

SHA1 hash は:

```
3afec3b4765f8f0a07b78f98c07b83f013567a0a
```

ダウンロードされ、以下のファイルに保存されます:

```
<IMAGES_STORE>/full/3afec3b4765f8f0a07b78f98c07b83f013567a0a.jpg
```

ここで:

- <IMAGES_STORE> は、画像パイプラインの *IMAGES_STORE* 設定で定義されているディレクトリです。
- ‘‘ full‘‘ は、(使用する場合) サムネイルから完全な画像を分離するためのサブディレクトリです。詳細は **画像のサムネイル生成** を参照。

Amazon S3 ストレージ

FILES_STORE と *IMAGES_STORE* は Amazon S3 バケットを表すことができます。 Scrapy は自動的にファイルをバケットにアップロードします。

たとえば、以下は有効な *IMAGES_STORE* 値です:

```
IMAGES_STORE = 's3://bucket/images'
```

あなたは *FILES_STORE_S3_ACL* と *IMAGES_STORE_S3_ACL* 設定によって定義される、保存されたファイルに使用されるアクセス制御リスト (ACL) ポリシーを変更できます。デフォルトでは、ACL は `private` に設定されています。ファイルを公開するには、 `public-read` ポリシーを使用します:

```
IMAGES_STORE_S3_ACL = 'public-read'
```

詳細については、Amazon S3 開発者ガイドの canned ACLs を参照してください。

Scrapy は `boto` / `botocore` を内部で使用するため、他の S3 のようなストレージも使用できます。自己ホスト型の Minio や s3.scality のようなストレージです。あなたがする必要があるのはあなたの Scrapy 設定でエンドポイント・オプションを設定することです:

```
AWS_ENDPOINT_URL = 'http://minio.example.com:9000'
```

セルフホスティングの場合は、SSL を使用する必要はなく、SSL 接続を確認する必要もないと感じるかもしれません:

```
AWS_USE_SSL = False # or True (None by default)
AWS_VERIFY = False # or True (None by default)
```

Google Cloud ストレージ

FILES_STORE と *IMAGES_STORE* は、Google Cloud Storage バケットを表すことができます。Scrapy は自動的にファイルをバケットにアップロードします (google-cloud-storage が必要です)。

たとえば、以下は有効な *IMAGES_STORE* および *GCS_PROJECT_ID* 設定です:

```
IMAGES_STORE = 'gs://bucket/images/'
GCS_PROJECT_ID = 'project_id'
```

認証については、この documentation を参照してください。

FILES_STORE_GCS_ACL および *IMAGES_STORE_GCS_ACL* 設定によって定義される、保存されたファイルに使用されるアクセス制御リスト (ACL) ポリシーを変更できます。デフォルトでは、ACL は `''` (空の文字列) に設定されます。これは、Cloud Storage がバケットのデフォルト・オブジェクト ACL をオブジェクトに適用することを意味します。ファイルを公開するには、 `publicRead` ポリシーを使用します:

```
IMAGES_STORE_GCS_ACL = 'publicRead'
```

詳細については、Google Cloud Platform Developer Guide の Predefined ACLs を参照してください。

5.9.5 使用例

媒体パイプラインを使用するには、まず、 **媒体パイプラインを有効にする** を行います。

次に、スパイダーが URL キー (ファイルまたは画像パイプラインの場合はそれぞれ `file_urls` または `image_urls`) を含む辞書を返すと、パイプラインは結果をそれぞれのキー (`files` または `images`) の値として返します。

Item を使用する場合は、以下の Images Pipeline の例のように、必要なフィールドを持つカスタム・アイテムを定義します:

```
import scrapy

class MyItem(scrapy.Item):

    # ... other item fields ...
    image_urls = scrapy.Field()
    images = scrapy.Field()
```

URL キーまたは結果キーに別のフィールド名を使用する場合は、それをオーバーライドすることもできます。

ファイル・パイプラインでは、 *FILES_URLS_FIELD* and/or *FILES_RESULT_FIELD* 設定を設定します:

```
FILES_URLS_FIELD = 'field_name_for_your_files_urls'
FILES_RESULT_FIELD = 'field_name_for_your_processed_files'
```

画像パイプラインでは、 *IMAGES_URLS_FIELD* and/or *IMAGES_RESULT_FIELD* 設定を設定します:

```
IMAGES_URLS_FIELD = 'field_name_for_your_images_urls'
IMAGES_RESULT_FIELD = 'field_name_for_your_processed_images'
```

より複雑なものが必要で、カスタム・パイプラインの動作をオーバーライドする場合は、**媒体パイプラインの拡張**を参照してください。

ImagePipeline を継承する複数の画像パイプラインがあり、あなたが異なるパイプラインで異なる設定を行いたい場合は、パイプライン・クラス名を大文字化した名前を先頭に付けた設定キーを設定できます。例えば、パイプラインの名前が MyPipeline で、カスタム IMAGES_URLS_FIELD が必要な場合は、設定 MYPIPELINE_IMAGES_URLS_FIELD を定義すると、カスタム設定が使用されます。

5.9.6 追加機能

ファイルの有効期限

画像パイプラインは、最近ダウンロードされたファイルのダウンロードを回避します。この保持遅延を調整するには、 *FILES_EXPIRES* (または、画像パイプラインの場合は *IMAGES_EXPIRES*) 設定を使用します。これは、日数で保持遅延を指定します:

```
# 120 days of delay for files expiration
FILES_EXPIRES = 120

# 30 days of delay for images expiration
IMAGES_EXPIRES = 30
```

両方の設定のデフォルト値は 90 日です。

FilesPipeline をサブクラス化するパイプラインがあり、それに対して別の設定が必要な場合は、クラス名を大文字化した名前を先頭に付けた設定キーを設定できます。例えば、パイプラインの名前が MyPipeline で、カスタム FIlE_EXPIERS が必要な場合は、以下のように設定すると、カスタム設定が使用されます。

MYPIPELINE_FILES_EXPIRES = 180

こうすると、パイプライン・クラス MyPipeline の有効期限は 180 に設定されます。

画像のサムネイル生成

画像 Pipeline は、ダウンロードした画像のサムネイルを自動的に作成できます。 この機能を使用するには、*IMAGES_THUMBS* を辞書に設定する必要があります。ここで、キーはサムネイル名であり、値はその寸法です。

例えば:

```
IMAGES_THUMBS = {
    'small': (50, 50),
```

(次のページに続く)

(前のページからの続き)

```
    'big': (270, 270),
}
```

この機能を使用すると、画像パイプラインは指定された各サイズのサムネイルをこの形式で作成します:

```
<IMAGES_STORE>/thumbs/<size_name>/<image_id>.jpg
```

ここで:

- <size_name> は *IMAGES_THUMBS* 辞書キーで指定されたもの (small 、 big など) です
- <image_id> は画像の URL の SHA1 hash です

small および big のサムネイル名を使用して保存された画像ファイルの例:

```
<IMAGES_STORE>/full/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/small/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/big/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
```

最初のものは、サイトからダウンロードされたフル画像です。

小さな画像を除外する

画像パイプラインを使用する場合、 *IMAGES_MIN_HEIGHT* および *IMAGES_MIN_WIDTH* 設定で最小許容サイズを指定することにより、小さすぎる画像をドロップできます。

例えば:

```
IMAGES_MIN_HEIGHT = 110
IMAGES_MIN_WIDTH = 110
```

注釈: サイズの制約は、サムネイルの生成にはまったく影響しません。

---

1 つのサイズ制約または両方を設定することができます。両方を設定すると、両方の最小サイズを満たす画像のみが保存されます。上記の例では、サイズが (105 × 105) または (105 × 200) または (200 × 105) の画像はすべて削除されます。これは、少なくとも 1 つの寸法が制約よりも短いためです。

デフォルトでは、サイズの制限はないため、すべての画像が処理されます。

リダイレクトを許可する

デフォルトでは、媒体パイプラインはリダイレクトを無視します。つまり、媒体アファイル URL リクエストへの HTTP リダイレクトは、媒体のダウンロードが失敗したと見なされることを意味します。

媒体のリダイレクトを処理するには、この設定を True に設定します:

```
MEDIA_ALLOW_REDIRECTS = True
```

5.9.7 媒体パイプラインの拡張

ここで、カスタム・ファイル・パイプラインでオーバーライドできるメソッドを参照してください:

class scrapy.pipelines.files.**FilesPipeline**

file_path(*request*, *response*, *info*)

このメソッドは、ダウンロードされたアイテムごとに 1 回呼び出されます。 指定された *response* から始まるファイルのダウンロード・パスを返します。

response に加えて、このメソッドは元の request と info を受け取ります。

このメソッドをオーバーライドして、各ファイルのダウンロード・パスをカスタマイズできます。

たとえば、ファイルの URL が通常のパスのように終わる場合 (例 https://example.com/a/b/c/foo.png)、次のアプローチを使用して、全てのファイルを元のファイル (例 files/foo.png) で files フォルダーにダウンロードできます:

```
import os
from urllib.parse import urlparse

from scrapy.pipelines.files import FilesPipeline

class MyFilesPipeline(FilesPipeline):

    def file_path(self, request, response, info):
        return 'files/' + os.path.basename(urlparse(request.url).path)
```

デフォルトでは *file_path()* メソッドは full/<request URL hash>.<extension> を返します。

get_media_requests(*item*, *info*)

ワークフローにあるように、パイプラインはアイテムからダウンロードする画像の URL を取得します。これを行うには、 *get_media_requests()* メソッドをオーバーライドして、各ファイル URL のリクエストを返すことができます:

```
def get_media_requests(self, item, info):
    for file_url in item['file_urls']:
        yield scrapy.Request(file_url)
```

これらのリクエストはパイプラインによって処理され、ダウンロードが完了すると、結果が *item_completed()* メソッドに 2 要素タプルのリストとして送信されます。各タプルには (success, file_info_or_error) が含まれます:

- success は、画像が正常にダウンロードされた場合は True であるブール値であり、何らかの理由で失敗した場合は False です
- file_info_or_error は、(success True の場合) 以下のキーを含む辞書です。(success False の場合) 問題が発生した場合は Twisted Failure を含みます。
  - url - ファイルのダウンロード元の URL。これは *get_media_requests()* メソッドから返されるリクエストの URL です。
  - path - ファイルが保存されたパス (*FILES_STORE* への相対パス)
  - checksum - 画像コンテンツの MD5 ハッシュ (MD5 hash)

item_completed() が受け取るタプルのリストは、 *get_media_requests()* メソッドから返されたリクエストと同じ順序を保持することが保証されています。

以下は results 引数の典型的な値です:

```
[(True,
  {'checksum': '2b00042f7481c7b056c4b410d28f33cf',
   'path': 'full/0a79c461a4062ac383dc4fade7bc09f1384a3910.jpg',
   'url': 'http://www.example.com/files/product1.pdf'}),
 (False,
  Failure(...))]
```

デフォルトでは、 *get_media_requests()* メソッドは None を返します。これは、アイテムにダウンロードするファイルがないことを意味します。

item_completed(*results*, *item*, *info*)

FilesPipeline.item_completed() メソッドは、1 つのアイテムに対するすべてのファイルリクエストが完了した (ダウンロードが完了したか、何らかの理由で失敗した) ときに呼び出されます。

item_completed() メソッドは、後続のアイテム・パイプライン・ステージに送信される出力を返す必要があるため、パイプラインの場合と同様に、アイテムを返す (またはドロップする) 必要があります。

以下は *item_completed()* メソッドの例で、(結果で渡された) ダウンロードしたファイル・パスを file_paths アイテム・フィールドに保存し、ファイルが含まれていない場合はアイテムをドロップします:

```
from scrapy.exceptions import DropItem

def item_completed(self, results, item, info):
```

(次のページに続く)

(前のページからの続き)

```
    file_paths = [x['path'] for ok, x in results if ok]
    if not file_paths:
        raise DropItem("Item contains no files")
    item['file_paths'] = file_paths
    return item
```

デフォルトでは、 *item_completed()* メソッドはアイテムを返します:

カスタム画像パイプラインでオーバーライドできるメソッドをご覧ください:

class scrapy.pipelines.images.**ImagesPipeline**

ImagesPipeline は FilesPipeline の拡張であり、フィールド名をカスタマイズして画像のカスタム動作を追加します。

file_path(*request*, *response*, *info*)

このメソッドは、ダウンロードされたアイテムごとに 1 回呼び出されます。 指定された *response* から始まるファイルのダウンロード・パスを返します。

response に加えて、このメソッドは元の request と info を受け取ります。

このメソッドをオーバーライドして、各ファイルのダウンロード・パスをカスタマイズできます。

たとえば、ファイルの URL が通常のパスのように終わる場合 (例 https://example.com/a/b/c/foo.png)、次のアプローチを使用して、全てのファイルを元のファイル (例 files/foo.png) で files フォルダーにダウンロードできます:

```
import os
from urllib.parse import urlparse

from scrapy.pipelines.images import ImagesPipeline

class MyImagesPipeline(ImagesPipeline):

    def file_path(self, request, response, info):
        return 'files/' + os.path.basename(urlparse(request.url).path)
```

デフォルトでは *file_path()* メソッドは full/<request URL hash>.<extension> を返します。

get_media_requests(*item*, *info*)

FilesPipeline.get_media_requests() メソッドと同じように機能しますが、画像の URL に異なるフィールド名を使用します。

各画像 URL のリクエストを返す必要があります。

item_completed(*results*, *item*, *info*)

ImagesPipeline.item_completed() メソッドは、1 つのアイテムに対するすべての画像リクエストが完了した (ダウンロードが完了したか、何らかの理由で失敗した) ときに呼び出されます。

FilesPipeline.item_completed() メソッドと同じように機能しますが、画像のダウンロード結果を保存するために異なるフィールド名を使用します。

デフォルトでは、 *item_completed()* メソッドはアイテムを返します:

5.9.8 カスタム画像パイプライン例

上記の例示されたメソッドでの画像パイプラインの完全な例を以下に示します:

```
import scrapy
from scrapy.pipelines.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
```

5.10 スパイダーのデプロイ

この節では、Scrapy スパイダーをデプロイして定期的に実行するためのさまざまなオプションについて説明します。ローカルマシンで Scrapy スパイダーを実行することは (初期の) 開発段階には非常に便利ですが、長時間実行されるスパイダーを実行したり、スパイダーを動かして実稼働で実行する必要がある場合はそれほど便利ではありません。そこで、Scrapy スパイダーをデプロイするためのソリューションの出番です。

Scrapy スパイダーをデプロイするための一般的な選択肢は以下の通りです:

- *Scrapyd* (オープン・ソース)
- *Scrapy Cloud* (クラウド)

5.10.1 Scrapyd サーバへのデプロイ

Scrapyd は、Scrapy スパイダーを実行するためのオープン・ソース・アプリケーションです。Scrapy スパイダーを実行および監視できる HTTP API をサーバーに提供します。

スパイダーを Scrapyd にデプロイするには、 scrapyd-client パッケージで提供される scrapyd-deploy ツールを使用できます。 詳細については、 scrapyd-deploy documentation を参照してください。

Scrapyd は、一部の Scrapy 開発者によって管理されています。

5.10.2 Scrapy クラウドへのデプロイ

Scrapy Cloud は、Scrapy の背後にある会社である Scrapinghub によってクラウドベースのサービスとしてホストされています。

Scrapy クラウドは、サーバーをセットアップおよび監視する必要をなくし、スパイダーを管理し、スクレイプされたアイテム、ログ、および統計を確認するための素晴らしい UI を提供します。

スパイダーを Scrapy Cloud にデプロイするには、 shub コマンドラインツールを使用できます。詳細については、Scrapy Cloud documentation を参照してください。

Scrapy クラウドは Scrapyd と互換性があり、必要に応じてそれらを切り替えることができます。設定は `scrapyd-deploy` と同様に `scrapy.cfg` ファイルから読み込まれます。

5.11 AutoThrottle 拡張機能

これは、Scrapy サーバーとクロールする Web サイトの両方の負荷に基づいて、クロール速度を自動的に落とすための拡張機能です。

5.11.1 設計目標

1. デフォルト・ゼロのダウンロード遅延を使用する代わりに、サイトに優しくする
2. Scrapy を最適なクロール速度に自動的に調整するので、ユーザーは最適なものを見つけるためにダウンロードの遅延を調整する必要がありません。 ユーザーは、許可する同時要求の最大数を指定するだけでよく、拡張機能が残りを行います。

5.11.2 仕組み

AutoThrottle 拡張機能はダウンロード遅延を動的に調整して、スパイダーが各リモート Web サイトに平均的に *AUTOTHROTTLE_TARGET_CONCURRENCY* 並列リクエストを送信するようにします。

ダウンロード遅延を使用して遅延を計算します。主なアイデアは次のとおりです。サーバーが応答するのに latency 秒が必要な場合、クライアントは latency/N 秒ごとにリクエストを送信して、N 個のリクエストを並行して処理する必要があります。

遅延を調整する代わりに、小さな固定ダウンロード遅延を設定し、 *CONCURRENT_REQUESTS_PER_DOMAIN* または *CONCURRENT_REQUESTS_PER_IP* オプションを使用して同時実行に厳しい制限を課すことができます。この 2 つは同様の効果を提供しますが、いくつかの重要な違いがあります:

- ダウンロードの遅延が小さいため、リクエストのバーストがときどき発生します;
- 多くの場合、200 以外の (エラー) レスポンスは通常のレスポンスよりも速く返される可能性があるため、サーバーがエラーを返し始めると、ダウンロード遅延が小さく、ハード同時実行制限のクローラーがサーバーにリクエストをより速く送信します。 しかし、これはクローラーが行うべきことの反対です。エラーが発生した場合、速度を落とすのがより理にかなっています。これらのエラーはリクエスト率が高いことが原因である可能性があります。

AutoThrottle にはこれらの問題はありません。

5.11.3 速度を絞るアルゴリズム

AutoThrottle アルゴリズムは、次のルールに基づいてダウンロード遅延を調整します:

1. スパイダーは常に *AUTOTHROTTLE_START_DELAY* のダウンロード遅延で始まります;
2. レスポンスが受信されると、ターゲットのダウンロード遅延は latency / N として計算されます。latency は応答の遅延であり、 N は *AUTOTHROTTLE_TARGET_CONCURRENCY* です。
3. 次のリクエストのダウンロード遅延は、以前のダウンロード遅延とターゲット・ダウンロード遅延の平均に設定されます;
4. 200 以外のレスポンスのレイテンシ (latency) では、遅延を減らすことはできません;
5. ダウンロードの遅延が *DOWNLOAD_DELAY* よりも小さくなったり、 *AUTOTHROTTLE_MAX_DELAY* よりも大きくなったりすることはできません

注釈: AutoThrottle 拡張機能は、並列性と遅延の標準的な Scrapy 設定を優先します。これは、*CONCURRENT_REQUESTS_PER_DOMAIN* と *CONCURRENT_REQUESTS_PER_IP* オプションを尊重し、かつ、*DOWNLOAD_DELAY* よりも低いダウンロード遅延を設定しないことを意味します。

Scrapy では、ダウンロードの遅延は、TCP 接続の確立から HTTP ヘッダーの受信までの経過時間として測定されます。

Scrapy は、例えば、スパイダー・コールバックの処理に忙しいとダウンロードに参加できないため、これらのレイテンシ (latency) を協調マルチタスク環境で正確に測定するのは非常に難しいことに注意してください。 ただし、

これらのレイテンシ (latency) は、Scrapy(および最終的にはサーバー) がどれだけビジー (busy) であるかの妥当な推定値を提供する必要があり、この拡張機能はその前提に基づいています。

5.11.4 設定

AutoThrottle 拡張機能の制御に使用される設定は次のとおりです:

- *AUTOTHROTTLE_ENABLED*
- *AUTOTHROTTLE_START_DELAY*
- *AUTOTHROTTLE_MAX_DELAY*
- *AUTOTHROTTLE_TARGET_CONCURRENCY*
- *AUTOTHROTTLE_DEBUG*
- *CONCURRENT_REQUESTS_PER_DOMAIN*
- *CONCURRENT_REQUESTS_PER_IP*
- *DOWNLOAD_DELAY*

詳細については **仕組み** を参照してください。

AUTOTHROTTLE_ENABLED

デフォルト: `False`

AutoThrottle 拡張機能を有効にします。

AUTOTHROTTLE_START_DELAY

デフォルト: `5.0`

初期ダウンロード遅延 (秒単位)。

AUTOTHROTTLE_MAX_DELAY

デフォルト: `60.0`

待ち時間が長い場合に設定される最大ダウンロード遅延 (秒単位)。

AUTOTHROTTLE_TARGET_CONCURRENCY

バージョン 1.1 で追加.

デフォルト: 1.0

Scrapy がリモート Web サイトに並列して送信するリクエストの平均数。

デフォルトでは、AutoThrottle は遅延を調整して、各リモート Web サイトに単一の同時リクエストを送信します。このオプションをより高い値 (例: 2.0) に設定すると、リモート・サーバーのスループットと負荷が増加します。AUTOTHROTTLE_TARGET_CONCURRENCY の値 (例: 0.5) が低いと、クローラーはより保守的で礼儀正しくなります。

AutoThrottle 拡張機能が有効になっている場合、 *CONCURRENT_REQUESTS_PER_DOMAIN* および *CONCURRENT_REQUESTS_PER_IP* オプションは引き続き尊重されることに注意してください。つまり、 AUTOTHROTTLE_TARGET_CONCURRENCY が *CONCURRENT_REQUESTS_PER_DOMAIN* または *CONCURRENT_REQUESTS_PER_IP* よりも高い値に設定されている場合、クローラーはこの同時リクエスト数に到達しません。

指定されたすべての時点で、Scrapy は AUTOTHROTTLE_TARGET_CONCURRENCY よりも多いまたは少ない同時リクエストを送信できます。クローラーが接近しようとする推奨値であり、ハードリミットではありません。

AUTOTHROTTLE_DEBUG

デフォルト: False

AutoThrottle デバッグモードを有効にすると、受信したすべてのレスポンスの統計が表示されるため、スロットル・パラメーターがリアルタイムでどのように調整されているかを確認できます。

5.12 ベンチマーキング

バージョン 0.17 で追加.

Scrapy には、ローカル HTTP サーバーを生成し、可能な最大速度でクロールする単純なベンチマーキング・スイートが付属しています。 このベンチマークの目標は、ハードウェアで Scrapy がどのように機能するかを把握し、比較のための共通のベースラインを得ることにあります。 何もせず、リンクをたどるシンプルなスパイダーを使用します。

その実行は:

```
scrapy bench
```

次のような出力が表示されるはずです:

```
2016-12-16 21:18:48 [scrapy.utils.log] INFO: Scrapy 1.2.2 started (bot: quotesbot)
2016-12-16 21:18:48 [scrapy.utils.log] INFO: Overridden settings: {'CLOSESPIDER_TIMEOUT
↪': 10, 'ROBOTSTXT_OBEY': True, 'SPIDER_MODULES': ['quotesbot.spiders'], 'LOGSTATS_
↪INTERVAL': 1, 'BOT_NAME': 'quotesbot', 'LOG_LEVEL': 'INFO', 'NEWSPIDER_MODULE':
↪'quotesbot.spiders'}
```

(次のページに続く)

(前のページからの続き)

```
2016-12-16 21:18:49 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.closespider.CloseSpider',
 'scrapy.extensions.logstats.LogStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.corestats.CoreStats']
2016-12-16 21:18:49 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',
 'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2016-12-16 21:18:49 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2016-12-16 21:18:49 [scrapy.middleware] INFO: Enabled item pipelines:
[]
2016-12-16 21:18:49 [scrapy.core.engine] INFO: Spider opened
2016-12-16 21:18:49 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/
↪min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:50 [scrapy.extensions.logstats] INFO: Crawled 70 pages (at 4200 pages/
↪min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:51 [scrapy.extensions.logstats] INFO: Crawled 134 pages (at 3840␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:52 [scrapy.extensions.logstats] INFO: Crawled 198 pages (at 3840␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:53 [scrapy.extensions.logstats] INFO: Crawled 254 pages (at 3360␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:54 [scrapy.extensions.logstats] INFO: Crawled 302 pages (at 2880␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:55 [scrapy.extensions.logstats] INFO: Crawled 358 pages (at 3360␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:56 [scrapy.extensions.logstats] INFO: Crawled 406 pages (at 2880␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:57 [scrapy.extensions.logstats] INFO: Crawled 438 pages (at 1920␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:58 [scrapy.extensions.logstats] INFO: Crawled 470 pages (at 1920␣
↪pages/min), scraped 0 items (at 0 items/min)
2016-12-16 21:18:59 [scrapy.core.engine] INFO: Closing spider (closespider_timeout)
2016-12-16 21:18:59 [scrapy.extensions.logstats] INFO: Crawled 518 pages (at 2880␣
↪pages/min), scraped 0 items (at 0 items/min)
```

(次のページに続く)

(前のページからの続き)

```
2016-12-16 21:19:00 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 229995,
 'downloader/request_count': 534,
 'downloader/request_method_count/GET': 534,
 'downloader/response_bytes': 1565504,
 'downloader/response_count': 534,
 'downloader/response_status_count/200': 534,
 'finish_reason': 'closespider_timeout',
 'finish_time': datetime.datetime(2016, 12, 16, 16, 19, 0, 647725),
 'log_count/INFO': 17,
 'request_depth_max': 19,
 'response_received_count': 534,
 'scheduler/dequeued': 533,
 'scheduler/dequeued/memory': 533,
 'scheduler/enqueued': 10661,
 'scheduler/enqueued/memory': 10661,
 'start_time': datetime.datetime(2016, 12, 16, 16, 18, 49, 799869)}
2016-12-16 21:19:00 [scrapy.core.engine] INFO: Spider closed (closespider_timeout)
```

これは、Scrapy を実行するあなたのハードウェアで 1 分あたり約 3000 ページをクロールできることを示しています。 これは、リンクをたどることを目的とした非常に単純なスパイダーであることに注意してください。作成するカスタム・スパイダーは、おそらくより多くの処理を行うため、クロール速度が遅くなります。 どれだけ遅くなるかは、スパイダーがどれだけ行うかと、それがどれだけうまく書かれているかに依存します。

将来的には、他の一般的なシナリオをカバーするために、ベンチマーキング・スイートにさらにケースが追加されるでしょう。

5.13 ジョブ制御: クロールの一時停止と再開

大きなサイトでは、クロールを一時停止して後で再開できることが望ましい場合があります。

Scrapy は、以下の機構を提供することにより、この機能をいち早くサポートしています:

- スケジュールされたリクエストをディスクに保持するスケジューラー
- 訪問したリクエストをディスク上に保持する重複フィルター
- バッチ間でいくつかのスパイダーの状態 (キー/値のペア) を保持する拡張機能

5.13.1 Job ディレクトリ

永続性サポートを有効にするには、 `JOBDIR` 設定で **job ディレクトリ** を定義するだけです。このディレクトリは、単一のジョブ (スパイダー実行など) の状態を維持するために必要なすべてのデータを保存するためのもので

す。このディレクトリは、**単一ジョブ** の状態を保存するために使用されるため、異なるスパイダー、または同じスパイダーの異なる job/run でさえ共有してはならないことに注意することが重要です。

5.13.2 使い方

永続性サポートを有効にしてスパイダーを起動するには、次のように実行します:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

その後、いつでも (Ctrl-C を押すかシグナルを送信することにより) スパイダーを安全に停止し、同じコマンドを発行して後で再開できます:

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

5.13.3 バッチ間で永続的な状態を維持する

時々、一時停止/再開バッチ間で永続的なスパイダー状態を維持したい場合があります。 それには `spider.state` 属性を使用できます。これは辞書である必要があります。スパイダーが開始および停止するときに、ジョブ・ディレクトリからの属性のシリアル化、保存、および読み込みを処理する組み込みの拡張機能があります。

スパイダー状態を使用するコールバックの例を次に示します (簡潔にするため、他のスパイダー・コードは省略されています):

```
def parse_item(self, response):
    # parse item here
    self.state['items_count'] = self.state.get('items_count', 0) + 1
```

5.13.4 永続性サポートあるある

Scrapy 永続性サポートを使用できるようにしたい場合、留意すべきことがいくつかあります:

クッキーの有効期限

Cookie の有効期限が切れる場合があります。 そのため、スパイダーをすぐに再開しないと、スケジュールされたリクエストが機能しなくなる可能性があります。 クモがクッキーに依存していない場合、これは問題になりません。

リクエストのシリアル化

永続性が機能するためには、リクエストは pickle モジュールによってシリアル化可能でなければならないため、リクエストがシリアル化可能であることを確認する必要があります。

ここで最も一般的な問題は、永続化できないリクエスト・コールバックで lambda 関数を使用することです。

たとえば、これは機能しません:

```
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: %s" % somearg)
```

しかし、これは動作します:

```
def some_callback(self, response):
    somearg = 'test'
    return scrapy.Request('http://www.example.com',
                          callback=self.other_callback, cb_kwargs={'somearg': somearg})

def other_callback(self, response, somearg):
    print("the argument passed is: %s" % somearg)
```

シリアル化できなかったリクエストをログに記録したい場合、プロジェクトの設定ページで *SCHEDULER_DEBUG* 設定を True に設定できます。デフォルトでは False です。

F.A.Q.(よくある質問と回答) よくある質問とその回答。

スパイダーのデバッグ scrapy スパイダーの一般的な問題をデバッグする方法を学びます。

スパイダー コントラクト スパイダーをテストするためにコントラクト (訳注: @url などの @ で始まる、テスト用に用意された属性指定子) を使用する方法を学びます。

よくある例 Scrapy のいくつかの一般的な慣例に親しみます。

広範なクロール 多数のドメインを並行してクロールするための Scrapy のチューニング。

***Web* ブラウザの開発ツールを使ってスクレイピングする** Web ブラウザ付属の開発ツールを使用してスクレイプする方法を学びます。

動的に読み込まれたコンテンツの選択 動的にロードされる Web ページ・データを読み取ります。

メモリ・リークのデバッグ クローラーで、どのようにメモリ・リークを探して、そしてどのように取り除くかを学びます。

ファイルと画像のダウンロードおよび処理　あなたがスクレイプしたアイテムに関連するファイルや画像をダウンロードします。

スパイダーのデプロイ　リモート・サーバに、あなたの Scrapy スパイダーを配置し実行します。

AutoThrottle **拡張機能**　負荷に応じてクロール速度を動的に調整します。

ベンチマーキング　Scrapy があなたのハードウェアでどのように機能するかをチェックします。

ジョブ制御*: クロールの一時停止と再開*　大きなスパイダーのクロールを一時停止および再開する方法を学習します。

第 6 章

Scrapy の拡張

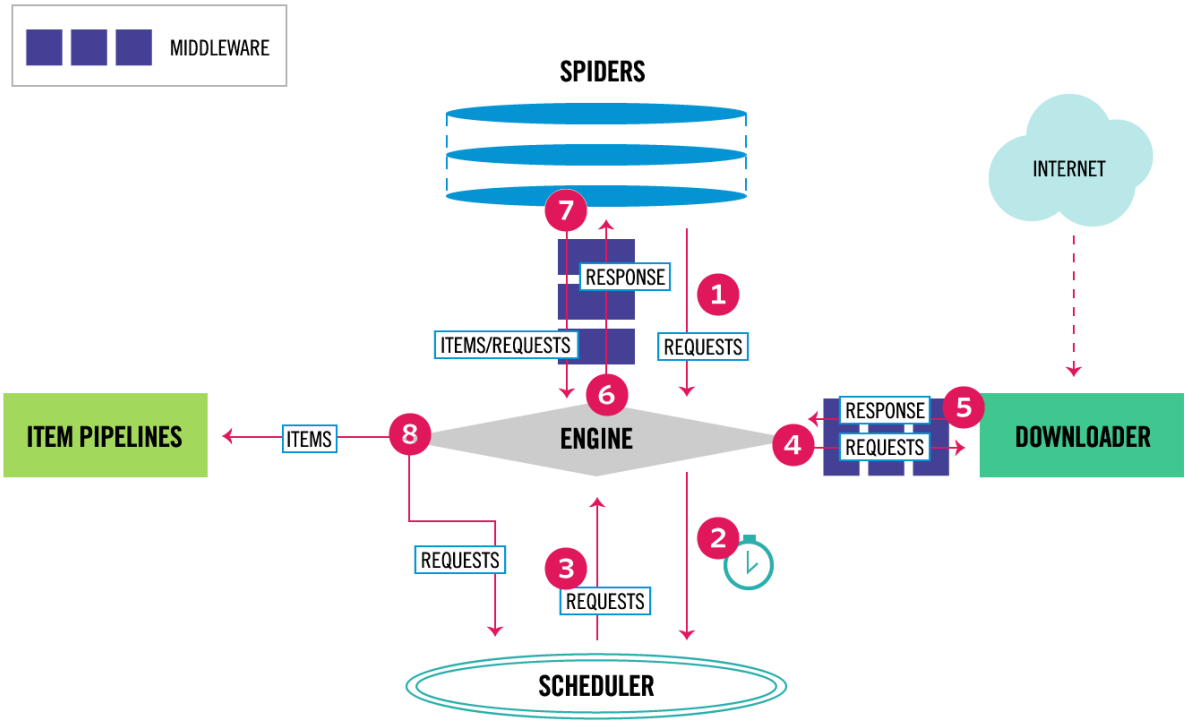
6.1 アーキテクチャ概観

この文書では、Scrapy のアーキテクチャとそのコンポーネントの相互作用について説明します。

6.1.1 概観

以下の図は、Scrapy アーキテクチャの概要とそのコンポーネント、およびシステム内で発生するデータフローの概要 (赤い矢印で表示) を示しています。コンポーネントの簡単な説明と、それらの詳細情報へのリンクを以下に示します。データフローについても以下で説明します。

6.1.2 データ・フロー



Scrapy のデータ・フローは実行エンジンによって制御され、次のようになります:

1. *Scrapy* **エンジン** は、 **スパイダー** からクロールする最初のリクエストを取得します。
2. *Scrapy* **エンジン** は、 **スケジューラ** でリクエストをスケジュールし、クロールする次のリクエストを求めます。
3. **スケジューラ** は、次のリクエストを *Scrapy* **エンジン** に返します。
4. *Scrapy* **エンジン** は、 **ダウンローダー** にリクエストを送信し、 **ダウンローダー・ミドルウェア** を通過します (`process_request()` 参照)。
5. ページのダウンロードが完了すると、 **ダウンローダー** は (そのページの) レスポンスを生成し、Scrapy エンジンに送信し、 **ダウンローダー・ミドルウェア** を通過します (`process_response()` 参照)。
6. *Scrapy* **エンジン** は **ダウンローダー** から Response を受け取り、それを **スパイダー** に送信して処理し、 **スパイダー・ミドルウェア** を通過します (see `process_spider_input()`)。
7. **スパイダー** はレスポンスを処理し、スクレイピングされたアイテムと (後に続く) 新しいリクエストを *Scrapy* **エンジン** に返し、 **スパイダー・ミドルウェア** を通過します (`process_spider_output()` 参照)。

8. *Scrapy* **エンジン** は処理済みのアイテムを **アイテム・パイプライン** に送信し、処理済みのリクエストを **スケジューラ** に送信し、可能なら次のクロール要求を求めます。

9. **スケジューラ** からの要求がなくなるまで、プロセスは (ステップ 1 から) 繰り返されます。

6.1.3 コンポーネント

Scrapy エンジン

エンジンは、システムのすべてのコンポーネント間のデータ・フローを制御し、特定のアクションが発生したときにイベントをトリガーします。詳細については、上記の **データ・フロー** 節を参照してください。

スケジューラ

スケジューラは Scrapy エンジンからリクエストを受信し、後で Scrapy エンジンが求めたときにリクエストをキューに入れて (Scrapy エンジンにも) 送信します。

ダウンローダー

ダウンローダーは Web ページを取得し、それらを Scrapy エンジンに送り、Scrapy エンジンがそれらをスパイダーに送ります。

スパイダー

スパイダーは、Scrapy ユーザーによって記述されたカスタム・クラスであり、レスポンスをパースし、それらからアイテム (またはスクレイピング・アイテム) を抽出するか、追跡するための追加のリクエストを行います。詳細については、 **スパイダー** を参照してください。

アイテム・パイプライン

アイテム・パイプラインは、スパイダーによってアイテムが抽出 (またはスクレイピング) された後にアイテムを処理する役割を果たします。典型的なタスクには、クレンジング、検証、永続化 (データベースへのアイテムの保存など) が含まれます。詳細については、 **アイテム・パイプライン** を参照してください。

ダウンローダー・ミドルウェア

ダウンローダー・ミドルウェアは、エンジンとダウンローダーの間にある特定のフックであり、エンジンからダウンローダーに渡されるときにリクエストを処理し、ダウンローダーからエンジンに渡されるリクエストを処理します。

あなたが以下のいずれかを行う必要がある場合は、ダウンローダー・ミドルウェアを使用します:

- ダウンローダーに送信される直前にリクエストを処理します (Scrapy がウェブサイトにリクエストを送信する直前);
- スパイダーに渡す前に受信したリクエストを変更する;
- 受信したレスポンスをスパイダーに渡すのではなく、新しいリクエストを送信します;
- Web ページを取得せずにスパイダーにレスポンスを渡す;
- いくつかのリクエストを黙って廃棄する。

詳細は **ダウンローダー・ミドルウェア** 参照。

スパイダー・ミドルウェア

スパイダー・ミドルウェアは、エンジンとスパイダーの間にある特定のフックであり、スパイダーの入力 (レスポンス) と出力 (アイテムとリクエスト) を処理できます。

あなたが必要な場合はスパイダー・ミドルウェアを使用します

- スパイダー・コールバックのポスト・プロセス出力 - リクエストまたはアイテムの変更/追加/削除;
- start_requests の後処理;
- スパイダー例外を処理します;
- レスポンス内容に基づいて、一部のリクエストに対してコールバックの代わりに偉ー・バック (errback) を呼び出します。

詳細については **スパイダー・ミドルウェア** 参照。

6.1.4 イベント駆動型ネットワーキング

Scrapy は、Python 用の人気のあるイベント駆動型ネットワーク・フレームワークである Twisted を使って記述されています。したがって、並行性のために非ブロッキング (別名「非同期」) コードを使用して実装されています。

非同期プログラミングと Twisted の詳細については、これらのリンクを参照してください:

- Twisted の遅延オブジェクトの紹介 (Introduction to Deferreds in Twisted)
- Twisted - こんにちは、非同期プログラミング (Twisted - hello, asynchronous programming)
- Twisted 紹介 - Krondo(Twisted Introduction - Krondo)

6.2 ダウンローダー・ミドルウェア

ダウンローダー・ミドルウェアは、Scrapy のリクエスト/レスポンス処理へのフックのフレームワークです。Scrapy のリクエストとレスポンスをグローバルに変更するための軽量で低レベルのシステムです。

6.2.1 ダウンローダーミドルウェアをアクティブにする

ダウンローダー・ミドルウェア・コンポーネントを有効にするには、それを *DOWNLOADER_MIDDLEWARES* 設定に追加します。これは、キーがミドルウェア・クラス・パスであり、値がミドルウェアの順序である辞書です。

ここに例があります:

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
}
```

DOWNLOADER_MIDDLEWARES 設定は、Scrapy で定義された *DOWNLOADER_MIDDLEWARES_BASE* 設定とマージされ (オーバーライドされることはありません)、有効なミドルウェアの最終ソートリストを取得するために順序でソートされます。1 つはエンジンに近く、最後はダウンローダーに近いものです。つまり、各ミドルウェアの *process_request()* メソッドは、ミドルウェアの昇順 (100、200、300…) で呼び出され、各ミドルウェアの *process_response()* メソッドは、降順で呼び出されます。

ミドルウェアに割り当てる順序を決定するには、 *DOWNLOADER_MIDDLEWARES_BASE* 設定を参照し、ミドルウェアを挿入する場所に応じて値を選択します。各ミドルウェアは異なるアクションを実行し、ミドルウェアは適用される以前の (または後続の) ミドルウェアに依存する可能性があるため、順序は重要です。

組み込みのミドルウェア (*DOWNLOADER_MIDDLEWARES_BASE* で定義され、デフォルトで有効になっているミドルウェア) を無効にするには、プロジェクトの *DOWNLOADER_MIDDLEWARES* 設定で定義し、その値として None を割り当てる必要があります。たとえば、ユーザーエージェントミドルウェアを無効にする場合、以下の通りです:

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
}
```

最後に、特定の設定で一部のミドルウェアを有効にする必要がある場合があることに注意してください。詳細については、各ミドルウェアのドキュメントを参照してください。

6.2.2 あなた自身のダウンローダー・ミドルウェアを書く

各ダウンローダー・ミドルウェアは、以下で定義される 1 つ以上のメソッドを定義する Python クラスです。

メインのエントリー・ポイントは from_crawler クラス・メソッドで、これは *Crawler* インスタンスを受け取ります。 *Crawler* オブジェクトは、たとえば **設定** へのアクセスを提供します。

class scrapy.downloadermiddlewares.**DownloaderMiddleware**

注釈: どのダウンローダー・ミドルウェア・メソッドも遅延オブジェクト (deferred) を返す場合があります。

---

process_request(*request*, *spider*)

このメソッドは、ダウンロード・ミドルウェアを通過するリクエストごとに呼び出されます。

process_request() の結果は次のいずれかでなければなりません: None を返す、 *Response* オブジェクトを返す、 *Request* オブジェクトを返す、 *IgnoreRequest* 例外を起こす。

None を返した場合、Scrapy はこのリクエストの処理を続け、最終的に適切なダウンローダー・ハンドラーが実行されたリクエスト (およびダウンロードされたレスポンス）が呼ばれるまで、他のすべてのミドルウェアを実行します。

Response オブジェクトを返す場合、Scrapy は他の **任意の** *process_request()* メソッドまたは *process_exception()* メソッド、または適切なダウンロード関数の呼び出しを考慮しません。そのレスポンスオブジェクトそのものを返します。なお、インストールされたミドルウェアの *process_response()* メソッドは、すべてのレスポンスで常に呼び出されます。

Request オブジェクトを返す場合、Scrapy は process_request メソッドの呼び出しを停止し、返されたリクエストを再スケジュールします。 新しく返されたリクエストが実行されると、ダウンロードされたレスポンスで適切なミドルウェア・チェーンが呼び出されます。

IgnoreRequest 例外が発生した場合、インストールされたダウンローダー・ミドルウェアの *process_exception()* メソッドが呼び出されます。それらのいずれもが例外を処理しない場合、リクエストの errback 関数 (Request.errback) が呼び出されます。発生した例外を処理するコードがない場合、無視され、(他の例外とは異なり) ログに記録されません。

パラメータ

- **request** (*Request* object) – 処理中のリクエスト
- **spider** (*Spider* object) – このリクエストの対象となるスパイダー

process_response(*request*, *response*, *spider*)

process_response() は次のいずれかの結果でなければなりません: *Response* オブジェクトを返す、 *Request* オブジェクトを返す、 *IgnoreRequest* 例外を起こす。

Response が返された場合 (指定されたレスポンスと同じか、まったく新しいレスポンスである可能性があります)、そのレスポンスは次のチェーン内のミドルウェアの *process_response()* で引き続き処理されます。

Request オブジェクトを返す場合、ミドルウェア・チェーンは停止し、返されたリクエストは将来ダウンロードされるように再スケジュールされます。これは、リクエストが *process_request()* から返される場合と同じ動作です。

IgnoreRequest 例外が発生した場合、リクエストの errback 関数 (`Request.errback`) が呼び出されます。発生した例外を処理するコードがない場合、無視され、(他の例外とは異なり) ログに記録されません。

パラメータ

- **request** (is a *Request* object) – レスポンスを引き起こしたリクエスト
- **response** (*Response* object) – 処理中のレスポンス
- **spider** (*Spider* object) – このレスポンスを意図したスパイダー

process_exception(*request*, *exception*, *spider*)

Scrapy は、ダウンロード・ハンドラーまたは、(ダウンローダー・ミドルウェアから) *process_request()* が例外 (*IgnoreRequest* 例外を含む) を発生させると、*process_exception()* を呼び出します

process_exception() は、 `None` または *Response* オブジェクトまたは *Request* オブジェクトを返す必要があります。

`None` を返す場合、Scrapy はこの例外の処理を続け、ミドルウェアが無くなりデフォルトの例外処理が開始されるまで、順にインストールされた他のミドルウェアの *process_exception()* メソッドを実行します。

Response オブジェクトを返す場合、インストールされたミドルウェアの *process_response()* メソッド・チェーンが開始され、Scrapy は他のミドルウェアの *process_exception()* メソッドを呼び出すことはありません。

Request オブジェクトを返す場合、返されたリクエストは将来ダウンロードされるように再スケジュールされます。これは、レスポンスを返すのと同様に、ミドルウェアの *process_exception()* メソッドの実行を停止します。

パラメータ

- **request** (is a *Request* object) – 例外を生成したリクエスト
- **exception** (an `Exception` object) – 発生した例外
- **spider** (*Spider* object) – このリクエストの対象となるスパイダー

from_crawler(*cls*, *crawler*)

存在する場合、このクラス・メソッドは *Crawler* からミドルウェア・インスタンスを作成するために呼び出されます。ミドルウェアの新しいインスタンスを返す必要があります。クローラー・オブジェク

トは、設定や信号などのすべての Scrapy コアコンポーネントへのアクセスを提供します。それはミドルウェアがそれらにアクセスし、その機能を Scrapy にフックする方法です。

パラメータ **crawler** (*Crawler* object) – このミドルウェアを使用するクローラー

6.2.3 組み込みダウンローダー・ミドルウェア・リファレンス

この文書では、Scrapy に付属するすべてのダウンローダー・ミドルウェア・コンポーネントについて説明します。それらの使用方法と独自のダウンローダ・ミドルウェアの作成方法については、**ダウンローダミドルウェア使用ガイド** を参照してください。

デフォルトで有効になっているコンポーネント (およびその順序) のリストについては、*DOWNLOADER_MIDDLEWARES_BASE* 設定を参照してください。

CookiesMiddleware

class scrapy.downloadermiddlewares.cookies.**CookiesMiddleware**

このミドルウェアにより、セッションを使用するサイトなど、クッキーを必要とするサイトを操作できます。Web サーバーが送信したクッキーを追跡し、Web ブラウザーが行うように、その後の (スパイダーからの) リクエストでそれらを送り返します。

次の設定を使用して、クッキー・ミドルウェアを構成 (configure) できます:

- *COOKIES_ENABLED*
- *COOKIES_DEBUG*

スパイダーごとの複数のクッキー・セッション

バージョン 0.15 で追加.

cookiejar リクエスト・メタ・キーを使用することにより、スパイダーごとに複数のクッキー・セッションの保持をサポートします。デフォルトでは、単一の cookie jar(セッション) を使用しますが、異なるものを使用するために識別子を渡すことができます。

例えば:

```
for i, url in enumerate(urls):
    yield scrapy.Request(url, meta={'cookiejar': i},
        callback=self.parse_page)
```

cookiejar メタ・キーは "sticky" ではないことに注意してください (訳注:つまり volatile)。 後続のリクエストでそれを渡し続ける必要があります:

```
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

デフォルト: True

クッキー・ミドルウェアを有効にするかどうか。 無効にすると、クッキーは Web サーバーに送信されません。

Request.*meta['dont_merge_cookies']* が True と評価される場合、 *COOKIES_ENABLED* 設定にかかわらず、リクエス・トクッキーは Web サーバーに **送信されません** 。そして *Response* で受信されたクッキーは、既存のクッキーとは **マージされません** 。

詳細については、 *Request* の cookies パラメーターを参照してください。

COOKIES_DEBUG

デフォルト: False

有効にすると、Scrapy はリクエストで送信されたすべてのクッキー (Cookie ヘッダー) およびレスポンスで受信されたすべてのクッキー (Set-Cookie ヘッダー) をログに記録します。

COOKIES_DEBUG が有効になっているログの例を次に示します:

```
2011-04-06 14:35:10-0300 [scrapy.core.engine] INFO: Spider opened
2011-04-06 14:35:10-0300 [scrapy.downloadermiddlewares.cookies] DEBUG: Sending cookies␣
↪to: <GET http://www.diningcity.com/netherlands/index.html>
        Cookie: clientlanguage_nl=en_EN
2011-04-06 14:35:14-0300 [scrapy.downloadermiddlewares.cookies] 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 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://www.
↪diningcity.com/netherlands/index.html> (referer: None)
[...]
```

DefaultHeadersMiddleware

class scrapy.downloadermiddlewares.defaultheaders.**DefaultHeadersMiddleware**

このミドルウェアは、 *DEFAULT_REQUEST_HEADERS* 設定で指定されたすべてのデフォルト・リクエス

ト・ヘッダーを設定します。

DownloadTimeoutMiddleware

class scrapy.downloadermiddlewares.downloadtimeout.**DownloadTimeoutMiddleware**

このミドルウェアは、 *DOWNLOAD_TIMEOUT* 設定または download_timeout スパイダー属性で指定されたリクエストのダウンロード・タイムアウトを設定します。

注釈: *download_timeout* リクエスト・メタ・キーを使用して、リクエストごとにダウンロード・タイムアウトを設定することもできます。これは、DownloadTimeoutMiddleware が無効になっている場合でもサポートされます。

HttpAuthMiddleware

class scrapy.downloadermiddlewares.httpauth.**HttpAuthMiddleware**

このミドルウェアは、 Basic access authentication (別名 BASIC 認証) を使用して、特定のスパイダーから生成されたすべてのリクエストを認証します。

特定のスパイダーからの BASIC 認証を有効にするには、それらのスパイダーの http_user と http_pass 属性を設定します。

例:

```
from scrapy.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.downloadermiddlewares.httpcache.**HttpCacheMiddleware**

このミドルウェアは、すべての HTTP リクエストおよびレスポンスに低レベルのキャッシュを提供します。キャッシュ・ストレージ・バックエンドおよびキャッシュ・ポリシーと組み合わせる必要があります。

Scrapy には 3 つの HTTP キャッシュ・ストレージ・バックエンドが付属しています:

- **ファイルシステム・ストレージ・バックエンド (デフォルト)**

- *DBM* ストレージ・バックエンド
- *LevelDB* ストレージ・バックエンド

HTTPCACHE_STORAGE 設定で HTTP キャッシュ・ストレージ・バックエンドを変更できます。また、 独自のストレージ・バックエンドの実装 もできます。

Scrapy には 2 つの HTTP キャッシュ・ポリシーが付属しています:

- *RFC2616* ポリシー
- ダミー・ポリシー *(*デフォルト*)*

HTTPCACHE_POLICY 設定で HTTP キャッシュ・ポリシーを変更できます。または、独自のポリシーを実装することもできます。 *dont_cache* メタ・キーが `True` に等しいことを使用して、すべてのポリシーで応答をキャッシュしないようにすることもできます。

ダミー・ポリシー (デフォルト)

class scrapy.extensions.httpcache.**DummyPolicy**

このポリシーは、HTTP キャッシュ制御ディレクティブを認識しません。すべてのリクエストとそれに対応するレスポンスはキャッシュされます。同じリクエストが再び現れるると、インターネットから何も転送せずにレスポンスが返されます。

ダミー・ポリシーは、スパイダーをより高速にテストする (毎回ダウンロードを待つ必要なしに) ときや、インターネット接続が利用できないときにスパイダーをオフラインで試すのに役立ちます。設計目標は、スパイダーの実行を前に実行したとおりに再現できるようにすることです。

RFC2616 ポリシー

class scrapy.extensions.httpcache.**RFC2616Policy**

このポリシーは、RFC2616 準拠の HTTP キャッシュ、つまり HTTP Cache-Control 認識を提供します。これは、新たな生成物に狙いを定め、変更されていないデータのダウンロードを回避するために連続実行で使用されます (帯域幅を節約し、クロールを高速化するため)。

実装されているもの:

- `no-store` キャッシュ制御ディレクティブが設定されている場合、レスポンス/リクエストを保存しようとしないでください。
- `no-cache` キャッシュ制御ディレクティブが新しいレスポンスに対しても設定されている場合、キャッシュからレスポンスを提供しません
- `max-age` キャッシュ制御ディレクティブから鮮度寿命を計算します
- `Expires` レスポンス・ヘッダーから鮮度寿命を計算します

- Last-Modified レスポンス・ヘッダーから鮮度の有効期間を計算します (Firefox で使用されるヒューリスティック)
- Age レスポンス・ヘッダーから現在の年齢を計算する
- Date ヘッダーから現在の年齢を計算
- Last-Modified レスポンス・ヘッダーに基づいて古いレスポンスを再検証します
- ETag レスポンス・ヘッダーに基づいて古いレスポンスを再検証します
- 受信したレスポンスがない場合に Date ヘッダーを設定します
- リクエストで max-stale キャッシュ制御ディレクティブをサポート

これにより、スパイダーを完全な RFC2616 キャッシュ・ポリシーで構成できますが、HTTP 仕様に準拠したまま、リクエストごとの再検証を回避できます。

例:

Cache-Control: max-stale=600 をリクエスト・ヘッダーに追加して、600 秒以内ならば有効期限を超えたレスポンスを受け入れます。

RFC2616, 14.9.3 も参照下さい。

実装されてないもの:

- Pragma: no-cache サポート https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1
- Vary ヘッダー・サポート https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.6
- 更新または削除後の無効化 https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.10
- …その他いろいろ…

ファイルシステム・ストレージ・バックエンド (デフォルト)

class scrapy.extensions.httpcache.**FilesystemCacheStorage**

ファイルシステム・ストレージ・バックエンドは、HTTP キャッシュ・ミドルウェアで使用できます。

各リクエスト/レスポンスのペアは、次のファイルを含む異なるディレクトリに保存されます:

- request_body - 生のリクエスト・ボディそのもの
- request_headers - リクエスト・ヘッダ (生 HTTP 書式)
- response_body - 生のレスポンス・ボディそのもの
- response_headers - リクエスト・ヘッダ (生 HTTP 書式)
- meta - Python repr() 形式の、このキャッシュ・リソースのメタ・データ (grep に優しい形式)

- `pickled_meta` - ‘‘ meta‘‘ と同じメタデータですが、より効率的な逆シリアル化のために直列化 (pickled) されています

ディレクトリ名はリクエストのフィンガー・プリントから作成され (`scrapy.utils.request.fingerprint` 参照)、1 つのレベルのサブ・ディレクトリを使用して、同じディレクトリに多くのファイルを作成しないようにします (多くのファイルシステムでは非効率的です)。ディレクトリの例以下です:

```
/path/to/cache/dir/example.com/72/72811f648e718090f041317756c03adb0ada46c7
```

DBM ストレージ・バックエンド

class `scrapy.extensions.httpcache.`**DbmCacheStorage**

バージョン 0.13 で追加.

DBM ストレージ・バックエンドは、HTTP キャッシュ・ミドルウェアでも使用できます。

デフォルトでは、anydbm モジュールを使用しますが、 *HTTPCACHE_DBM_MODULE* 設定で変更できます。

LevelDB ストレージ・バックエンド

class `scrapy.extensions.httpcache.`**LeveldbCacheStorage**

バージョン 0.23 で追加.

LevelDB ストレージ・バックエンドは、HTTP キャッシュ・ミドルウェアでも使用できます。

LevelDB データベースに同時にアクセスできるプロセスは 1 つだけなので、このバックエンドは開発にはお勧めできません。そのため、同じスパイダーに対してクロールを実行して Scrapy シェルを並行して開くことはできません。

このストレージ・バックエンドを使用するには、 LevelDB python bindings をインストールします (例: `pip install leveldb`)。

あなた自身のストレージ・バックエンドを書く

以下に説明するメソッドを定義する Python クラスを作成することにより、キャッシュス・トレージ・バックエンドを実装できます。

class `scrapy.extensions.httpcache.`**CacheStorage**

open_spider(*spider*)

このメソッドは、クロールのためにスパイダーがオープンされた後に呼び出されます。 *open_spider* シグナルを処理します。

パラメータ **spider** (*Spider* object) – オープンされたスパイダー

close_spider(*spider*)
このメソッドは、スパイダーがクローズされた後に呼び出されます。 *close_spider* シグナルを処理します。

パラメータ **spider** (*Spider* object) – クローズされたスパイダー

retrieve_response(*spider*, *request*)
キャッシュに存在する場合はレスポンスを返し、そうでない場合は None を返します。

パラメータ

- **spider** (*Spider* object) – リクエストを生成したスパイダー
- **request** (*Request* object) – キャッシュされたレスポンスを見つけるためのリクエスト

store_response(*spider*, *request*, *response*)
与えられたレスポンスをキャッシュに保存します。

パラメータ

- **spider** (*Spider* object) – このレスポンスを意図したスパイダー
- **request** (*Request* object) – スパイダーが生成した対応するリクエスト
- **response** (*Response* object) – キャッシュに保存するレスポンス

ストレージ・バックエンドを使用するには、以下の設定をします:

- *HTTPCACHE_STORAGE* をあなたのカスタム・ストレージ・クラスの Python インポート・パスに設定します。

HTTPCache middleware 設定

HttpCacheMiddleware は次の設定で構成 (configure) できます:

HTTPCACHE_ENABLED

バージョン 0.11 で追加.

デフォルト: False

HTTP キャッシュを有効にするかどうか。

バージョン 0.11 で変更: バージョン 0.11 より前は、 *HTTPCACHE_DIR* がキャッシュを有効にするのに使われていました。

HTTPCACHE_EXPIRATION_SECS

デフォルト: 0

キャッシュされたリクエストの有効期限 (秒単位)。

この時間より古いキャッシュされたリクエストは再ダウンロードされます。ゼロの場合、キャッシュされたリクエストは期限切れになりません。

バージョン 0.11 で変更: バージョン 0.11 より前は、ゼロは、キャッシュされたリクエストが常に期限切れになることを意味しました。

HTTPCACHE_DIR

デフォルト: `'httpcache'`

(低レベル)HTTP キャッシュを保存するために使用するディレクトリ。空の場合、HTTP キャッシュは無効になります。相対パスが指定されている場合、プロジェクト・データ・ディレクトリからの相対パスが使用されます。詳細については、 *Scrapy* **プロジェクトのデフォルト構造** を参照してください。

HTTPCACHE_IGNORE_HTTP_CODES

バージョン 0.10 で追加.

デフォルト: `[]`

これらの HTTP コードでレスポンスをキャッシュしないでください。

HTTPCACHE_IGNORE_MISSING

デフォルト: `False`

有効にすると、キャッシュに見つからないリクエストはダウンロードされずに無視されます。

HTTPCACHE_IGNORE_SCHEMES

バージョン 0.10 で追加.

デフォルト: `['file']`

これらの URI スキームでレスポンスをキャッシュしないでください。

HTTPCACHE_STORAGE

デフォルト: `'scrapy.extensions.httpcache.FilesystemCacheStorage'`

キャッシュ・ストレージ・バックエンドを実装するクラス。

HTTPCACHE_DBM_MODULE

バージョン 0.13 で追加.

デフォルト: 'anydbm'

DBM ストレージ・バックエンド で使用するデータベース・モジュール。この設定は、DBM バックエンド固有です。

HTTPCACHE_POLICY

バージョン 0.18 で追加.

デフォルト: 'scrapy.extensions.httpcache.DummyPolicy'

キャッシュ・ポリシーを実装するクラス。

HTTPCACHE_GZIP

バージョン 1.0 で追加.

デフォルト: False

有効にすると、キャッシュされたすべてのデータが gzip で圧縮されます。この設定は、ファイルシステム・バックエンド固有です。

HTTPCACHE_ALWAYS_STORE

バージョン 1.1 で追加.

デフォルト: False

有効にすると、ページを無条件にキャッシュします。

スパイダーは、たとえば Cache-Control：max-stale で将来使用するために、すべてのレスポンスをキャッシュで利用可能にしたい場合があります。 DummyPolicy はすべてのレスポンスをキャッシュしますが、それらを再検証することはありません。また、より微妙なポリシーが望ましい場合があります。

この設定は、レスポンスの Cache-Control：no-store ディレクティブを引き続き尊重します。不要な場合は、キャッシュ・ミドルウェアにフィードするレスポンスの Cache-Control ヘッダーから no-store をフィルターします。

HTTPCACHE_IGNORE_RESPONSE_CACHE_CONTROLS

バージョン 1.1 で追加.

デフォルト: []

無視されるレスポンスのキャッシュ制御ディレクティブのリスト。

サイトは "no-store"、"no-cache"、"must-revalidate"などを設定することがよくありますが、これらのディレクティブを尊重するとスパイダーが生成できる取引 (traffic) をろうばいさせます。この設定により、クロールされるサイ

トにとって重要ではないことがわかっている Cache-Control ディレクティブを選択的に無視できます。

スパイダーは、実際に必要な場合を除き、リクエストで Cache-Control ディレクティブを発行しないため、リクエストのディレクティブはフィルタリングされません。

HttpCompressionMiddleware

class scrapy.downloadermiddlewares.httpcompression.**HttpCompressionMiddleware**

このミドルウェアにより、圧縮 (gzip、deflate) 取引 (traffic) を Web サイトから送受信できます。

このミドルウェアは、 brotlipy がインストールされている場合、 brotli-compressed レスポンスのデコードもサポートします。

HttpCompressionMiddleware 設定

COMPRESSION_ENABLED

デフォルト: `True`

圧縮ミドルウェアを有効にするかどうか。

HttpProxyMiddleware

バージョン 0.8 で追加.

class scrapy.downloadermiddlewares.httpproxy.**HttpProxyMiddleware**

このミドルウェアは、 *Request* オブジェクトに `proxy` メタ値を設定することにより、リクエストに使用する HTTP プロキシを設定します。

Python 標準ライブラリモジュール urllib および urllib2 と同様に、以下の環境変数に従います:

- `http_proxy`
- `https_proxy`
- `no_proxy`

リクエストごとにメタ・キー `proxy` を `http://some_proxy_server:port` または `http://username:password@some_proxy_server:port` のような値に設定することもできます。この値は `http_proxy` / `https_proxy` 環境変数よりも優先され、また `no_proxy` 環境変数も無視することに注意してください。

RedirectMiddleware

class scrapy.downloadermiddlewares.redirect.**RedirectMiddleware**

このミドルウェアは、レスポンス・ステータスに基づいてリクエストのリダイレクトを処理します。

(リダイレクト中に) リクエストが通過する URL は *Request.meta* の redirect_urls キーで見つけることができます。 *redirect_urls* の各リダイレクトの理由は、 *Request.meta* の redirect_reasons キーにあります。例: [301, 302, 307, 'meta refresh']

理由の形式は、対応するリダイレクトを処理したミドルウェアによって異なります。 たとえば、 *RedirectMiddleware* はトリガーとなったレスポンス・ステータ・スコードを整数で示しますが、 *MetaRefreshMiddleware* は常に 'meta refresh' 文字列を理由として使用します。

RedirectMiddleware は次の設定で設定できます (詳細については設定ドキュメントを参照してください):

- *REDIRECT_ENABLED*
- *REDIRECT_MAX_TIMES*

Request.meta の dont_redirect キーが True に設定されている場合、リクエストはこのミドルウェアによって無視されます。

スパイダーでいくつかのリダイレクト・ステータス・コードを処理したい場合、これらを handle_httpstatus_list スパイダー属性で指定できます。

たとえば、リダイレクト・ミドルウェアで、301 と 302 レスポンスを無視する (およびそれらをスパイダーに渡す) 場合は、以下のようにします:

```
class MySpider(CrawlSpider):
    handle_httpstatus_list = [301, 302]
```

Request.meta の handle_httpstatus_list キーは、リクエストごとに許可するレスポンス・コードを指定するためにも使用できます。リクエストに対するレスポンス・コードを許可したい場合、メタ・キー handle_httpstatus_all を True に設定することもできます。

RedirectMiddleware 設定

REDIRECT_ENABLED

バージョン 0.13 で追加.

デフォルト: True

リダイレクト・ミドルウェアを有効にするかどうか。

REDIRECT_MAX_TIMES

デフォルト: `20`

1 回のリクエストで追跡されるリダイレクトの最大数。

MetaRefreshMiddleware

class scrapy.downloadermiddlewares.redirect.**MetaRefreshMiddleware**

このミドルウェアは、メタ・リフレッシュ html タグに基づいてリクエストのリダイレクトを処理します。

MetaRefreshMiddleware は次の設定で設定できます (詳細については設定ドキュメントを参照してください):

- *METAREFRESH_ENABLED*
- *METAREFRESH_IGNORE_TAGS*
- *METAREFRESH_MAXDELAY*

このミドルウェアは、 *RedirectMiddleware* で説明されているように、 :*REDIRECT_MAX_TIMES* 設定と *dont_redirect* リクエスト・メタ・キーと *redirect_urls* リクエスト・メタ・キーと *redirect_reasons* リクエスト・メタ・キーに従います。

MetaRefreshMiddleware 設定

METAREFRESH_ENABLED

バージョン 0.17 で追加.

デフォルト: `True`

メタ・リフレッシュ・ミドルウェアを有効にするかどうか。

METAREFRESH_IGNORE_TAGS

デフォルト: `['script', 'noscript']`

これらのタグ内のメタ・タグは無視されます。

METAREFRESH_MAXDELAY

デフォルト: `100`

リダイレクト後の最大メタリフレッシュ遅延 (秒単位)。一部のサイトでは、セッションの期限切れページへのリダイレクトにメタ・リフレッシュを使用しているため、自動リダイレクトを最大遅延に制限しています。

RetryMiddleware

class scrapy.downloadermiddlewares.retry.**RetryMiddleware**

接続タイムアウトや HTTP 500 エラーなどの一時的な問題が原因である可能性のある失敗した要求を再試行するミドルウェア。

スパイダーがすべての通常の (失敗していない) ページのクロールを完了すると、失敗したページはスクレイピング・プロセスで収集され、最後に再スケジュールされます。

RetryMiddleware は次の設定で設定できます (詳細については設定ドキュメントを参照):

- *RETRY_ENABLED*
- *RETRY_TIMES*
- *RETRY_HTTP_CODES*

Request.meta の dont_retry キーが True に設定されている場合、リクエストはこのミドルウェアによって無視されます。

RetryMiddleware 設定

RETRY_ENABLED

バージョン 0.13 で追加.

デフォルト: True

再試行ミドルウェアを有効にするかどうか。

RETRY_TIMES

デフォルト: 2

(最初のダウンロードに加えて) 再試行する最大回数。

再試行の最大回数は、 *Request.meta* の *max_retry_times* 属性を使用してリクエストごとに指定することもできます。初期化されると、 *max_retry_times* メタ・キーは *RETRY_TIMES* 設定よりも優先されます。

RETRY_HTTP_CODES

デフォルト: [500, 502, 503, 504, 522, 524, 408, 429]

再試行する HTTP レスポンス・コード。その他のエラー (DNS ルックアップの問題、接続の切断など) は常に再試行されます。

場合によっては、400 を *RETRY_HTTP_CODES* に追加することもできます。これは、サーバーの過負荷を示すために使用される一般的なコードだからです。HTTP の仕様ではそうなっているため、デフォルトでは含まれていません。

RobotsTxtMiddleware

class scrapy.downloadermiddlewares.robotstxt.**RobotsTxtMiddleware**

このミドルウェアは、robots.txt 除外標準で禁止されているリクエストを除外します。

Scrapy が robots.txt を尊重するようにするには、ミドルウェアが有効になっており、かつ、*ROBOTSTXT_OBEY* 設定が有効になっていることを確認してください。

ROBOTSTXT_USER_AGENT 設定を使用して、robots.txt ファイルでの照合に使用するユーザー・エージェント文字列を指定できます。 None の場合、リクエストで送信している User-Agent ヘッダーまたは *USER_AGENT* 設定 (この順序で) は、robots.txt ファイルで使用するユーザー・エージェントを決定するために使用されます。

このミドルウェアは robots.txt パーサと組み合わせる必要があります。

Scrapy には、次の robots.txt パーサがサポートされています:

- *RobotFileParser* (デフォルト)
- *Reppy*
- *Robotexclusionrulesparser*
- *Protego*

robots.txt パーサは、 *ROBOTSTXT_PARSER* 設定で変更できます。または、 **新しいパーサ** のサポートを実装することもできます。

Request.meta の dont_obey_robotstxt キーが True に設定されている場合、 *ROBOTSTXT_OBEY* が有効になっていても、このミドルウェアは要求を無視します。

RobotFileParser

RobotFileParser は Python に組み込まれている robots.txt パーサです。 パーサは、Martijn Koster の 1996 年のドラフト仕様 に完全に準拠しています。 ワイルド・カード・マッチングのサポートはありません。Scrapy はデフォルトでこのパーサを使用します。

このパーサーを使用するには、以下を設定します:

- *ROBOTSTXT_PARSER* 設定に scrapy.robotstxt.PythonRobotParser をセット

Robotexclusionrulesparser

Robotexclusionrulesparser は、 Martijn Koster's 1996 draft specification に完全に準拠しており、サポートされています。ワイルド・カード・マッチング用です。

このパーサーを使用するには:

- `pip install robotexclusionrulesparser` を実行して、Robotexclusionrulesparser をインストールします
- *ROBOTSTXT_PARSER* 設定に `scrapy.robotstxt.RerpRobotParser` をセットします

Reppy パーサ

Reppy は、 Robots Exclusion Protocol Parser for C++ の Python ラッパーです。 パーサーは、 Martijn Koster's 1996 draft specification に完全に準拠しており、ワイルド・カード・マッチングをサポートしています。 RobotFileParser や `Robotexclusionrulesparser <http://nikitathespider.com/python/rerp/>`_ **とは異なり、特に、** ``Allow と `Disallow` ディレクティブでは、パスの長さに基づいた最も具体的なルールがより具体的ではない (より短い) ルールより優先される、長さベースのルールを使用します。

このパーサーを使用するには:

- `pip install reppy` を実行して Reppy をインストールします
- *ROBOTSTXT_PARSER* 設定に `scrapy.robotstxt.ReppyRobotParser` をセットします

Protego パーサ

Protego は、純粋な Python robots.txt パーサです。パーサは Google's Robots.txt Specification に完全に準拠しているため、ワイルドカード・マッチングをサポートし、 Reppy に類似した長さベースのルールを使用します。

このパーサーを使用するには:

- `pip install protego` を実行して Protego をインストールします
- *ROBOTSTXT_PARSER* 設定に `scrapy.robotstxt.ProtegoRobotParser` をセットします

新しいパーサのサポートの実装

抽象基本クラス `RobotParser` をサブクラス化し、以下に説明するメソッドを実装することにより、新しい robots.txt パーサーのサポートを実装できます。

DownloaderStats

class scrapy.downloadermiddlewares.stats.**DownloaderStats**
通過するすべてのリクエストとレスポンスと例外の統計を保存するミドルウェア。

このミドルウェアを使用するには、 *DOWNLOADER_STATS* 設定を有効にする必要があります。

UserAgentMiddleware

class scrapy.downloadermiddlewares.useragent.**UserAgentMiddleware**
スパイダーがデフォルトのユーザ・エージェントをオーバーライドできるようにするミドルウェア。

スパイダーがデフォルトのユーザー・エージェントを上書きするには、その `user_agent` 属性を設定する必要があります。

AjaxCrawlMiddleware

class scrapy.downloadermiddlewares.ajaxcrawl.**AjaxCrawlMiddleware**
メタ・フラグメント HTML タグに基づいて「AJAX クロール可能な」ページ・バリアントを検出するミドルウェア。 詳細については、https：//developers.google.com/webmasters/ajax-crawling/docs/getting-started をご覧ください。

注釈: Scrapy は、このミドルウェアがなくても、「 'http://example.com/!#foo=bar'」などの URL の「AJAX クロール可能な」ページを検出します。AjaxCrawlMiddleware は、URL に `'!#'` が含まれていない場合に必要です。これは多くの場合、 'index' または 'main' の Web サイトページの場合です。

AjaxCrawlMiddleware 設定

AJAXCRAWL_ENABLED

バージョン 0.21 で追加.

デフォルト: `False`

AjaxCrawlMiddleware を有効にするかどうか。 *broad crawls* に対して有効にすることをお勧めします。

HttpProxyMiddleware 設定

HTTPPROXY_ENABLED

デフォルト: `True`

HttpProxyMiddleware を有効にするかどうか。

HTTPPROXY_AUTH_ENCODING

デフォルト: "latin-1"

HttpProxyMiddleware のプロキシ認証のデフォルトのエンコーディング。

6.3 スパイダー・ミドルウェア

スパイダー・ミドルウェアは、Scrapy のスパイダー処理メカニズムへのフックのフレームワークであり、カスタム機能をプラグインして、処理のために **スパイダー** に送信されるレスポンスを処理し、スパイダーから生成されたリクエストとアイテムを処理できます。

6.3.1 スパイダー・ミドルウェアをアクティブにする

スパイダー・ミドルウェア・コンポーネントをアクティブにするには、それを *SPIDER_MIDDLEWARES* 設定に追加します。これは、キーがミドルウェア・クラス・パスであり、値がミドルウェアの順序値である辞書です。

以下に例があります:

```
SPIDER_MIDDLEWARES = {
    'myproject.middlewares.CustomSpiderMiddleware': 543,
}
```

SPIDER_MIDDLEWARES 設定は、Scrapy で定義された *SPIDER_MIDDLEWARES_BASE* 設定とマージされ (オーバーライドされることはありません)、有効なミドルウェアの最終ソート・リストを取得するために順序値の昇順にソートされます。最初がエンジンに近い方でスパイダーに近い方が最後です。 つまり、各ミドルウェアの *process_spider_input()* メソッドは、ミドルウェアの昇順 (100、200、300、…) で呼び出され、各ミドルウェアの *process_spider_output()* メソッドは、降順に呼び出されます。

ミドルウェアに割り当てる順序を決定するには、 *SPIDER_MIDDLEWARES_BASE* 設定を参照し、ミドルウェアを挿入する場所に応じて値を選択します。 各ミドルウェアは異なるアクションを実行し、ミドルウェアは適用される以前の (または後続の) ミドルウェアに依存する可能性があるため、順序は重要です。

組み込みミドルウェア (*SPIDER_MIDDLEWARES_BASE* で定義され、デフォルトで有効になっているミドルウェア) を無効にする場合は、プロジェクトの *SPIDER_MIDDLEWARES* 設定で定義し、その値として None を割り当てる必要があります。たとえば、オフサイト・ミドルウェアを無効にする場合は次の通りです:

```
SPIDER_MIDDLEWARES = {
    'myproject.middlewares.CustomSpiderMiddleware': 543,
    'scrapy.spidermiddlewares.offsite.OffsiteMiddleware': None,
}
```

最後に、特定の設定で一部のミドルウェアを有効にする必要がある場合があることに注意してください。 詳細については、各ミドルウェアのドキュメントを参照してください。

6.3.2 あなた自身のスパイダー・ミドルウェアを書く

各スパイダー・ミドルウェアは、以下で定義される 1 つ以上のメソッドを定義する Python クラスです。

メインのエントリーポイントは `from_crawler` クラス・メソッドで、これは *Crawler* インスタンスを受け取ります。 *Crawler* オブジェクトは、たとえば **設定** へのアクセスを提供します。

class scrapy.spidermiddlewares.**SpiderMiddleware**

process_spider_input(*response*, *spider*)

このメソッドは、処理のためにスパイダー・ミドルウェアを通過してスパイダーに送信される各レスポンスに対して呼び出されます。

process_spider_input() は None を返すか、例外を発生させます。

None を返す場合、Scrapy はこのレスポンスの処理を続行し、最後にレスポンスがスパイダーに渡されて処理されるまで、他のすべてのミドルウェアを実行します。

例外が発生した場合、Scrapy は他のスパイダーミドルウェアの *process_spider_input()* を呼び出さず、リクエストがある場合はリクエストの errback を呼び出し、そうでない場合は *process_spider_exception()* チェーンを開始します。errback の出力は、*process_spider_output()* が処理するために別の方向に戻される (chain back) か、または、例外がを発生した場合は *process_spider_exception()* に戻り (chain back) ます。

パラメータ

- **response** (*Response* object) – 処理中のレスポンス
- **spider** (*Spider* object) – このレスポンスを意図したスパイダー

process_spider_output(*response*, *result*, *spider*)

このメソッドは、レスポンスを処理した後、Spider から返された結果で呼び出されます。

process_spider_output() は、 *Request* または辞書または *Item* オブジェクトの反復可能オブジェクト (iterable) を返す必要があります。

パラメータ

- **response** (*Response* object) – スパイダーからこの出力を生成したレスポンス
- **result** (an iterable of *Request*, dict or *Item* objects) – スパイダーによって返された結果
- **spider** (*Spider* object) – 結果が処理されているスパイダー

process_spider_exception(*response*, *exception*, *spider*)

このメソッドは、スパイダーまたは、(以前のスパイダー・ミドルウェアの) *process_spider_output()* メソッドが例外を発生させたときに呼び出されます。

process_spider_exception() は None または *Request* または辞書または *Item* オブジェクトの反復可能オブジェクト (iterable) のいずれかを返す必要があります。

None が返された場合、Scrapy はこの例外の処理を続行し、処理するミドルウェア・コンポーネントが無くなってエンジンに到達するまで、続くミドルウェア・コンポーネントで *process_spider_exception()* を実行します。

反復可能オブジェクト (iterable) を返す場合、 *process_spider_output()* パイプラインは次のスパイダー・ミドルウェアから開始され、他の *process_spider_exception()* は呼び出されません。

パラメータ

- **response** (*Response* object) – 例外が発生したときに処理されているレスポンス
- **exception** (Exception object) – 発生した例外
- **spider** (*Spider* object) – 例外を発生させたスパイダー

process_start_requests(*start_requests*, *spider*)

バージョン 0.15 で追加.

このメソッドは、スパイダーの開始リクエストで呼び出され、レスポンスが関連付けられておらず、リクエストのみ (アイテムではなく) を返す必要があることを除いて、 *process_spider_output()* メソッドと同様に機能します。

(start_requests パラメーターで) 反復可能オブジェクト (iterable) を受け取り、別の *Request* オブジェクトの反復可能オブジェクト (iterable) を返さなければなりません。

注釈: スパイダー・ミドルウェアでこのメソッドを実装する場合、(入力に従って) 常に反復可能オブジェクトを返す必要があり、start_requests イテレータを消費しないでください。 Scrapy エンジンは、リクエスト開始を処理する能力がある間はリクエスト開始求を呼ぶように設計されているため、リクエスト開始イテレータは、スパイダーを停止するための他の条件 (時間制限やアイテム/ページ数など) がある場合、事実上無限になります。

パラメータ

- **start_requests** (an iterable of *Request*) – リクエストの開始
- **spider** (*Spider* object) – 開始したリクエストが属するスパイダー

from_crawler(*cls*, *crawler*)

存在する場合、このクラスメソッドは *Crawler* からミドルウェア・インスタンスを作成するために呼び出されます。ミドルウェアの新しいインスタンスを返す必要があります。クローラー・オブジェクトは、設定や信号などのすべての Scrapy コアコンポーネントへのアクセスを提供します。それはミドルウェアがそれらにアクセスし、その機能を Scrapy にフックする方法です。

パラメータ **crawler** (*Crawler* object) – このミドルウェアを使用するクローラー

6.3.3 組み込みのスパイダー・ミドルウェア・リファレンス

この文書では、Scrapy に付属するすべてのスパイダー・ミドルウェア・コンポーネントについて説明します。それらの使用方法と独自のスパイダー・ミドルウェアの作成方法については、 **スパイダーミドルウェア使用ガイド** を参照してください。

デフォルトで有効になっているコンポーネント (およびその順序) のリストについては、*SPIDER_MIDDLEWARES_BASE* 設定を参照してください。

DepthMiddleware

class scrapy.spidermiddlewares.depth.**DepthMiddleware**

DepthMiddleware は、スクレイピングされるサイト内の各リクエストの深さを追跡するために使用されます。以前に値が設定された事がない場合は、`request.meta['depth'] = 0` を設定し (通常は最初のリクエストのみ)、それ以外の場合は 1 インクリメントします。

スクレイピングする最大深度を制限したり、深度に基づいてリクエストの優先度を制御したりすることができます。

DepthMiddleware は次の設定で設定できます (詳細については設定ドキュメントを参照してください):

- *DEPTH_LIMIT* - 任意のサイトでクロールできる最大深度。ゼロの場合、制限は課されません。
- *DEPTH_STATS_VERBOSE* - 各深さのレベルでリクエスト数を収集するかどうか。
- *DEPTH_PRIORITY* - 深さに基づいてリクエストに優先順位を付けるかどうか。

HttpErrorMiddleware

class scrapy.spidermiddlewares.httperror.**HttpErrorMiddleware**

失敗した (誤った)HTTP レスポンスをフィルター処理して、スパイダーがそれらに対処する必要がないようにします。これにより、(ほとんどの場合) オーバーヘッドが発生し、より多くのリソースが消費され、スパイダー・ロジックがより複雑になります。

HTTP standard によると、成功したレスポンスとは、ステータスコードが 200〜300 の範囲です。

それでもその範囲外のレスポンス・コードを処理したい場合は、 handle_httpstatus_list スパイダー属性または *HTTPERROR_ALLOWED_CODES* 設定を使用して、スパイダーが処理できるレスポンス・コードを指定できます。

たとえば、スパイダーに 404 レスポンスを処理させたい場合、以下を行うことができます:

```
class MySpider(CrawlSpider):
    handle_httpstatus_list = [404]
```

Request.meta の handle_httpstatus_list キーは、リクエストごとに許可するレスポンス・コードを指定するためにも使用できます。リクエストに対するレスポンス・コードを許可したい場合、メタ・キー handle_httpstatus_all を True に設定することもできます。

ただし、自分が何をしているのか本当にわかっていない限り、200 以外の応答を処理することは通常良くない考えです。

詳細情報は HTTP Status Code Definitions を参照ください。

HttpErrorMiddleware 設定

HTTPERROR_ALLOWED_CODES

デフォルト: []

このリストに含まれる、200 以外のステータ・スコードを持つすべてのレスポンスを渡します。

HTTPERROR_ALLOW_ALL

デフォルト: False

ステータスコードに関係なく、すべてのレスポンスを渡します。

OffsiteMiddleware

class scrapy.spidermiddlewares.offsite.**OffsiteMiddleware**

スパイダーが対象とするドメインから外れている URL のリクエストを除外します。

このミドルウェアは、スパイダーの allowed_domains 属性にない全てのホスト名のリクエストを除外します。なお、リスト内のドメインのすべてのサブドメインも許可されます。 例えば、ルール www.example.org は bob.www.example.org も許可しますが、www2.example.com も example.com も許可しません。

スパイダーがカバーするドメインに属していないドメインへのリクエストをスパイダーが返すと、このミドルウェアは、以下に似たデバッグ・メッセージを記録します:

```
DEBUG: Filtered offsite request to 'www.othersite.com': <GET http://www.othersite.
↪com/some/page.html>
```

ログが過剰なノイズでいっぱいになるのを避けるため、フィルターされた新しいドメインごとにこれらのメッセージの 1 つのみを出力します。そのため、たとえば、 www.othersite.com への別のリクエストがフィルタリングされた場合、ログメッセージは出力されません。 しかし、 someothersite.com へのリクエストがフィルターされると、メッセージが出力されます (ただし、フィルターされる最初のリクエストのみ)。

スパイダーが *allowed_domains* 属性を定義していない場合、または属性が空の場合、オフサイト・ミドルウェアはすべてのリクエストを許可します。

リクエストに dont_filter 属性が設定されている場合、そのドメインが許可されたドメインにリストされていなくても、オフサイト・ミドルウェアはリクエストを許可します。

RefererMiddleware

class scrapy.spidermiddlewares.referer.**RefererMiddleware**

リクエストを生成したレスポンスの URL に基づいて、リクエスト Referer ヘッダーを生成します。

RefererMiddleware 設定

REFERER_ENABLED

バージョン 0.15 で追加.

デフォルト: True

リファラー・ミドルウェアを有効にするかどうか。

REFERRER_POLICY

バージョン 1.4 で追加.

デフォルト: 'scrapy.spidermiddlewares.referer.DefaultReferrerPolicy' リクエストの "Referer" ヘッダーを設定するときに適用する Referrer Policy

---

注釈: *Request.meta* の特別な "referrer_policy" キーを使用して、 REFERRER_POLICY 設定と同じ許容値を使用して、リクエストごとにリファラー・ポリシーを設定することもできます。

REFERRER_POLICY が受け入れる値

- `scrapy.spidermiddlewares.referer.ReferrerPolicy` サブクラスへのパス - カスタム・ポリシーまたは組み込みポリシーのいずれか (以下のクラスを参照)
- または、標準の W3C 定義の文字列値のいずれか
- または特別な `"scrapy-default"` 。

文字列値	クラス名 (文字列)
`"scrapy-default"` (デフォルト)	`scrapy.spidermiddlewares.referer.DefaultReferrerPolicy`
"no-referrer"	`scrapy.spidermiddlewares.referer.NoReferrerPolicy`
"no-referrer-when-downgrade"	`scrapy.spidermiddlewares.referer.NoReferrerWhenDowngradePolicy`
"same-origin"	`scrapy.spidermiddlewares.referer.SameOriginPolicy`
"origin"	`scrapy.spidermiddlewares.referer.OriginPolicy`
"strict-origin"	`scrapy.spidermiddlewares.referer.StrictOriginPolicy`
"origin-when-cross-origin"	`scrapy.spidermiddlewares.referer.OriginWhenCrossOriginPolicy`
"strict-origin-when-cross-origin"	`scrapy.spidermiddlewares.referer.StrictOriginWhenCrossOriginPolicy`
"unsafe-url"	`scrapy.spidermiddlewares.referer.UnsafeUrlPolicy`

警告: " no-referrer-when-downgrade " のように、ブラウザの W3C 推奨値である、Scrapy のデフォルトのリファラー・ポリシーは、ドメインが異なっていても、 任意の `http(s)://` から空でない "Referer" ヘッダーをすべての `https://` に送信します。

クロス・ドメイン・リクエストのリファラー情報を削除する場合は、 "same-origin" の方が適している場合があります。

注釈: "no-referrer-when-downgrade" ポリシーは W3C 推奨のデフォルトであり、主要な Web ブラウザーで使用されます。

ただし、それは Scrapy のデフォルトのリファラー・ポリシーではありません (`DefaultReferrerPolicy` を参照)。

警告: "unsafe-url" ポリシーは **推奨されません** 。

UrlLengthMiddleware

class scrapy.spidermiddlewares.urllength.**UrlLengthMiddleware**

URLLENGTH_LIMIT より長い URL を持つリクエストを除外します

UrlLengthMiddleware は次の設定で構成 (configure) できます (詳細については設定ドキュメントをご覧ください):

- *URLLENGTH_LIMIT* - クロールする URL の最大長。

6.4 拡張機能

拡張フレームワークは、独自のカスタム機能を Scrapy に挿入するメカニズムを提供します。

拡張機能は、拡張機能が初期化されるときに、Scrapy の起動時にインスタンス化される単なる通常のクラスです。

6.4.1 拡張機能の設定

拡張機能は *Scrapy* **設定** を使用して、他の Scrapy コードと同様に設定を管理します。

既存の (および将来の) 拡張機能との衝突を避けるために、拡張機能が独自の名前を設定の前に付けるのが慣例です。たとえば、 Google Sitemaps を処理する仮想拡張機能では、 GOOGLESITEMAP_ENABLED や GOOGLESITEMAP_DEPTH などの設定を使用します。

6.4.2 拡張機能の読み込みとアクティブ化

拡張機能は、拡張機能クラスの単一インスタンスをインスタンス化することにより、起動時にロードおよびアクティブ化されます。したがって、すべての拡張機能初期化コードはクラス・コンストラクター (__init__ メソッド) で実行する必要があります。

拡張機能を使用可能にするには、Scrapy 設定の *EXTENSIONS* 設定に追加します。 *EXTENSIONS* では、各拡張機能は文字列 (拡張機能のクラス名への完全な Python パス) で表されます。例えば以下のようにします:

```
EXTENSIONS = {
    'scrapy.extensions.corestats.CoreStats': 500,
    'scrapy.extensions.telnet.TelnetConsole': 500,
}
```

ご覧のとおり、 *EXTENSIONS* 設定はキーが拡張パスであり、その値が拡張機能 **読み込み** 順序値を定義する辞書です。 *EXTENSIONS* 設定は、Scrapy で定義された *EXTENSIONS_BASE* 設定とマージされ (但し、オーバーライドされることはありません)、有効な拡張機能の最終ソート・リストを取得するために順序値でソートされます。

通常、拡張機能は相互に依存しないため、ほとんどの場合、読み込み順序は無関係です。 これが *EXTENSIONS_BASE* 設定がすべての拡張機能を同じ順序値 (`0`) で定義する理由です。ただし、すでに読み込まれているされている他の拡張機能に依存する拡張機能を追加する必要がある場合、この機能を利用できます。

6.4.3 利用可能な、デフォルトで有効およびデフォルトで無効な拡張機能

利用可能なすべての拡張機能が有効になるわけではありません。それらのいくつかは通常、特定の設定に依存しています。たとえば、HTTP キャッシュ拡張機能はデフォルトで使用可能ですが、 *HTTPCACHE_ENABLED* 設定が設定されていない限り無効になっています。

6.4.4 拡張機能を無効にする

デフォルトで有効になっている拡張機能 (つまり、 *EXTENSIONS_BASE* 設定に含まれている拡張機能) を無効にするには、その順序値を `None` に設定する必要があります。 例えば以下のようにします:

```
EXTENSIONS = {
    'scrapy.extensions.corestats.CoreStats': None,
}
```

6.4.5 あなた自身の拡張機能を書く

各拡張機能は Python クラスです。 Scrapy 拡張機能の主なエントリ・ポイント (これにはミドルウェアとパイプラインも含まれます) は、 `Crawler` インスタンスを受け取る `from_crawler` クラスメソッドです。Crawler オブジェクトを使用して、設定、信号、統計にアクセスしたり、クロール動作を制御したりできます。

通常、拡張機能は **シグナル** に接続し、それらによってトリガーされるタスクを実行します。

最後に、 `from_crawler` メソッドが *NotConfigured* 例外を発生させた場合、拡張機能は無効になります。それ以外の場合、拡張機能は有効になります。

拡張機能例

ここでは、前のセクションで説明した概念を説明するために、簡単な拡張機能を実装します。この拡張機能は毎回メッセージを記録します:

- スパイダーがオープンされます
- スパイダーがクローズされます

- 指定の数のアイテムがスクレイプされます

拡張機能は MYEXT_ENABLED 設定で有効になり、アイテムの数は MYEXT_ITEMCOUNT 設定で指定されます。

そのような拡張機能のコードは次のとおりです:

```
import logging
from scrapy import signals
from scrapy.exceptions import NotConfigured

logger = logging.getLogger(__name__)

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):
        logger.info("opened spider %s", spider.name)

    def spider_closed(self, spider):
        logger.info("closed spider %s", spider.name)

    def item_scraped(self, item, spider):
        self.items_scraped += 1
        if self.items_scraped % self.item_count == 0:
            logger.info("scraped %d items", self.items_scraped)
```

6.4.6 組み込み拡張機能リファレンス

汎用拡張機能

ログ統計拡張機能

class scrapy.extensions.logstats.**LogStats**

クロールされたページやスクレイプされたアイテムなどの基本的な統計情報を記録します。

コア統計拡張機能

class scrapy.extensions.corestats.**CoreStats**

統計コレクションが有効になっている場合、コア統計のコレクションを有効にします (**統計をとる** 参照)。

Telnet コンソール拡張機能

class scrapy.extensions.telnet.**TelnetConsole**

現在実行中の Scrapy プロセス内の Python インタープリターに入るための Telnet コンソールを提供します。これはデバッグに非常に役立ちます。

telnet コンソールは *TELNETCONSOLE_ENABLED* 設定を有効にする必要があり、サーバーは *TELNETCONSOLE_PORT* で指定されたポートでリッスンします。

メモリ使用量の拡張機能

class scrapy.extensions.memusage.**MemoryUsage**

注釈: この拡張機能は Windows では機能しません。

スパイダーを実行する Scrapy プロセスが使用するメモリを監視し、そして:

1. 特定の値を超えたときに通知メールを送信します
2. 特定の値を超えたときにスパイダーを閉じます

MEMUSAGE_WARNING_MB が特定の警告値に達し、かつ、 *MEMUSAGE_LIMIT_MB* が最大値に達すると通知メールがトリガーされ、スパイダーが閉じられて Scrapy プロセスが終了 (terminate) します。

この拡張機能は *MEMUSAGE_ENABLED* 設定によって有効になり、以下設定で構成 (configure) できます:

- *MEMUSAGE_LIMIT_MB*

- *MEMUSAGE_WARNING_MB*
- *MEMUSAGE_NOTIFY_MAIL*
- *MEMUSAGE_CHECK_INTERVAL_SECONDS*

メモリ・デバッガー拡張機能

class scrapy.extensions.memdebug.**MemoryDebugger**

メモリ使用量をデバッグするための拡張機能。 以下に関する情報を収集します:

- Python ガベージコレクターによって収集されないオブジェクト
- 生存しているべきではないオブジェクト。詳細は *trackref* を使用したメモリ・リークのデバッグ 参照。

この拡張機能を有効にするには、 *MEMDEBUG_ENABLED* 設定をオンにします。情報は統計に保存されます。

スパイダー拡張機能を閉じる

class scrapy.extensions.closespider.**CloseSpider**

各条件に特定の終了理由を使用して、いくつかの条件が満たされたときにスパイダーを自動的に閉じます。

スパイダーを閉じるための条件は、以下の設定で構成 (configure) できます:

- *CLOSESPIDER_TIMEOUT*
- *CLOSESPIDER_ITEMCOUNT*
- *CLOSESPIDER_PAGECOUNT*
- *CLOSESPIDER_ERRORCOUNT*

CLOSESPIDER_TIMEOUT

デフォルト: 0

秒数を指定する整数。 スパイダーがその秒数を超えて開いたままの場合、理由を closespider_timeout として自動的に閉じられます。ゼロ (または設定されていない) の場合、スパイダーはタイムアウトによって閉じられません。

CLOSESPIDER_ITEMCOUNT

デフォルト: 0

アイテムの数を指定する整数。 スパイダーがその量より多くスクレイピングし、それらのアイテムがアイテム・パイプラインによって渡される場合、スパイダーは理由 closespider_itemcount によって閉じられます。現在

ダウンローダー・キューにあるリクエスト (*CONCURRENT_REQUESTS* リクエストまで) は引き続き処理されます。ゼロ (または未設定) の場合、スパイダーは渡されたアイテムの数によって制限され超えた分をクローズさせられる事はありません。

CLOSESPIDER_PAGECOUNT

バージョン 0.11 で追加.

デフォルト: `0`

クロールするレスポンスの最大数を指定する整数。 スパイダーがそれ以上クロールした場合、スパイダーは理由 ‘‘ closespider_pagecount‘‘ によって閉じられます。ゼロ (または未設定) の場合、クロールされたレスポンスの数に応じてスパイダーが閉じられることはありません。

CLOSESPIDER_ERRORCOUNT

バージョン 0.11 で追加.

デフォルト: `0`

スパイダーを閉じる前に受け取るエラーの最大数を指定する整数。 スパイダーがその数を超えるエラーを生成した場合、 `closespider_errorcount` の理由で閉じられます。ゼロ (または設定されていない) の場合、スパイダーはエラーの数に応じて閉じられることはありせん。

StatsMailer 拡張機能

class scrapy.extensions.statsmailer.**StatsMailer**

この単純な拡張機能を使用して、収集された Scrapy 統計など、ドメインがスクレイピングを完了するたびに通知メールを送信できます。 メールは、 *STATSMAILER_RCPTS* 設定で指定されたすべての受信者に送信されます。

拡張機能のデバッグ

拡張機能のスタックトレースをダンプする

class scrapy.extensions.debug.**StackTraceDump**

SIGQUIT または SIGUSR2 シグナルを受信したときに、実行中のプロセスに関する情報をダンプします。ダンプされる情報は次のとおりです:

1. エンジンの状態（ `scrapy.utils.engine.get_engine_status()` を使用)
2. 生存中の参照 (*trackref* **を使用したメモリ・リークのデバッグ** を参照)
3. 全てのスレッドのスタックトレース

スタック・トレースとエンジン・ステータスがダンプされた後、Scrapy プロセスは正常に実行を続けます。

SIGQUIT および SIGUSR2 シグナルは Windows では利用できないため、この拡張機能は POSIX 準拠のプラットフォーム (つまり、Windows ではない) でのみ機能します。

Scrapy に SIGQUIT シグナルを送信するには、少なくとも 2 つの方法があります:

1. Scrapy プロセスの実行中に Ctrl-を押す (Linux のみ?)
2. 以下のコマンドを実行 (`<pid>` が Scrapy プロセスのプロセス ID であるとして):

```
kill -QUIT <pid>
```

デバッガー拡張機能

class scrapy.extensions.debug.**Debugger**

SIGUSR2 シグナルを受信すると、実行中の Scrapy プロセス内で Python debugger を呼び出します。 デバッガーが終了した後、Scrapy プロセスは正常に実行を続けます。

詳細は Debugging in Python 参照。

この拡張機能は POSIX 準拠のプラットフォームでのみ機能します (つまり、Windows では機能しません)。

6.5 コア API

バージョン 0.15 で追加.

この節は、Scrapy コア API について説明します。これは、拡張機能とミドルウェアの開発者を対象としています。

6.5.1 クローラー API

Scrapy API の主要なエントリポイントは、 `from_crawler` クラス・メソッドを通じて拡張機能に渡される *`Crawler`* オブジェクトです。 このオブジェクトは、すべての Scrapy コア・コンポーネントへのアクセスを提供し、拡張機能がそれらにアクセスし、その機能を Scrapy にフックする唯一の方法です。

拡張機能マネージャーは、インストールされた拡張機能を読み込んで追跡する責任があり、利用可能な全ての拡張機能の辞書と、 **ダウンローダー・ミドルウェアの構成** *(configure)* 方法と類似した順序を含む *`EXTENSIONS`* 設定で構成 (configure) されます。

class scrapy.crawler.**Crawler**(*spidercls*, *settings*)

Crawler オブジェクトは、*`scrapy.spiders.Spider`* のサブクラスと `scrapy.settings.Settings` オブジェクトでインスタンス化する必要があります。

settings
このクローラーの設定マネージャ

これは、拡張機能とミドルウェアがこのクローラーの Scrapy 設定にアクセスするために使用します。

Scrapy 設定の概要については、**設定** を参照してください。

API については、 `Settings` クラスを参照してください。

signals
このクローラーのシグナル・マネージャ

これは、拡張機能およびミドルウェアが Scrapy 機能にフックするために使用されます。

シグナルの概要については、 **シグナル** を参照してください。

API については、 `SignalManager` クラスを参照してください。

stats
このクローラーの統計収集器

これは拡張機能とミドルウェアから使用され、その動作の統計を記録したり、他の拡張機能によって収集された統計にアクセスしたりします。

統計収集器の概要は **統計をとる** を参照下さい。

API については *`StatsCollector`* クラス参照。

extensions
有効な拡張機能を追跡 (track) する拡張機能マネージャ

ほとんどの拡張機能は、この属性にアクセスする必要はありません。

拡張機能の紹介と、Scrapy で利用可能な拡張機能のリストについては、 **拡張機能** を参照してください。

engine
スケジューラ、ダウンローダー、スパイダーの間のコア・クロール・ロジックを調整する実行エンジン。

一部の拡張機能では、Scrapy エンジンにアクセスして、ダウンローダーとスケジューラの動作を検査または変更することができますが、これは高度な使用方法であり、この API はまだ安定していません。

spider
現在スパイダーがクロールされています。これはクローラーの構築中に提供されるスパイダー・クラスのインスタンスであり、 *`crawl()`* メソッドで指定された引数の後に作成されます。

crawl(**args*, ***kwargs*)
指定された `args` 引数と `kwargs` 引数を使用してスパイダー・クラスをインスタンス化することでクローラーを起動し、実行エンジンを起動します。

クロールが終了したときに起動される遅延オブジェクトを返します。

6.5.2 API の設定

scrapy.settings.**SETTINGS_PRIORITIES**

Scrapy で使用されるデフォルト設定の優先度のキー名と優先度を設定する辞書。

各項目は設定エントリ・ポイントを定義し、識別のためのコード名と整数の優先度を与えます。 Settings クラスで値を設定および取得する場合、優先順位が高いほど順番値が小さくなります。

```
SETTINGS_PRIORITIES = {
    'default': 0,
    'command': 10,
    'project': 20,
    'spider': 30,
    'cmdline': 40,
}
```

各設定ソースの詳細な説明については、 **設定** を参照してください。

6.5.3 SpiderLoader API

class scrapy.spiderloader.**SpiderLoader**

このクラスは、プロジェクト全体で定義されたスパイダー・クラスの取得と処理を担当します。

SPIDER_LOADER_CLASS プロジェクト設定でパスを指定することにより、カスタム・スパイダー・ローダーを使用できます。エラーのない実行を保証するには、 scrapy.interfaces.ISpiderLoader インターフェースを完全に実装する必要があります。

from_settings(*settings*)

このクラスメソッドは、クラスのインスタンスを作成するために Scrapy によって使用されます。現在のプロジェクト設定で呼び出され、 *SPIDER_MODULES* 設定のモジュールで見つかったスパイダーを再帰的にロードします。

パラメータ **settings** (Settings instance) – プロジェクト設定

load(*spider_name*)

指定された名前の Spider クラスを取得します。 spider_name という名前のスパイダークラスの、以前にロードされたスパイダーを調べ、見つからない場合は KeyError を発生させます。

パラメータ **spider_name** (*str*) – スパイダー・クラス名

list()

プロジェクトで利用可能なスパイダーの名前を取得します。

find_by_request(*request*)

指定されたリクエストを処理できるスパイダーの名前をリストします。リクエストの URL をスパイダーのドメインと照合しようとします。

パラメータ **request** (*Request* instance) – クエリされたリクエスト

6.5.4 シグナル API

6.5.5 統計収集器 API

scrapy.statscollectors モジュールの下にいくつかの統計収集器があり、それらはすべて *StatsCollector* クラス (すべての継承元) で定義された統計収集器 API を実装します。

class scrapy.statscollectors.**StatsCollector**

get_value(*key*, *default=None*)
指定された統計キーの値を返します。値が存在しない場合はデフォルトを返します。

get_stats()
現在実行中のスパイダーからすべての統計を辞書として取得します。

set_value(*key*, *value*)
与えられた統計キーに指定の値を設定します。

set_stats(*stats*)
`stats` 引数で渡された辞書で現在の統計を上書きします。

inc_value(*key*, *count=1*, *start=0*)
(設定されていない場合は開始値を想定して、) 指定された統計キーの値を指定されたカウントでインクリメントします。

max_value(*key*, *value*)
同じキーの現在の値が value より小さい場合にのみ、指定されたキーに指定された値を設定します。指定されたキーに現在の値がない場合、値は常に設定されます。

min_value(*key*, *value*)
同じキーの現在の値が value より大きい場合にのみ、指定されたキーに指定された値を設定します。指定されたキーに現在の値がない場合、値は常に設定されます。

clear_stats()
全ての統計をクリアする

次のメソッドは、統計収集 API の一部ではありませんが、代わりにカスタム統計収集器を実装するときに使用されます:

open_spider(*spider*)
統計収集のために、指定されたスパイダーを開きます。

close_spider(*spider*)
指定されたスパイダーを閉じます。これが呼び出された後、これ以上特定の統計にアクセスしたり収集したりすることはできません。

6.6 シグナル

Scrapy は、特定のイベントが発生したときに通知するためにシグナルを広範囲に使用します。 Scrapy プロジェクトでこれらのシグナルの一部をキャッチして (たとえば **拡張機能** で)、追加のタスクを実行したり、Scrapy を拡張してすぐに使用できない機能を追加したりできます。

シグナルはいくつかの引数を提供しますが、それらをキャッチするハンドラーはそれらのすべてを受け入れる必要はありません。シグナル・ディスパッチ・メカニズムはハンドラーが受け取る引数のみを配信します。

あなたは **シグナル** *API* を介してシグナルに接続 (または独自に送信) できます。

シグナルをキャッチして何らかのアクションを実行する方法を示す簡単な例を次に示します:

```
from scrapy import signals
from scrapy import Spider


class DmozSpider(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/",
    ]


    @classmethod
    def from_crawler(cls, crawler, *args, **kwargs):
        spider = super(DmozSpider, cls).from_crawler(crawler, *args, **kwargs)
        crawler.signals.connect(spider.spider_closed, signal=signals.spider_closed)
        return spider


    def spider_closed(self, spider):
        spider.logger.info('Spider closed: %s', spider.name)


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

6.6.1 シグナル・ハンドラーの遅延 (deferred)

いくつかのシグナルは、ハンドラーから Twisted deferreds を返すことをサポートしています。それがどのシグナルか知るには、以下の **組み込みシグナル・リファレンス** を参照してください。

6.6.2 組み込みシグナル・リファレンス

Scrapy 組み込みシグナルとその意味のリストを以下に示します。

engine_started

scrapy.signals.**engine_started**()

Scrapy エンジンがクロールを開始すると送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

注釈: このシグナルは、スパイダーの起動方法に応じて、 *spider_opened* シグナルの **後に** 起動される場合があります。 そのため、 *spider_opened* の前にこのシグナルが発生することに依存しないでください。

---

engine_stopped

scrapy.signals.**engine_stopped**()

Scrapy エンジンが停止 (stop) したときに送信されます (たとえば、クロール・プロセスが終了したとき)。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

item_scraped

scrapy.signals.**item_scraped**(*item*, *response*, *spider*)

すべての **アイテム・パイプライン** ステージを (ドロップされることなく) 通過した後、アイテムがスクレイプされたときに送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

パラメータ

- **item** (dict or *Item* object) – スクレイプされたアイテム
- **spider** (*Spider* object) – アイテムをスクレイプしたスパイダー
- **response** (*Response* object) – アイテムがスクレイピングされたレスポンス

item_dropped

scrapy.signals.**item_dropped**(*item*, *response*, *exception*, *spider*)

あるステージで *DropItem* 例外が発生したときに、アイテムが **アイテム・パイプライン** からドロップされた後に送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

パラメータ

- **item** (dict or *Item* object) – **アイテム・パイプライン** からドロップされたアイテム
- **spider** (*Spider* object) – アイテムをスクレイプしたスパイダー
- **response** (*Response* object) – アイテムがドロップされたレスポンス
- **exception** (*DropItem* exception) – アイテムがドロップされる原因となった例外 (*DropItem* のサブクラスでなければなりません)

item_error

scrapy.signals.**item_error**(*item*, *response*, *spider*, *failure*)

DropItem 例外を除き、 **アイテム・パイプライン** がエラーを生成した (つまり、例外を発生させた) ときに送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

パラメータ

- **item** (dict or *Item* object) – **アイテム・パイプライン** からドロップされたアイテム
- **response** (*Response* object) – 例外が発生したときに処理されていたレスポンス
- **spider** (*Spider* object) – 例外を発生させたスパイダー
- **failure** (Failure object) – Twisted Failure オブジェクトとして発生した例外

spider_closed

scrapy.signals.**spider_closed**(*spider*, *reason*)

スパイダーが閉じられた後に送信されます。 これは、 *spider_opened* で予約しているスパイダーごとのリソースを解放するために使用できます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

パラメータ

- **spider** (*Spider* object) – スパイダーがクローズされた

- **reason** (*str*) – スパイダーが閉じられた理由を説明する文字列。 スパイダーがスクレイピングを完了したために閉じられた場合、その理由は `'finished'` です。そうでなければ、`close_spider` エンジン・メソッドを呼び出してスパイダーを手動で閉じた場合、その理由はそのメソッドの `reason` 数に渡されたものが使われます (デフォルトは `'cancelled'` です)。エンジンがシャットダウン (たとえば、Ctrl-C を押してエンジンを停止）された場合、理由は `'shutdown'` です。

spider_opened

scrapy.signals.**spider_opened**(*spider*)

クロールのためにスパイダーがオープンされた後に送信されます。これは通常、スパイダーごとのリソースを予約するために使用されますが、スパイダーが開かれたときに実行する必要があるタスクに使用できます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしています。

パラメータ **spider** (*Spider* object) – スパイダーがオープンされた

spider_idle

scrapy.signals.**spider_idle**(*spider*)

スパイダーがアイドル状態になったときに送信されます。つまり、スパイダーはそれ以降何もしない事を意味します:

- リクエストがダウンロード待ち
- リクエストがスケジュールされた
- アイテムがアイテム・パイプラインで処理中

このシグナルのすべてのハンドラーが終了した後もアイドル状態が続く場合、エンジンはスパイダーを閉じ始めます。スパイダーのクローズが完了すると、 *spider_closed* シグナルが送信されます。

あなたは *DontCloseSpider* 例外を発生させて、スパイダーが閉じられないようにすることができます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ **spider** (*Spider* object) – アイドルに移行したスパイダー

注釈: あなたの *spider_idle* ハンドラーでいくつかのリクエストをスケジュールすると、スパイダーが閉じられるのを防ぐことができるという保証はありませんが、できる場合もあります。これは、スケジュールされたすべてのリクエストがスケジューラによって拒否された場合 (たとえば、重複のためにフィルター処理された場合)、スパイダーがアイドル状態のままになる可能性があるためです。

---

spider_error

scrapy.signals.**spider_error**(*failure*, *response*, *spider*)

スパイダー・コールバックがエラーを生成する (つまり、例外を発生させる) ときに送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **failure** (Failure object) – Twisted Failure オブジェクトとして発生した例外
- **response** (*Response* object) – 例外が発生したときに処理されていたレスポンス
- **spider** (*Spider* object) – 例外を発生させたスパイダー

request_scheduled

scrapy.signals.**request_scheduled**(*request*, *spider*)

エンジンが *Request* をスケジュールしたときに送信され、後でダウンロードされます。

シグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **request** (*Request* object) – リクエストはスケジューラに到達した
- **spider** (*Spider* object) – スパイダーはリクエストを生成 (yield) した

request_dropped

scrapy.signals.**request_dropped**(*request*, *spider*)

後でダウンロードされるようにエンジンによってスケジュールされた *Request* がスケジューラーによって拒否されたときに送信されます。

シグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **request** (*Request* object) – リクエストはスケジューラに到達した
- **spider** (*Spider* object) – スパイダーはリクエストを生成 (yield) した

request_reached_downloader

scrapy.signals.**request_reached_downloader**(*request*, *spider*)

Request がダウンローダーに到達すると送信されます。

シグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **request** (*Request* object) – リクエストはダウンローダーに到達した
- **spider** (*Spider* object) – スパイダーはリクエストを生成 (yield) した

response_received

scrapy.signals.**response_received**(*response*, *request*, *spider*)

エンジンがダウンローダーから新しい *Response* を受信すると送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **response** (*Response* object) – レスポンスを受信した
- **request** (*Request* object) – レスポンスを生成したリクエスト
- **spider** (*Spider* object) – そのレスポンスを意図したスパイダー

response_downloaded

scrapy.signals.**response_downloaded**(*response*, *request*, *spider*)

HTTPResponse がダウンロードされた直後に、ダウンローダーによって送信されます。

このシグナルは、ハンドラーから遅延オブジェクト (deferred) を返すことをサポートしていません。

パラメータ

- **response** (*Response* object) – レスポンスがダウンロードされた
- **request** (*Request* object) – レスポンスを生成したリクエスト
- **spider** (*Spider* object) – そのレスポンスを意図したスパイダー

6.7 アイテム・エクスポーター

アイテムをスクレイピングしたら、他のアプリケーションでデータを使用するために、それらのアイテムを永続化またはエクスポートすることがよくあります。 つまり、結局のところ、それがスクレイピング・プロセス全体の目的です。

この目的のために、Scrapy は、XML、CSV、JSON などのさまざまな出力形式のアイテム・エクスポーターのコレクションを提供します。

6.7.1 アイテム・エクスポーターの使用

あなたが急いでいて、アイテム・エクスポータを使用してスクレイプ・データを出力するだけの場合は、 **フィード・エクスポート** を参照してください。それ以外の場合または、アイテム・エクスポータがどのように機能するかを知りたい場合または、またはより多くのカスタム機能 (デフォルトのエクスポートではカバーされてない機能) が必要な場合は、以下をお読みください。

アイテム・エクスポーターを使用するには、必要な引数でインスタンス化する必要があります。 各アイテム・エクスポーターには異なる引数が必要なため、 **組み込みアイテム・エクスポーター・リファレンス** で各エクスポーターのドキュメントを確認してください。エクスポーターをインスタンス化した後、以下の作業が必要です:

1. エクスポート・プロセスの開始を通知するために、メソッド `start_export()` を呼び出します。

2. エクスポートする各アイテムに対して *`export_item()`* メソッドを呼び出します

3. 最後に `finish_export()` を呼び出して、エクスポート・プロセスの終了を通知します

以下の **アイテム・パイプライン** をご覧ください。これは、複数のアイテム・エクスポーターを使用して、それらのフィールドの値に従って、スクレイプされたアイテムを異なるファイルにグループ化します:

```
from scrapy.exporters import XmlItemExporter

class PerYearXmlExportPipeline(object):
    """Distribute items across multiple XML files according to their 'year' field"""

    def open_spider(self, spider):
        self.year_to_exporter = {}

    def close_spider(self, spider):
        for exporter in self.year_to_exporter.values():
            exporter.finish_exporting()
            exporter.file.close()

    def _exporter_for_item(self, item):
        year = item['year']
        if year not in self.year_to_exporter:
            f = open('{}.xml'.format(year), 'wb')
            exporter = XmlItemExporter(f)
            exporter.start_exporting()
            self.year_to_exporter[year] = exporter
        return self.year_to_exporter[year]

    def process_item(self, item, spider):
        exporter = self._exporter_for_item(item)
        exporter.export_item(item)
        return item
```

6.7.2 アイテム・フィールドのシリアル化

デフォルトでは、フィールド値は変更されずに基礎となるシリアル化ライブラリに渡され、それらをシリアル化する方法の決定は特定の各シリアル化ライブラリに委任されます。

ただし、あなたは、各フィールド値をシリアル化する方法を、 **シリアル化ライブラリに渡す前の段階で** カスタマイズできます。

フィールドのシリアル化方法をカスタマイズするには、以下の 2 つの方法があります。

1. フィールドでシリアライザーを宣言する

あなたが *Item* を使用する場合、 **フィールド・メタ・データ** でシリアライザーを宣言できます。シリアライザーは、値を受け取り、シリアル化された形式を返す、呼び出し可能オブジェクトでなければなりません。

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. serialize_field() メソッドをオーバーライドする

あなたは *serialize_field()* メソッドをオーバーライドして、フィールド値のエクスポート方法をカスタマイズすることもできます。

カスタムコードの後に必ずベースクラス *serialize_field()* メソッドを呼び出してください。

例:

```
from scrapy.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)
```

6.7.3 組み込みアイテム・エクスポーター・リファレンス

Scrapy にバンドルされているアイテム・エクスポーターのリストを次に示します。それらの一部には、以下の 2 つのアイテムをエクスポートすることを想定した出力例が含まれています:

```
Item(name='Color TV', price='1200')
Item(name='DVD player', price='200')
```

BaseItemExporter

class scrapy.exporters.**BaseItemExporter**(*fields_to_export=None*, *export_empty_fields=False*, *encoding='utf-8'*, *indent=0*)

これは、すべてのアイテム・エクスポーターの (抽象的な) 基本クラスです。 エクスポートするフィールド、空のフィールドをエクスポートするか、使用するエンコードを定義するなど、すべての (具体的な) アイテム・エクスポーターで使用される共通機能のサポートを提供します。

これらの機能は、それぞれのインスタンス属性を設定するコンストラクター引数で構成 (configure) できます： *fields_to_export* 、 *export_empty_fields* 、 *encoding* 、 *indent*

export_item(*item*)

与えられたアイテムをエクスポートします。このメソッドはサブクラスで実装する必要があります。

serialize_field(*field*, *name*, *value*)

指定のフィールドのシリアル化された値を返します。特定のフィールドまたは値をシリアル化/エクスポートする方法を制御する場合は、カスタム・アイテム・エクスポーターでこのメソッドをオーバーライドできます。

デフォルトでは、このメソッドは **項目フィールドで宣言** されたシリアライザーを探し、そして、そのシリアライザーを値に適用した結果を返します。シリアライザーが見つからない場合、 *encoding* 属性で宣言されたエンコーディングを使用して `str` にエンコードされる `unicode` 値を除いて、値を変更せずに返します。

パラメータ

- **field** (*Field* object or an empty dict) – シリアル化されているフィールド。生の辞書がエクスポートされる場合 (*Item* ではなく) **フィールド** 値は空の辞書です。
- **name** (*str*) – シリアル化されるフィールドの名前
- **value** – シリアル化される値

start_exporting()

エクスポート・プロセスの開始を通知します。一部のエクスポーターはこれを使用して、必要なヘッダー (*XmlItemExporter* など) を生成します。アイテムをエクスポートする前に、このメソッドを呼び出す必要があります。

finish_exporting()
エクスポート・プロセスの終了を通知します。一部のエクスポーターはこれを使用して、必要なフッター (たとえば、 *XmlItemExporter*) を生成します。エクスポートするアイテムがなくなったら、常にこのメソッドを呼び出す必要があります。

fields_to_export
エクスポートされるフィールドの名前のリスト、またはすべてのフィールドをエクスポートする場合は None。デフォルトは None です。

一部のエクスポーター (*CsvItemExporter* など) は、この属性で定義されたフィールドの順序を尊重します。

一部のエクスポーターは、スパイダーが辞書を返すときにデータを適切にエクスポートするために、fields_to_export リストを必要とする場合があります (`Item` インスタンスではありません)。

export_empty_fields
エクスポートされたデータに空/未入力の項目フィールドを含めるかどうか。デフォルトは `False` です。 一部のエクスポーター (*CsvItemExporter* など) はこの属性を無視し、常にすべての空のフィールドをエクスポートします。

このオプションは、辞書アイテムでは無視されます。

encoding
ユニコード値をエンコードするために使用されるエンコード。これは、ユニコード値 (このエンコードを使用して常に str にシリアル化される) にのみ影響します。他の値の型は変更されずに特定のシリアル化ライブラリに渡されます。

indent
各レベルで出力をインデントするために使用されるスペースの量。デフォルトは `0` です。

- `indent=None` は、全てのアイテムを同一行にインテント無しで出力する、最もコンパクトな表現を選択します
- `indent<=0` はアイテム毎に行を分けますが、インデントはありません
- `indent>0` はアイテム毎に行を分け、指定した数値でインデントします

PythonItemExporter

XmlItemExporter

class scrapy.exporters.**XmlItemExporter**(*file*, *item_element='item'*, *root_element='items'*, ***kwargs*)
指定のファイルオブジェクトにアイテムを XML 形式でエクスポートします。

パラメータ

- **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります
- **root_element** (*str*) – エクスポートされた XML のルート要素の名前。
- **item_element** (*str*) – エクスポートされた XML の各アイテム要素の名前。

このコンストラクタの追加のキーワード引数は、 *BaseItemExporter* コンストラクタに渡されます。

このエクスポーターの典型的な出力は次のようになります:

```
<?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>
```

`serialize_field()` メソッドでオーバーライドされない限り、複数の値を持つフィールドは `<value>` 要素内の各値をシリアル化することでエクスポートされます。複数値フィールドは非常に一般的であるため、これは便宜上のものです。

例えば、以下のアイテム:

```
Item(name=['John', 'Doe'], age='23')
```

これがシリアル化されると以下のようになります:

```
<?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.exporters.**CsvItemExporter**(*file*, *include_headers_line=True*, *join_multivalued=', '*, ***kwargs*)

与えられたファイルのようなオブジェクトに CSV 形式でアイテムをエクスポートします。 `fields_to_export` 属性が設定されている場合、CSV 列とその順序を定義するために使用されます。 `export_empty_fields` 属性はこのエクスポーターには影響しません。

パラメータ

- **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります
- **include_headers_line** (*str*) – 有効にすると、エクスポーターは *BaseItemExporter.fields_to_export* または最初にエクスポートされたアイテム・フィールドから取得したフィールド名を含むヘッダー行を出力します。
- **join_multivalued** – 複数値フィールドを結合するために使用される単一文字 (または複数の文字)。

このコンストラクタの追加のキーワード引数は *BaseItemExporter* コンストラクタに渡され、残りの引数は csv.writer コンストラクタに渡されるため、任意の `csv.writer` コンストラクタ引数を使用してエクスポーターをカスタマイズできます。

このエクスポーターの典型的な出力は次のようになります:

```
product,price
Color TV,1200
DVD player,200
```

PickleItemExporter

class scrapy.exporters.**PickleItemExporter**(*file*, *protocol=0*, ***kwargs*)

アイテムを pickle 形式で与えられたファイルのようなオブジェクトにエクスポートします。

パラメータ

- **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります
- **protocol** (*int*) – 使用する pickle プロトコル。

詳細については、 pickle module documentation を参照してください。

このコンストラクタの追加のキーワード引数は、 *BaseItemExporter* コンストラクタに渡されます。

pickle は人間が読める形式ではないため、出力例はありません。

PprintItemExporter

class scrapy.exporters.**PprintItemExporter**(*file*, ***kwargs*)

指定のファイルオブジェクトにきれいな (pretty) 印刷形式でアイテムをエクスポートします。

パラメータ **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります

このコンストラクタの追加のキーワード引数は、 *BaseItemExporter* コンストラクタに渡されます。

このエクスポーターの典型的な出力は次のようになります:

```
{'name': 'Color TV', 'price': '1200'}
{'name': 'DVD player', 'price': '200'}
```

(存在する場合) 長い行はきれいに (pretty) フォーマットされます。

JsonItemExporter

class scrapy.exporters.**JsonItemExporter**(*file*, ***kwargs*)

アイテムを JSON 形式で、指定されたファイルのようなオブジェクトにエクスポートし、すべてのオブジェクトをオブジェクトのリストとして書き込みます。追加のコンストラクター引数は *BaseItemExporter* コンストラクターに、残りの引数は JSONEncoder コンストラクターに渡されるため、任意の JSONEncoder コンストラクター引数を使用してこのエクスポーターをカスタマイズできます。

パラメータ **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります

このエクスポーターの典型的な出力は次のようになります:

```
[{"name": "Color TV", "price": "1200"},
{"name": "DVD player", "price": "200"}]
```

警告: JSON は非常にシンプルで柔軟なシリアル化形式ですが、(すべての言語で)JSON パーサ間でインクリメンタル (別名ストリームモード) 解析が (もしあれば) 十分にサポートされていないため、大量のデータに対して適切に拡張できません。それらのほとんどは、メモリ内のオブジェクト全体を解析するだけです。よりストリーム・フレンドリーな形式で JSON のパワーとシンプルさを望む場合は、代わりに *JsonLinesItemExporter* を使用するか、出力を複数のチャンクに分割することを検討してください。

JsonLinesItemExporter

class scrapy.exporters.**JsonLinesItemExporter**(*file*, ***kwargs*)

JSON 形式のアイテムを、指定されたファイルのようなオブジェクトにエクスポートし、1 行に JSON エンコードされたアイテムを 1 つ書き込みます。追加のコンストラクター引数は *BaseItemExporter* コンストラクターに、残りの引数は JSONEncoder コンストラクターに渡されるため、任意の JSONEncoder コンストラクター引数を使用してこのエクスポーターをカスタマイズできます。

パラメータ **file** – データのエクスポートに使用するファイルのようなオブジェクト。 その `write` メソッドは `bytes` (バイナリ・モードで開かれたディスクファイルや、 `io.BytesIO` オブジェクトなど) を受け入れる必要があります

このエクスポーターの典型的な出力は次のようになります:

```
{"name": "Color TV", "price": "1200"}
{"name": "DVD player", "price": "200"}
```

JsonItemExporter によって生成される形式とは異なり、このエクスポーターによって生成される形式は、大量のデータをシリアル化するのに適しています。

MarshalItemExporter

アーキテクチャ概観 Scrapy アーキテクチャを理解する。

ダウンローダー・ミドルウェア ページのリクエストとダウンロードの方法をカスタマイズします。

スパイダー・ミドルウェア あなたのスパイダーの入力と出力をカスタマイズします。

拡張機能 あなたのカスタム機能で Scrapy を拡張する。

コア *API* Scrapy 機能を拡張するために拡張機能やミドルウェアを使用します。

シグナル 利用可能なすべてのシグナルとそれらがどのように動くかをご覧下さい。

アイテム・エクスポーター スクレイプしたアイテムをファイル（XML、CSV など）にすばやくエクスポートします。

第 7 章

その他すべて

7.1 リリース・ノート

注釈: Scrapy 1.x は Python2 をサポートする最後のシリーズになります。Scrapy2.0 は 2019 年第 4 四半期または 2020 年第 1 四半期に予定されており、 **Python3 のみ** をサポートします。

7.1.1 Scrapy 1.7.3 (2019-08-01)

Python 3.4 では lxml 4.3.5 以下を強制する (issue 3912, issue 3918)

7.1.2 Scrapy 1.7.2 (2019-07-23)

Python 2 サポートを修正 (issue 3889, issue 3893, issue 3896).

7.1.3 Scrapy 1.7.1 (2019-07-18)

Scrapy 1.7.0 の再パッケージ化。PyPI の一部の変更が欠落していました。

7.1.4 Scrapy 1.7.0 (2019-07-18)

注釈: Scrapy 1.7.1 を必ずインストールしてください。 PyPI の Scrapy 1.7.0 パッケージは誤ったコミットタグ付けの結果であり、以下で説明するすべての変更が含まれていません。

ハイライト:

- 複数のドメインをターゲットとするクロールの改善
- 引数をコールバックに渡すよりクリーンな方法
- JSON リクエストの新しいクラス
- ルールベースのスパイダーの改善
- フィード・エクスポートの新機能

後方互換性のない変更

- `429` はデフォルトで *RETRY_HTTP_CODES* 設定の一部になりました

  この変更は **後方互換性がありません** 。 `429` を再試行したくない場合は、それに応じて *RETRY_HTTP_CODES* をオーバーライドする必要があります。

- *Crawler* と `CrawlerRunner.crawl` と `CrawlerRunner.create_crawler` は、もはや *Spider* のサブクラスのインスタンスを受け入れなくなり、*Spider* サブクラスのみを受け入れます。

  〜`scrapy.spiders.Spider` サブクラスのインスタンスは実行することを意図していなかったため、期待どおりに機能していませんでした。渡された 〜`scrapy.spiders.Spider` サブクラスのインスタンスを使用する代わりに、〜`scrapy.spiders.Spider.from_crawler` メソッドは、新しいインスタンスを生成するために呼び出されました。

- *SCHEDULER_PRIORITY_QUEUE* 設定のデフォルト以外の値が機能しなくなる場合があります。 スケジューラープライオリティキュークラスは、任意の Python データ構造ではなく 〜`scrapy.http.Request` オブジェクトを処理する必要があります。

下記の **非推奨による削除** も参照してください。

新機能

- 新しいスケジューラーの優先度キュー `scrapy.pqueues.DownloaderAwarePriorityQueue` は、*enabled* で、*CONCURRENT_REQUESTS_PER_IP* をサポートしないという犠牲を払えば、複数の Web ドメインをターゲットとするクロールのスケジューリングを大幅に改善できます。
- 新しい *Request.cb_kwargs* 属性は、キーワード引数をコールバック・メソッドに渡すよりクリーンな方法を提供します (issue 1138, issue 3563)
- 新しい *JSONRequest* クラスは、JSON リクエストを作成するより便利な方法を提供します (issue 3504, issue 3505)
- A `process_request` callback passed to the *Rule* constructor now receives the *Response* object that originated the request as its second argument (issue 3682)

- A new `restrict_text` parameter for the *`LinkExtractor`* constructor allows filtering links by linking text (issue 3622, issue 3635)
- A new *`FEED_STORAGE_S3_ACL`* setting allows defining a custom ACL for feeds exported to Amazon S3 (issue 3607)
- A new *`FEED_STORAGE_FTP_ACTIVE`* setting allows using FTP 's active connection mode for feeds exported to FTP servers (issue 3829)
- A new *`METAREFRESH_IGNORE_TAGS`* setting allows overriding which HTML tags are ignored when searching a response for HTML meta tags that trigger a redirect (issue 1422, issue 3768)
- A new *`redirect_reasons`* request meta key exposes the reason (status code, meta refresh) behind every followed redirect (issue 3581, issue 3687)
- The `SCRAPY_CHECK` variable is now set to the `true` string during runs of the *`check`* command, which allows *detecting contract check runs from code* (issue 3704, issue 3739)
- A new `Item.deepcopy()` method makes it easier to *deep-copy items* (issue 1493, issue 3671)
- *`CoreStats`* also logs `elapsed_time_seconds` now (issue 3638)
- Exceptions from *`ItemLoader`* *input and output processors* are now more verbose (issue 3836, issue 3840)
- *`Crawler`*, `CrawlerRunner.crawl` and `CrawlerRunner.create_crawler` now fail gracefully if they receive a *`Spider`* subclass instance instead of the subclass itself (issue 2283, issue 3610, issue 3872)

バグ修正

- *`process_spider_exception()`* is now also invoked for generators (issue 220, issue 2061)
- System exceptions like KeyboardInterrupt are no longer caught (issue 3726)
- *`ItemLoader.load_item()`* no longer makes later calls to *`ItemLoader.get_output_value()`* or *`ItemLoader.load_item()`* return empty data (issue 3804, issue 3819)
- The images pipeline (*`ImagesPipeline`*) no longer ignores these Amazon S3 settings: *`AWS_ENDPOINT_URL`*, *`AWS_REGION_NAME`*, *`AWS_USE_SSL`*, *`AWS_VERIFY`* (issue 3625)
- Fixed a memory leak in `MediaPipeline` affecting, for example, non-200 responses and exceptions from custom middlewares (issue 3813)
- Requests with private callbacks are now correctly unserialized from disk (issue 3790)
- *`FormRequest.from_response()`* now handles invalid methods like major web browsers (issue 3777, issue 3794)

Documentation

- A new topic, **動的に読み込まれたコンテンツの選択**, covers recommended approaches to read dynamically-loaded data (issue 3703)
- **広範なクロール** now features information about memory usage (issue 1264, issue 3866)
- The documentation of *Rule* now covers how to access the text of a link when using *CrawlSpider* (issue 3711, issue 3712)
- A new section, **あなた自身のストレージ・バックエンドを書く**, covers writing a custom cache storage backend for *HttpCacheMiddleware* (issue 3683, issue 3692)
- A new *FAQ* entry, **アイテム・パイプラインでアイテムを複数のアイテムに分割する方法は？**, explains what to do when you want to split an item into multiple items from an item pipeline (issue 2240, issue 3672)
- Updated the *FAQ entry about crawl order* to explain why the first few requests rarely follow the desired order (issue 1739, issue 3621)
- The *LOGSTATS_INTERVAL* setting (issue 3730), the *FilesPipeline.file_path* and *ImagesPipeline.file_path* methods (issue 2253, issue 3609) and the `Crawler.stop()` method (issue 3842) are now documented
- Some parts of the documentation that were confusing or misleading are now clearer (issue 1347, issue 1789, issue 2289, issue 3069, issue 3615, issue 3626, issue 3668, issue 3670, issue 3673, issue 3728, issue 3762, issue 3861, issue 3882)
- Minor documentation fixes (issue 3648, issue 3649, issue 3662, issue 3674, issue 3676, issue 3694, issue 3724, issue 3764, issue 3767, issue 3791, issue 3797, issue 3806, issue 3812)

非推奨による削除

次の非推奨 API は削除されました (issue 3578):

- `scrapy.conf` (use *Crawler.settings*)
- From `scrapy.core.downloader.handlers`:
  - `http.HttpDownloadHandler` (use `http10.HTTP10DownloadHandler`)
- `scrapy.loader.ItemLoader._get_values` (use `_get_xpathvalues`)
- `scrapy.loader.XPathItemLoader` (use *ItemLoader*)
- `scrapy.log` (see **ロギング** *(logging)*)
- From `scrapy.pipelines`:
  - `files.FilesPipeline.file_key` (use `file_path`)

  - images.ImagesPipeline.file_key (use file_path)
  - images.ImagesPipeline.image_key (use file_path)
  - images.ImagesPipeline.thumb_key (use thumb_path)
- From both scrapy.selector and scrapy.selector.lxmlsel:
  - HtmlXPathSelector (use Selector)
  - XmlXPathSelector (use Selector)
  - XPathSelector (use Selector)
  - XPathSelectorList (use Selector)
- From scrapy.selector.csstranslator:
  - ScrapyGenericTranslator (use parsel.csstranslator.GenericTranslator)
  - ScrapyHTMLTranslator (use parsel.csstranslator.HTMLTranslator)
  - ScrapyXPathExpr (use parsel.csstranslator.XPathExpr)
- From Selector:
  - _root (both the constructor argument and the object property, use root)
  - extract_unquoted (use getall)
  - select (use xpath)
- From SelectorList:
  - extract_unquoted (use getall)
  - select (use xpath)
  - x (use xpath)
- scrapy.spiders.BaseSpider (use *Spider*)
- From *Spider* (and subclasses):
  - DOWNLOAD_DELAY (use *download_delay*)
  - set_crawler (use *from_crawler()*)
- scrapy.spiders.spiders (use *SpiderLoader*)
- scrapy.telnet (use *scrapy.extensions.telnet*)
- From scrapy.utils.python:

  - `str_to_unicode` (use `to_unicode`)
  - `unicode_to_str` (use `to_bytes`)
- `scrapy.utils.response.body_or_str`

また、以下の非推奨の設定も削除されました (issue 3578):

- `SPIDER_MANAGER_CLASS` (use *`SPIDER_LOADER_CLASS`*)

非推奨

- The `queuelib.PriorityQueue` value for the *`SCHEDULER_PRIORITY_QUEUE`* setting is deprecated. Use `scrapy.pqueues.ScrapyPriorityQueue` instead.
- `process_request` callbacks passed to *`Rule`* that do not accept two arguments are deprecated.
- The following modules are deprecated:
  - `scrapy.utils.http` (use w3lib.http)
  - `scrapy.utils.markup` (use w3lib.html)
  - `scrapy.utils.multipart` (use urllib3)
- The `scrapy.utils.datatypes.MergeDict` class is deprecated for Python 3 code bases. Use `ChainMap` instead. (issue 3878)
- The `scrapy.utils.gz.is_gzipped` function is deprecated. Use `scrapy.utils.gz.gzip_magic_number` instead.

Other changes

- It is now possible to run all tests from the same tox environment in parallel; the documentation now covers *this and other ways to run tests* (issue 3707)
- It is now possible to generate an API documentation coverage report (issue 3806, issue 3810, issue 3860)
- The *documentation policies* now require docstrings (issue 3701) that follow PEP 257 (issue 3748)
- Internal fixes and cleanup (issue 3629, issue 3643, issue 3684, issue 3698, issue 3734, issue 3735, issue 3736, issue 3737, issue 3809, issue 3821, issue 3825, issue 3827, issue 3833, issue 3857, issue 3877)

7.1.5 Scrapy 1.6.0 (2019-01-30)

ハイライト:

- better Windows support;
- Python 3.7 compatibility;
- big documentation improvements, including a switch from `.extract_first()` + `.extract()` API to `.get()` + `.getall()` API;
- feed exports, FilePipeline and MediaPipeline improvements;
- better extensibility: *item_error* and *request_reached_downloader* signals; `from_crawler` support for feed exporters, feed storages and dupefilters.
- `scrapy.contracts` fixes and new features;
- telnet console security improvements, first released as a backport in *Scrapy 1.5.2 (2019-01-22)*;
- clean-up of the deprecated code;
- various bug fixes, small new features and usability improvements across the codebase.

セレクター API 変更

While these are not changes in Scrapy itself, but rather in the parsel library which Scrapy uses for xpath/css selectors, these changes are worth mentioning here. Scrapy now depends on parsel >= 1.5, and Scrapy documentation is updated to follow recent `parsel` API conventions.

Most visible change is that `.get()` and `.getall()` selector methods are now preferred over `.extract_first()` and `.extract()`. We feel that these new methods result in a more concise and readable code. See *extract() と extract_first()* for more details.

注釈: There are currently **no plans** to deprecate `.extract()` and `.extract_first()` methods.

Another useful new feature is the introduction of `Selector.attrib` and `SelectorList.attrib` properties, which make it easier to get attributes of HTML elements. See **要素属性の選択**.

CSS selectors are cached in parsel >= 1.5, which makes them faster when the same CSS path is used many times. This is very common in case of Scrapy spiders: callbacks are usually called several times, on different pages.

カスタム `Selector` または `SelectorList` サブクラスを使用している場合、parsel の **後方互換性のない** 変更がコードに影響する可能性があります。 詳細な説明と改善点の完全なリストについては、 parsel changelog を参照してください。

Telnet コンソール

下位互換性なし：Scrapy の telnet コンソールには、ユーザー名とパスワードが必要になりました。詳細については、 *Telnet* **コンソール** を参照してください。この変更により、 **セキュリティの問題** が修正されます。詳細については、 *Scrapy 1.5.2 (2019-01-22)* リリースノートを参照してください。

新しい拡張機能

- `from_crawler` support is added to feed exporters and feed storages. This, among other things, allows to access Scrapy settings from custom feed storages and exporters (issue 1605, issue 3348).
- `from_crawler` support is added to dupefilters (issue 2956); this allows to access e.g. settings or a spider from a dupefilter.
- *`item_error`* is fired when an error happens in a pipeline (issue 3256);
- *`request_reached_downloader`* is fired when Downloader gets a new Request; this signal can be useful e.g. for custom Schedulers (issue 3393).
- new SitemapSpider *`sitemap_filter()`* method which allows to select sitemap entries based on their attributes in SitemapSpider subclasses (issue 3512).
- Lazy loading of Downloader Handlers is now optional; this enables better initialization error handling in custom Downloader Handlers (issue 3394).

New FilePipeline and MediaPipeline features

- Expose more options for S3FilesStore: *`AWS_ENDPOINT_URL`*, *`AWS_USE_SSL`*, *`AWS_VERIFY`*, *`AWS_REGION_NAME`*. For example, this allows to use alternative or self-hosted AWS-compatible providers (issue 2609, issue 3548).
- ACL support for Google Cloud Storage: *`FILES_STORE_GCS_ACL`* and *`IMAGES_STORE_GCS_ACL`* (issue 3199).

`scrapy.contracts` improvements

- Exceptions in contracts code are handled better (issue 3377);
- `dont_filter=True` is used for contract requests, which allows to test different callbacks with the same URL (issue 3381);
- `request_cls` attribute in Contract subclasses allow to use different Request classes in contracts, for example FormRequest (issue 3383).
- Fixed errback handling in contracts, e.g. for cases where a contract is executed for URL which returns non-200 response (issue 3371).

Usability improvements

- more stats for RobotsTxtMiddleware (issue 3100)
- INFO log level is used to show telnet host/port (issue 3115)
- a message is added to IgnoreRequest in RobotsTxtMiddleware (issue 3113)
- better validation of `url` argument in `Response.follow` (issue 3131)
- non-zero exit code is returned from Scrapy commands when error happens on spider inititalization (issue 3226)
- Link extraction improvements: "ftp" is added to scheme list (issue 3152); "flv" is added to common video extensions (issue 3165)
- better error message when an exporter is disabled (issue 3358);
- `scrapy shell --help` mentions syntax required for local files (`./file.html`) - issue 3496.
- Referer header value is added to RFPDupeFilter log messages (issue 3588)

バグ修正

- fixed issue with extra blank lines in .csv exports under Windows (issue 3039);
- proper handling of pickling errors in Python 3 when serializing objects for disk queues (issue 3082)
- flags are now preserved when copying Requests (issue 3342);
- FormRequest.from_response clickdata shouldn't ignore elements with `input[type=image]` (issue 3153).
- FormRequest.from_response should preserve duplicate keys (issue 3247)

Documentation improvements

- Docs are re-written to suggest .get/.getall API instead of .extract/.extract_first. Also, **セレクター** docs are updated and re-structured to match latest parsel docs; they now contain more topics, such as **要素属性の選択** or *CSS* **セレクターの拡張機能** (issue 3390).
- *Web* **ブラウザの開発ツールを使ってスクレイピングする** is a new tutorial which replaces old Firefox and Firebug tutorials (issue 3400).
- SCRAPY_PROJECT environment variable is documented (issue 3518);
- troubleshooting section is added to install instructions (issue 3517);
- improved links to beginner resources in the tutorial (issue 3367, issue 3468);
- fixed *RETRY_HTTP_CODES* default values in docs (issue 3335);

- remove unused `DEPTH_STATS` option from docs (issue 3245);
- other cleanups (issue 3347, issue 3350, issue 3445, issue 3544, issue 3605).

非推奨による削除

Compatibility shims for pre-1.0 Scrapy module names are removed (issue 3318):

- `scrapy.command`
- `scrapy.contrib` (with all submodules)
- `scrapy.contrib_exp` (with all submodules)
- `scrapy.dupefilter`
- `scrapy.linkextractor`
- `scrapy.project`
- `scrapy.spider`
- `scrapy.spidermanager`
- `scrapy.squeue`
- `scrapy.stats`
- `scrapy.statscol`
- `scrapy.utils.decorator`

See **モジュールの再配置** for more information, or use suggestions from Scrapy 1.5.x deprecation warnings to update your code.

他の非推奨削除:

- 非推奨の scrapy.interfaces.ISpiderManager は削除されました。scrapy.interfaces.ISpiderLoader を使って下さい。
- 非推奨の `CrawlerSettings` クラスは削除されました (issue 3327).
- 非推奨の `Settings.overrides` と `Settings.defaults` 属性は削除されました (issue 3327, issue 3359).

その他の改善、クリーンアップ

- All Scrapy tests now pass on Windows; Scrapy testing suite is executed in a Windows environment on CI (issue 3315).

- Python 3.7 サポート (issue 3326, issue 3150, issue 3547).
- Testing and CI fixes (issue 3526, issue 3538, issue 3308, issue 3311, issue 3309, issue 3305, issue 3210, issue 3299)
- `scrapy.http.cookies.CookieJar.clear` accepts "domain", "path" and "name" optional arguments (issue 3231).
- additional files are included to sdist (issue 3495);
- code style fixes (issue 3405, issue 3304);
- unneeded .strip() call is removed (issue 3519);
- collections.deque is used to store MiddlewareManager methods instead of a list (issue 3476)

7.1.6 Scrapy 1.5.2 (2019-01-22)

- **セキュリティバグ修正**: Telnet console extension can be easily exploited by rogue websites POSTing content to http://localhost:6023, we haven't found a way to exploit it from Scrapy, but it is very easy to trick a browser to do so and elevates the risk for local development environment.

  この修正は下位互換性がありません, it enables telnet user-password authentication by default with a random generated password. If you can't upgrade right away, please consider setting `TELNET_CONSOLE_PORT` out of its default value.

  詳細は *telnet* **コンソール** 文書参照

- Backport CI build failure under GCE environemnt due to boto import error.

7.1.7 Scrapy 1.5.1 (2018-07-12)

重要なバグ修正のためのメンテナンス・リリースです。新機能の追加はありません。

- `O(N^2)` gzip decompression issue which affected Python 3 and PyPy is fixed (issue 3281);
- skipping of TLS validation errors is improved (issue 3166);
- Ctrl-C handling is fixed in Python 3.5+ (issue 3096);
- testing fixes (issue 3092, issue 3263);
- documentation improvements (issue 3058, issue 3059, issue 3089, issue 3123, issue 3127, issue 3189, issue 3224, issue 3280, issue 3279, issue 3201, issue 3260, issue 3284, issue 3298, issue 3294).

7.1.8 Scrapy 1.5.0 (2017-12-29)

This release brings small new features and improvements across the codebase. Some highlights:

- Google Cloud Storage is supported in FilesPipeline and ImagesPipeline.
- Crawling with proxy servers becomes more efficient, as connections to proxies can be reused now.
- Warnings, exception and logging messages are improved to make debugging easier.
- `scrapy parse` command now allows to set custom request meta via `--meta` argument.
- Compatibility with Python 3.6, PyPy and PyPy3 is improved; PyPy and PyPy3 are now supported officially, by running tests on CI.
- Better default handling of HTTP 308, 522 and 524 status codes.
- Documentation is improved, as usual.

Backward Incompatible Changes

- Scrapy 1.5 drops support for Python 3.3.
- Default Scrapy User-Agent now uses https link to scrapy.org (issue 2983). **This is technically backward-incompatible**; override *`USER_AGENT`* if you relied on old value.
- Logging of settings overridden by `custom_settings` is fixed; **this is technically backward-incompatible** because the logger changes from `[scrapy.utils.log]` to `[scrapy.crawler]`. If you're parsing Scrapy logs, please update your log parsers (issue 1343).
- LinkExtractor now ignores `m4v` extension by default, this is change in behavior.
- 522 and 524 status codes are added to `RETRY_HTTP_CODES` (issue 2851)

新機能

- Support `<link>` tags in `Response.follow` (issue 2785)
- Support for `ptpython` REPL (issue 2654)
- Google Cloud Storage support for FilesPipeline and ImagesPipeline (issue 2923).
- New `--meta` option of the "scrapy parse" command allows to pass additional request.meta (issue 2883)
- Populate spider variable when using `shell.inspect_response` (issue 2812)
- Handle HTTP 308 Permanent Redirect (issue 2844)
- Add 522 and 524 to `RETRY_HTTP_CODES` (issue 2851)

- Log versions information at startup (issue 2857)
- `scrapy.mail.MailSender` now works in Python 3 (it requires Twisted 17.9.0)
- Connections to proxy servers are reused (issue 2743)
- Add template for a downloader middleware (issue 2755)
- Explicit message for NotImplementedError when parse callback not defined (issue 2831)
- CrawlerProcess got an option to disable installation of root log handler (issue 2921)
- LinkExtractor now ignores `m4v` extension by default
- Better log messages for responses over *DOWNLOAD_WARNSIZE* and *DOWNLOAD_MAXSIZE* limits (issue 2927)
- Show warning when a URL is put to `Spider.allowed_domains` instead of a domain (issue 2250).

バグ修正

- Fix logging of settings overridden by `custom_settings`; **this is technically backward-incompatible** because the logger changes from `[scrapy.utils.log]` to `[scrapy.crawler]`, so please update your log parsers if needed (issue 1343)
- Default Scrapy User-Agent now uses https link to scrapy.org (issue 2983). **This is technically backward-incompatible**; override *USER_AGENT* if you relied on old value.
- Fix PyPy and PyPy3 test failures, support them officially (issue 2793, issue 2935, issue 2990, issue 3050, issue 2213, issue 3048)
- Fix DNS resolver when `DNSCACHE_ENABLED=False` (issue 2811)
- Add `cryptography` for Debian Jessie tox test env (issue 2848)
- Add verification to check if Request callback is callable (issue 2766)
- Port `extras/qpsclient.py` to Python 3 (issue 2849)
- Use getfullargspec under the scenes for Python 3 to stop DeprecationWarning (issue 2862)
- Update deprecated test aliases (issue 2876)
- Fix `SitemapSpider` support for alternate links (issue 2853)

Docs

- Added missing bullet point for the `AUTOTHROTTLE_TARGET_CONCURRENCY` setting. (issue 2756)
- Update Contributing docs, document new support channels (issue 2762, issue:*3038*)

- Include references to Scrapy subreddit in the docs
- Fix broken links; use https:// for external links (issue 2978, issue 2982, issue 2958)
- Document CloseSpider extension better (issue 2759)
- Use `pymongo.collection.Collection.insert_one()` in MongoDB example (issue 2781)
- Spelling mistake and typos (issue 2828, issue 2837, issue 2884, issue 2924)
- Clarify `CSVFeedSpider.headers` documentation (issue 2826)
- Document `DontCloseSpider` exception and clarify `spider_idle` (issue 2791)
- Update "Releases" section in README (issue 2764)
- Fix rst syntax in `DOWNLOAD_FAIL_ON_DATALOSS` docs (issue 2763)
- Small fix in description of startproject arguments (issue 2866)
- Clarify data types in Response.body docs (issue 2922)
- Add a note about `request.meta['depth']` to DepthMiddleware docs (issue 2374)
- Add a note about `request.meta['dont_merge_cookies']` to CookiesMiddleware docs (issue 2999)
- Up-to-date example of project structure (issue 2964, issue 2976)
- A better example of ItemExporters usage (issue 2989)
- Document `from_crawler` methods for spider and downloader middlewares (issue 3019)

7.1.9 Scrapy 1.4.0 (2017-05-18)

Scrapy 1.4 does not bring that many breathtaking new features but quite a few handy improvements nonetheless.

Scrapy now supports anonymous FTP sessions with customizable user and password via the new *FTP_USER* and *FTP_PASSWORD* settings. And if you're using Twisted version 17.1.0 or above, FTP is now available with Python 3.

There's a new `response.follow` method for creating requests; **it is now a recommended way to create Requests in Scrapy spiders**. This method makes it easier to write correct spiders; `response.follow` has several advantages over creating `scrapy.Request` objects directly:

- it handles relative URLs;
- it works properly with non-ascii URLs on non-UTF8 pages;
- in addition to absolute and relative URLs it supports Selectors; for `<a>` elements it can also extract their href values.

For example, instead of this:

```
for href in response.css('li.page a::attr(href)').extract():
    url = response.urljoin(href)
    yield scrapy.Request(url, self.parse, encoding=response.encoding)
```

One can now write this:

```
for a in response.css('li.page a'):
    yield response.follow(a, self.parse)
```

Link extractors are also improved. They work similarly to what a regular modern browser would do: leading and trailing whitespace are removed from attributes (think `href=" http://example.com"`) when building `Link` objects. This whitespace-stripping also happens for `action` attributes with `FormRequest`.

Please also note that link extractors do not canonicalize URLs by default anymore. This was puzzling users every now and then, and it's not what browsers do in fact, so we removed that extra transformation on extracted links.

For those of you wanting more control on the `Referer:` header that Scrapy sends when following links, you can set your own `Referrer Policy`. Prior to Scrapy 1.4, the default `RefererMiddleware` would simply and blindly set it to the URL of the response that generated the HTTP request (which could leak information on your URL seeds). By default, Scrapy now behaves much like your regular browser does. And this policy is fully customizable with W3C standard values (or with something really custom of your own if you wish). See *REFERRER_POLICY* for details.

To make Scrapy spiders easier to debug, Scrapy logs more stats by default in 1.4: memory usage stats, detailed retry stats, detailed HTTP error code stats. A similar change is that HTTP cache path is also visible in logs now.

Last but not least, Scrapy now has the option to make JSON and XML items more human-readable, with newlines between items and even custom indenting offset, using the new *FEED_EXPORT_INDENT* setting.

Enjoy! (Or read on for the rest of changes in this release.)

非推奨と下位互換性のない変更

- Default to `canonicalize=False` in `scrapy.linkextractors.LinkExtractor` (issue 2537, fixes issue 1941 and issue 1982): **warning, this is technically backward-incompatible**
- Enable memusage extension by default (issue 2539, fixes issue 2187); **this is technically backward-incompatible** so please check if you have any non-default `MEMUSAGE_***` options set.
- `EDITOR` environment variable now takes precedence over `EDITOR` option defined in settings.py (issue 1829); Scrapy default settings no longer depend on environment variables. **This is technically a backward incompatible change**.
- `Spider.make_requests_from_url` is deprecated (issue 1728, fixes issue 1495).

New Features

- Accept proxy credentials in *proxy* request meta key (issue 2526)
- Support brotli-compressed content; requires optional brotlipy (issue 2535)
- New *response.follow* shortcut for creating requests (issue 1940)
- Added `flags` argument and attribute to *Request* objects (issue 2047)
- Support Anonymous FTP (issue 2342)
- Added `retry/count`, `retry/max_reached` and `retry/reason_count/<reason>` stats to *RetryMiddleware* (issue 2543)
- Added `httperror/response_ignored_count` and `httperror/response_ignored_status_count/<status>` stats to *HttpErrorMiddleware* (issue 2566)
- Customizable *Referrer policy* in *RefererMiddleware* (issue 2306)
- New `data:` URI download handler (issue 2334, fixes issue 2156)
- Log cache directory when HTTP Cache is used (issue 2611, fixes issue 2604)
- Warn users when project contains duplicate spider names (fixes issue 2181)
- `CaselessDict` now accepts `Mapping` instances and not only dicts (issue 2646)
- *Media downloads*, with `FilesPipelines` or `ImagesPipelines`, can now optionally handle HTTP redirects using the new *MEDIA_ALLOW_REDIRECTS* setting (issue 2616, fixes issue 2004)
- Accept non-complete responses from websites using a new *DOWNLOAD_FAIL_ON_DATALOSS* setting (issue 2590, fixes issue 2586)
- Optional pretty-printing of JSON and XML items via *FEED_EXPORT_INDENT* setting (issue 2456, fixes issue 1327)
- Allow dropping fields in `FormRequest.from_response` formdata when `None` value is passed (issue 667)
- Per-request retry times with the new *max_retry_times* meta key (issue 2642)
- `python -m scrapy` as a more explicit alternative to `scrapy` command (issue 2740)

バグ修正

- LinkExtractor now strips leading and trailing whitespaces from attributes (issue 2547, fixes issue 1614)
- Properly handle whitespaces in action attribute in `FormRequest` (issue 2548)
- Buffer CONNECT response bytes from proxy until all HTTP headers are received (issue 2495, fixes issue 2491)

- FTP downloader now works on Python 3, provided you use Twisted>=17.1 (issue 2599)
- Use body to choose response type after decompressing content (issue 2393, fixes issue 2145)
- Always decompress `Content-Encoding: gzip` at *`HttpCompressionMiddleware`* stage (issue 2391)
- Respect custom log level in `Spider.custom_settings` (issue 2581, fixes issue 1612)
- 'make htmlview' fix for macOS (issue 2661)
- Remove "commands" from the command list (issue 2695)
- Fix duplicate Content-Length header for POST requests with empty body (issue 2677)
- Properly cancel large downloads, i.e. above *`DOWNLOAD_MAXSIZE`* (issue 1616)
- ImagesPipeline: fixed processing of transparent PNG images with palette (issue 2675)

Cleanups & Refactoring

- Tests: remove temp files and folders (issue 2570), fixed ProjectUtilsTest on OS X (issue 2569), use portable pypy for Linux on Travis CI (issue 2710)
- Separate building request from `_requests_to_follow` in CrawlSpider (issue 2562)
- Remove " Python 3 progress " badge (issue 2567)
- Add a couple more lines to `.gitignore` (issue 2557)
- Remove bumpversion prerelease configuration (issue 2159)
- Add codecov.yml file (issue 2750)
- Set context factory implementation based on Twisted version (issue 2577, fixes issue 2560)
- Add omitted `self` arguments in default project middleware template (issue 2595)
- Remove redundant `slot.add_request()` call in ExecutionEngine (issue 2617)
- Catch more specific `os.error` exception in `FSFilesStore` (issue 2644)
- Change "localhost" test server certificate (issue 2720)
- Remove unused `MEMUSAGE_REPORT` setting (issue 2576)

Documentation

- Binary mode is required for exporters (issue 2564, fixes issue 2553)
- Mention issue with *`FormRequest.from_response`* due to bug in lxml (issue 2572)

- Use single quotes uniformly in templates (issue 2596)
- Document `ftp_user` and `ftp_password` meta keys (issue 2587)
- Removed section on deprecated `contrib/` (issue 2636)
- Recommend Anaconda when installing Scrapy on Windows (issue 2477, fixes issue 2475)
- FAQ: rewrite note on Python 3 support on Windows (issue 2690)
- Rearrange selector sections (issue 2705)
- Remove `__nonzero__` from `SelectorList` docs (issue 2683)
- Mention how to disable request filtering in documentation of *`DUPEFILTER_CLASS`* setting (issue 2714)
- Add sphinx_rtd_theme to docs setup readme (issue 2668)
- Open file in text mode in JSON item writer example (issue 2729)
- Clarify `allowed_domains` example (issue 2670)

7.1.10 Scrapy 1.3.3 (2017-03-10)

バグ修正

- Make `SpiderLoader` raise `ImportError` again by default for missing dependencies and wrong *`SPIDER_MODULES`*. These exceptions were silenced as warnings since 1.3.0. A new setting is introduced to toggle between warning or exception if needed ; see *`SPIDER_LOADER_WARN_ONLY`* for details.

7.1.11 Scrapy 1.3.2 (2017-02-13)

バグ修正

- Preserve request class when converting to/from dicts (utils.reqser) (issue 2510).
- Use consistent selectors for author field in tutorial (issue 2551).
- Fix TLS compatibility in Twisted 17+ (issue 2558)

7.1.12 Scrapy 1.3.1 (2017-02-08)

新機能

- Support `'True'` and `'False'` string values for boolean settings (issue 2519); you can now do something like `scrapy crawl myspider -s REDIRECT_ENABLED=False`.

- Support kwargs with `response.xpath()` to use *XPath variables* and ad-hoc namespaces declarations ; this requires at least Parsel v1.1 (issue 2457).
- Add support for Python 3.6 (issue 2485).
- Run tests on PyPy (warning: some tests still fail, so PyPy is not supported yet).

バグ修正

- Enforce `DNS_TIMEOUT` setting (issue 2496).
- Fix `view` command ; it was a regression in v1.3.0 (issue 2503).
- Fix tests regarding `*_EXPIRES settings` with Files/Images pipelines (issue 2460).
- Fix name of generated pipeline class when using basic project template (issue 2466).
- Fix compatiblity with Twisted 17+ (issue 2496, issue 2528).
- Fix `scrapy.Item` inheritance on Python 3.6 (issue 2511).
- Enforce numeric values for components order in `SPIDER_MIDDLEWARES`, `DOWNLOADER_MIDDLEWARES`, `EXTENIONS` and `SPIDER_CONTRACTS` (issue 2420).

Documentation

- Reword Code of Coduct section and upgrade to Contributor Covenant v1.4 (issue 2469).
- Clarify that passing spider arguments converts them to spider attributes (issue 2483).
- Document `formid` argument on `FormRequest.from_response()` (issue 2497).
- Add .rst extension to README files (issue 2507).
- Mention LevelDB cache storage backend (issue 2525).
- Use `yield` in sample callback code (issue 2533).
- Add note about HTML entities decoding with `.re()/.re_first()` (issue 1704).
- Typos (issue 2512, issue 2534, issue 2531).

Cleanups

- Remove reduntant check in `MetaRefreshMiddleware` (issue 2542).
- Faster checks in `LinkExtractor` for allow/deny patterns (issue 2538).
- Remove dead code supporting old Twisted versions (issue 2544).

7.1.13 Scrapy 1.3.0 (2016-12-21)

このリリースは、1.2.2 のある主要な理由により、1.2.2 の直後の版となります。0.18 から 1.2.2（含まれる）以降のリリースでは、Twisted(`scrapy.xlib.tx.*`) からのバックポートされたコードを使用することがわかりました。 より新しい Twisted モジュールが利用できる場合でも。 Scrapy は `twisted.web.client` と `twisted.internet.endpoints` を直接使用するようになりました。（以下のクリーンアップも参照してください。）

これは大きな変更であるため、1.2 シリーズを使用しているプロジェクトを壊すことなく、バグをすばやく修正したかったのです。

New Features

- `MailSender` は `to` 引数と `cc` 引数の値として単一の文字列を受け入れるようになりました (issue 2272)
- Scrapy シェル内の `scrapy fetch url` と `scrapy shell url` と `fetch(url)` はデフォルトで HTTP リダイレクトに従います。(issue 2290) 詳細については、*`fetch`* と *`shell`* を参照してください。
- `HttpErrorMiddleware` は `DEBUG` ではなく `INFO` レベルでエラーを記録するようになりました。 これは技術的には **後方互換性がない** ので、ログパーサーを確認してください。
- デフォルトでは、ロガー名は以前のリリースの短い「トップレベル」バリアント (例 `[scrapy]`) ではなく、長い形式のパス `[scrapy.extensions.logstats]` を使用するようになりました。短いロガー名の部分を期待するログパーサーがある場合、 **後方互換性がありません** 。 *`LOG_SHORT_NAMES`* を `True` に設定して、短いロガー名に戻すことができます。

Dependencies & Cleanups

- Scrapy now requires Twisted >= 13.1 which is the case for many Linux distributions already.
- As a consequence, we got rid of `scrapy.xlib.tx.*` modules, which copied some of Twisted code for users stuck with an "old" Twisted version
- `ChunkedTransferMiddleware` is deprecated and removed from the default downloader middlewares.

7.1.14 Scrapy 1.2.3 (2017-03-03)

- パッケージの修正：setup.py でサポートされていない Twisted バージョンを禁止します。

7.1.15 Scrapy 1.2.2 (2016-12-06)

バグ修正

- Fix a cryptic traceback when a pipeline fails on `open_spider()` (issue 2011)
- Fix embedded IPython shell variables (fixing issue 396 that re-appeared in 1.2.0, fixed in issue 2418)
- A couple of patches when dealing with robots.txt:
  - handle (non-standard) relative sitemap URLs (issue 2390)
  - handle non-ASCII URLs and User-Agents in Python 2 (issue 2373)

Documentation

- Document `"download_latency"` key in `Request`'s `meta` dict (issue 2033)
- Remove page on (deprecated & unsupported) Ubuntu packages from ToC (issue 2335)
- A few fixed typos (issue 2346, issue 2369, issue 2369, issue 2380) and clarifications (issue 2354, issue 2325, issue 2414)

Other changes

- Advertize conda-forge as Scrapy's official conda channel (issue 2387)
- More helpful error messages when trying to use `.css()` or `.xpath()` on non-Text Responses (issue 2264)
- `startproject` command now generates a sample `middlewares.py` file (issue 2335)
- Add more dependencies' version info in `scrapy version` verbose output (issue 2404)
- Remove all `*.pyc` files from source distribution (issue 2386)

7.1.16 Scrapy 1.2.1 (2016-10-21)

バグ修正

- Include OpenSSL's more permissive default ciphers when establishing TLS/SSL connections (issue 2314).
- Fix "Location" HTTP header decoding on non-ASCII URL redirects (issue 2321).

Documentation

- Fix JsonWriterPipeline example (issue 2302).
- Various notes: issue 2330 on spider names, issue 2329 on middleware methods processing order, issue 2327 on getting multi-valued HTTP headers as lists.

Other changes

- Removed `www.` from `start_urls` in built-in spider templates (issue 2299).

7.1.17 Scrapy 1.2.0 (2016-10-03)

New Features

- 新しい *FEED_EXPORT_ENCODING* 設定は、アイテムをファイルに書き込むときに使用されるエンコーディングをカスタマイズします。 これは、JSON 出力で `\uXXXX` エスケープをオフにするために使用できます。 これは、XML または CSV 出力に UTF-8 以外のものが必要な場合にも便利です。(issue 2034)
- `startproject` コマンドは、プロジェクト名に基づいてデフォルトのディレクトリを上書きするオプションの宛先ディレクトリをサポートするようになりました。(issue 2005)
- 新しい *SCHEDULER_DEBUG* 設定は、リクエストのシリアル化エラーをログに記録します (issue 1610)
- JSON エンコーダーが `set` インスタンスのシリアル化をサポートするようになりました (issue 2058)
- `application/json-amazonui-streaming` を `TextResponse` として解釈する (issue 1503).
- `scrapy` is imported by default when using shell tools (*shell*, *inspect_response*) (issue 2248).

バグ修正

- DefaultRequestHeaders middleware now runs before UserAgent middleware (issue 2088). **Warning: this is technically backward incompatible**, though we consider this a bug fix.
- HTTP cache extension and plugins that use the `.scrapy` data directory now work outside projects (issue 1581). **Warning: this is technically backward incompatible**, though we consider this a bug fix.
- `Selector` does not allow passing both `response` and `text` anymore (issue 2153).
- Fixed logging of wrong callback name with `scrapy parse` (issue 2169).
- Fix for an odd gzip decompression bug (issue 1606).
- Fix for selected callbacks when using `CrawlSpider` with *scrapy parse* (issue 2225).
- Fix for invalid JSON and XML files when spider yields no items (issue 872).
- Implement `flush()` fpr `StreamLogger` avoiding a warning in logs (issue 2125).

リファクタリング

- `canonicalize_url` has been moved to w3lib.url (issue 2168).

Tests & Requirements

Scrapy's new requirements baseline is Debian 8 "Jessie". It was previously Ubuntu 12.04 Precise. What this means in practice is that we run continuous integration tests with these (main) packages versions at a minimum: Twisted 14.0, pyOpenSSL 0.14, lxml 3.4.

Scrapy may very well work with older versions of these packages (the code base still has switches for older Twisted versions for example) but it is not guaranteed (because it's not tested anymore).

Documentation

- Grammar fixes: issue 2128, issue 1566.
- Download stats badge removed from README (issue 2160).
- New scrapy *architecture diagram* (issue 2165).
- Updated `Response` parameters documentation (issue 2197).
- Reworded misleading `RANDOMIZE_DOWNLOAD_DELAY` description (issue 2190).
- Add StackOverflow as a support channel (issue 2257).

7.1.18 Scrapy 1.1.4 (2017-03-03)

- パッケージの修正：setup.py でサポートされていない Twisted バージョンを禁止します。

7.1.19 Scrapy 1.1.3 (2016-09-22)

バグ修正

- Class attributes for subclasses of `ImagesPipeline` and `FilesPipeline` work as they did before 1.1.1 (issue 2243, fixes issue 2198)

Documentation

- *Overview* and *tutorial* rewritten to use http://toscrape.com websites (issue 2236, issue 2249, issue 2252).

7.1.20 Scrapy 1.1.2 (2016-08-18)

バグ修正

- Introduce a missing *IMAGES_STORE_S3_ACL* setting to override the default ACL policy in `ImagesPipeline` when uploading images to S3 (note that default ACL policy is "private" – instead of "public-read" – since Scrapy 1.1.0)
- *IMAGES_EXPIRES* default value set back to 90 (the regression was introduced in 1.1.1)

7.1.21 Scrapy 1.1.1 (2016-07-13)

バグ修正

- Add "Host" header in CONNECT requests to HTTPS proxies (issue 2069)
- Use response `body` when choosing response class (issue 2001, fixes issue 2000)
- Do not fail on canonicalizing URLs with wrong netlocs (issue 2038, fixes issue 2010)
- a few fixes for `HttpCompressionMiddleware` (and `SitemapSpider`):
  - Do not decode HEAD responses (issue 2008, fixes issue 1899)
  - Handle charset parameter in gzip Content-Type header (issue 2050, fixes issue 2049)
  - Do not decompress gzip octet-stream responses (issue 2065, fixes issue 2063)
- Catch (and ignore with a warning) exception when verifying certificate against IP-address hosts (issue 2094, fixes issue 2092)
- Make `FilesPipeline` and `ImagesPipeline` backward compatible again regarding the use of legacy class attributes for customization (issue 1989, fixes issue 1985)

新機能

- Enable genspider command outside project folder (issue 2052)
- Retry HTTPS CONNECT `TunnelError` by default (issue 1974)

Documentation

- `FEED_TEMPDIR` setting at lexicographical position (commit 9b3c72c)
- Use idiomatic `.extract_first()` in overview (issue 1994)
- Update years in copyright notice (commit c2c8036)
- Add information and example on errbacks (issue 1995)

- Use "url" variable in downloader middleware example (issue 2015)
- Grammar fixes (issue 2054, issue 2120)
- New FAQ entry on using BeautifulSoup in spider callbacks (issue 2048)
- Add notes about scrapy not working on Windows with Python 3 (issue 2060)
- Encourage complete titles in pull requests (issue 2026)

Tests

- Upgrade py.test requirement on Travis CI and Pin pytest-cov to 2.2.1 (issue 2095)

7.1.22 Scrapy 1.1.0 (2016-05-11)

This 1.1 release brings a lot of interesting features and bug fixes:

- Scrapy 1.1 has beta Python 3 support (requires Twisted >= 15.5). See *Beta Python 3 Support* for more details and some limitations.
- Hot new features:
  - Item loaders now support nested loaders (issue 1467).
  - `FormRequest.from_response` improvements (issue 1382, issue 1137).
  - Added setting *AUTOTHROTTLE_TARGET_CONCURRENCY* and improved AutoThrottle docs (issue 1324).
  - Added `response.text` to get body as unicode (issue 1730).
  - Anonymous S3 connections (issue 1358).
  - Deferreds in downloader middlewares (issue 1473). This enables better robots.txt handling (issue 1471).
  - HTTP caching now follows RFC2616 more closely, added settings *HTTPCACHE_ALWAYS_STORE* and *HTTPCACHE_IGNORE_RESPONSE_CACHE_CONTROLS* (issue 1151).
  - Selectors were extracted to the parsel library (issue 1409). This means you can use Scrapy Selectors without Scrapy and also upgrade the selectors engine without needing to upgrade Scrapy.
  - HTTPS downloader now does TLS protocol negotiation by default, instead of forcing TLS 1.0. You can also set the SSL/TLS method using the new *DOWNLOADER_CLIENT_TLS_METHOD*.
- These bug fixes may require your attention:

- Don't retry bad requests (HTTP 400) by default (issue 1289). If you need the old behavior, add `400` to *RETRY_HTTP_CODES*.
- Fix shell files argument handling (issue 1710, issue 1550). If you try `scrapy shell index.html` it will try to load the URL http://index.html, use `scrapy shell ./index.html` to load a local file.
- Robots.txt compliance is now enabled by default for newly-created projects (issue 1724). Scrapy will also wait for robots.txt to be downloaded before proceeding with the crawl (issue 1735). If you want to disable this behavior, update *ROBOTSTXT_OBEY* in `settings.py` file after creating a new project.
- Exporters now work on unicode, instead of bytes by default (issue 1080). If you use `PythonItemExporter`, you may want to update your code to disable binary mode which is now deprecated.
- Accept XML node names containing dots as valid (issue 1533).
- When uploading files or images to S3 (with `FilesPipeline` or `ImagesPipeline`), the default ACL policy is now "private" instead of "public" **Warning: backward incompatible!**. You can use *FILES_STORE_S3_ACL* to change it.
- We've reimplemented `canonicalize_url()` for more correct output, especially for URLs with non-ASCII characters (issue 1947). This could change link extractors output compared to previous scrapy versions. This may also invalidate some cache entries you could still have from pre-1.1 runs. **Warning: backward incompatible!**.

Keep reading for more details on other improvements and bug fixes.

Beta Python 3 Support

We have been hard at work to make Scrapy run on Python 3. As a result, now you can run spiders on Python 3.3, 3.4 and 3.5 (Twisted >= 15.5 required). Some features are still missing (and some may never be ported).

Almost all builtin extensions/middlewares are expected to work. However, we are aware of some limitations in Python 3:

- Scrapy does not work on Windows with Python 3
- Sending emails is not supported
- FTP download handler is not supported
- Telnet console is not supported

Additional New Features and Enhancements

- Scrapy now has a Code of Conduct (issue 1681).

- Command line tool now has completion for zsh (issue 934).
- Improvements to `scrapy shell`:
  - Support for bpython and configure preferred Python shell via `SCRAPY_PYTHON_SHELL` (issue 1100, issue 1444).
  - Support URLs without scheme (issue 1498) **Warning: backward incompatible!**
  - Bring back support for relative file path (issue 1710, issue 1550).
- Added *MEMUSAGE_CHECK_INTERVAL_SECONDS* setting to change default check interval (issue 1282).
- Download handlers are now lazy-loaded on first request using their scheme (issue 1390, issue 1421).
- HTTPS download handlers do not force TLS 1.0 anymore; instead, OpenSSL's `SSLv23_method()/TLS_method()` is used allowing to try negotiating with the remote hosts the highest TLS protocol version it can (issue 1794, issue 1629).
- `RedirectMiddleware` now skips the status codes from `handle_httpstatus_list` on spider attribute or in `Request`'s `meta` key (issue 1334, issue 1364, issue 1447).
- Form submission:
  - now works with `<button>` elements too (issue 1469).
  - an empty string is now used for submit buttons without a value (issue 1472)
- Dict-like settings now have per-key priorities (issue 1135, issue 1149 and issue 1586).
- Sending non-ASCII emails (issue 1662)
- `CloseSpider` and `SpiderState` extensions now get disabled if no relevant setting is set (issue 1723, issue 1725).
- Added method `ExecutionEngine.close` (issue 1423).
- Added method `CrawlerRunner.create_crawler` (issue 1528).
- Scheduler priority queue can now be customized via *SCHEDULER_PRIORITY_QUEUE* (issue 1822).
- `.pps` links are now ignored by default in link extractors (issue 1835).
- temporary data folder for FTP and S3 feed storages can be customized using a new *FEED_TEMPDIR* setting (issue 1847).
- `FilesPipeline` and `ImagesPipeline` settings are now instance attributes instead of class attributes, enabling spider-specific behaviors (issue 1891).
- `JsonItemExporter` now formats opening and closing square brackets on their own line (first and last lines of output file) (issue 1950).

- If available, `botocore` is used for `S3FeedStorage`, `S3DownloadHandler` and `S3FilesStore` (issue 1761, issue 1883).
- Tons of documentation updates and related fixes (issue 1291, issue 1302, issue 1335, issue 1683, issue 1660, issue 1642, issue 1721, issue 1727, issue 1879).
- Other refactoring, optimizations and cleanup (issue 1476, issue 1481, issue 1477, issue 1315, issue 1290, issue 1750, issue 1881).

非推奨と削除

- Added `to_bytes` and `to_unicode`, deprecated `str_to_unicode` and `unicode_to_str` functions (issue 778).
- `binary_is_text` is introduced, to replace use of `isbinarytext` (but with inverse return value) (issue 1851)
- The `optional_features` set has been removed (issue 1359).
- The `--lsprof` command line option has been removed (issue 1689). **Warning: backward incompatible**, but doesn't break user code.
- The following datatypes were deprecated (issue 1720):
  - `scrapy.utils.datatypes.MultiValueDictKeyError`
  - `scrapy.utils.datatypes.MultiValueDict`
  - `scrapy.utils.datatypes.SiteNode`
- The previously bundled `scrapy.xlib.pydispatch` library was deprecated and replaced by pydispatcher.

Relocations

- `telnetconsole` was relocated to `extensions/` (issue 1524).
  - Note: telnet is not enabled on Python 3 (https://github.com/scrapy/scrapy/pull/1524#issuecomment-146985595)

Bugfixes

- Scrapy does not retry requests that got a `HTTP 400 Bad Request` response anymore (issue 1289). **Warning: backward incompatible!**
- Support empty password for http_proxy config (issue 1274).
- Interpret `application/x-json` as `TextResponse` (issue 1333).

- Support link rel attribute with multiple values (issue 1201).
- Fixed `scrapy.http.FormRequest.from_response` when there is a `<base>` tag (issue 1564).
- Fixed *TEMPLATES_DIR* handling (issue 1575).
- Various `FormRequest` fixes (issue 1595, issue 1596, issue 1597).
- Makes `_monkeypatches` more robust (issue 1634).
- Fixed bug on `XMLItemExporter` with non-string fields in items (issue 1738).
- Fixed startproject command in OS X (issue 1635).
- Fixed `PythonItemExporter` and CSVExporter for non-string item types (issue 1737).
- Various logging related fixes (issue 1294, issue 1419, issue 1263, issue 1624, issue 1654, issue 1722, issue 1726 and issue 1303).
- Fixed bug in `utils.template.render_templatefile()` (issue 1212).
- sitemaps extraction from `robots.txt` is now case-insensitive (issue 1902).
- HTTPS+CONNECT tunnels could get mixed up when using multiple proxies to same remote host (issue 1912).

7.1.23 Scrapy 1.0.7 (2017-03-03)

- パッケージの修正：setup.py でサポートされていない Twisted バージョンを禁止します。

7.1.24 Scrapy 1.0.6 (2016-05-04)

- FIX: RetryMiddleware is now robust to non-standard HTTP status codes (issue 1857)
- FIX: Filestorage HTTP cache was checking wrong modified time (issue 1875)
- DOC: Support for Sphinx 1.4+ (issue 1893)
- DOC: Consistency in selectors examples (issue 1869)

7.1.25 Scrapy 1.0.5 (2016-02-04)

- FIX: [Backport] Ignore bogus links in LinkExtractors (fixes issue 907, commit 108195e)
- TST: Changed buildbot makefile to use 'pytest' (commit 1f3d90a)
- DOC: Fixed typos in tutorial and media-pipeline (commit 808a9ea and commit 803bd87)

- DOC: Add AjaxCrawlMiddleware to DOWNLOADER_MIDDLEWARES_BASE in settings docs (commit aa94121)

7.1.26 Scrapy 1.0.4 (2015-12-30)

- Ignoring xlib/tx folder, depending on Twisted version. (commit 7dfa979)
- Run on new travis-ci infra (commit 6e42f0b)
- Spelling fixes (commit 823a1cc)
- escape nodename in xmliter regex (commit da3c155)
- test xml nodename with dots (commit 4418fc3)
- TST don't use broken Pillow version in tests (commit a55078c)
- disable log on version command. closes #1426 (commit 86fc330)
- disable log on startproject command (commit db4c9fe)
- Add PyPI download stats badge (commit df2b944)
- don't run tests twice on Travis if a PR is made from a scrapy/scrapy branch (commit a83ab41)
- Add Python 3 porting status badge to the README (commit 73ac80d)
- fixed RFPDupeFilter persistence (commit 97d080e)
- TST a test to show that dupefilter persistence is not working (commit 97f2fb3)
- explicit close file on file:// scheme handler (commit d9b4850)
- Disable dupefilter in shell (commit c0d0734)
- DOC: Add captions to toctrees which appear in sidebar (commit aa239ad)
- DOC Removed pywin32 from install instructions as it's already declared as dependency. (commit 10eb400)
- Added installation notes about using Conda for Windows and other OSes. (commit 1c3600a)
- Fixed minor grammar issues. (commit 7f4ddd5)
- fixed a typo in the documentation. (commit b71f677)
- Version 1 now exists (commit 5456c0e)
- fix another invalid xpath error (commit 0a1366e)
- fix ValueError: Invalid XPath: //div/[id="not-exists"]/text() on selectors.rst (commit ca8d60f)

- Typos corrections (commit 7067117)
- fix typos in downloader-middleware.rst and exceptions.rst, middlware -> middleware (commit 32f115c)
- Add note to ubuntu install section about debian compatibility (commit 23fda69)
- Replace alternative OSX install workaround with virtualenv (commit 98b63ee)
- Reference Homebrew's homepage for installation instructions (commit 1925db1)
- Add oldest supported tox version to contributing docs (commit 5d10d6d)
- Note in install docs about pip being already included in python>=2.7.9 (commit 85c980e)
- Add non-python dependencies to Ubuntu install section in the docs (commit fbd010d)
- Add OS X installation section to docs (commit d8f4cba)
- DOC(ENH): specify path to rtd theme explicitly (commit de73b1a)
- minor: scrapy.Spider docs grammar (commit 1ddcc7b)
- Make common practices sample code match the comments (commit 1b85bcf)
- nextcall repetitive calls (heartbeats). (commit 55f7104)
- Backport fix compatibility with Twisted 15.4.0 (commit b262411)
- pin pytest to 2.7.3 (commit a6535c2)
- Merge pull request #1512 from mgedmin/patch-1 (commit 8876111)
- Merge pull request #1513 from mgedmin/patch-2 (commit 5d4daf8)
- Typo (commit f8d0682)
- Fix list formatting (commit 5f83a93)
- fix scrapy squeue tests after recent changes to queuelib (commit 3365c01)
- Merge pull request #1475 from rweindl/patch-1 (commit 2d688cd)
- Update tutorial.rst (commit fbc1f25)
- Merge pull request #1449 from rhoekman/patch-1 (commit 7d6538c)
- Small grammatical change (commit 8752294)
- Add openssl version to version command (commit 13c45ac)

7.1.27 Scrapy 1.0.3 (2015-08-11)

- add service_identity to scrapy install_requires (commit cbc2501)
- Workaround for travis#296 (commit 66af9cd)

7.1.28 Scrapy 1.0.2 (2015-08-06)

- Twisted 15.3.0 does not raises PicklingError serializing lambda functions (commit b04dd7d)
- Minor method name fix (commit 6f85c7f)
- minor: scrapy.Spider grammar and clarity (commit 9c9d2e0)
- Put a blurb about support channels in CONTRIBUTING (commit c63882b)
- Fixed typos (commit a9ae7b0)
- Fix doc reference. (commit 7c8a4fe)

7.1.29 Scrapy 1.0.1 (2015-07-01)

- Unquote request path before passing to FTPClient, it already escape paths (commit cc00ad2)
- include tests/ to source distribution in MANIFEST.in (commit eca227e)
- DOC Fix SelectJmes documentation (commit b8567bc)
- DOC Bring Ubuntu and Archlinux outside of Windows subsection (commit 392233f)
- DOC remove version suffix from ubuntu package (commit 5303c66)
- DOC Update release date for 1.0 (commit c89fa29)

7.1.30 Scrapy 1.0.0 (2015-06-19)

このメジャーリリースには、多くの新機能とバグ修正が含まれています。 更新された *overview* を確認して、磨かれた **チュートリアル** とともに変更の一部を確認してください。

スパイダーで辞書を返すためのサポート

スパイダーからスクレイピングされたデータを収集するために、スクレイピーアイテムを宣言して返す必要はなくなりました。代わりに、明示的な辞書を返すことができるようになりました。

Classic version

```
class MyItem(scrapy.Item):
    url = scrapy.Field()

class MySpider(scrapy.Spider):
    def parse(self, response):
        return MyItem(url=response.url)
```

New version

```
class MySpider(scrapy.Spider):
    def parse(self, response):
        return {'url': response.url}
```

Per-spider settings (GSoC 2014)

Last Google Summer of Code project accomplished an important redesign of the mechanism used for populating settings, introducing explicit priorities to override any given setting. As an extension of that goal, we included a new level of priority for settings that act exclusively for a single spider, allowing them to redefine project settings.

Start using it by defining a *custom_settings* class variable in your spider:

```
class MySpider(scrapy.Spider):
    custom_settings = {
        "DOWNLOAD_DELAY": 5.0,
        "RETRY_ENABLED": False,
    }
```

Read more about settings population: **設定**

Python Logging

Scrapy 1.0 has moved away from Twisted logging to support Python built in ' s as default logging system. We ' re maintaining backward compatibility for most of the old custom interface to call logging functions, but you ' ll get warnings to switch to the Python logging API entirely.

Old version

```
from scrapy import log
log.msg('MESSAGE', log.INFO)
```

New version

```
import logging
logging.info('MESSAGE')
```

Logging with spiders remains the same, but on top of the *log()* method you ' ll have access to a custom *logger* created for the spider to issue log events:

```
class MySpider(scrapy.Spider):
    def parse(self, response):
        self.logger.info('Response received')
```

Read more in the logging documentation: **ロギング** *(logging)*

Crawler API refactoring (GSoC 2014)

Another milestone for last Google Summer of Code was a refactoring of the internal API, seeking a simpler and easier usage. Check new core interface in: **コア** *API*

A common situation where you will face these changes is while running Scrapy from scripts. Here ' s a quick example of how to run a Spider manually with the new API:

```
from scrapy.crawler import CrawlerProcess

process = CrawlerProcess({
    'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)'
})
process.crawl(MySpider)
process.start()
```

Bear in mind this feature is still under development and its API may change until it reaches a stable status.

See more examples for scripts running Scrapy: **よくある例**

モジュールの再配置

There ' s been a large rearrangement of modules trying to improve the general structure of Scrapy. Main changes were separating various subpackages into new projects and dissolving both `scrapy.contrib` and `scrapy.contrib_exp` into top level packages. Backward compatibility was kept among internal relocations, while importing deprecated modules expect warnings indicating their new place.

Full list of relocations

Outsourced packages

注釈: These extensions went through some minor changes, e.g. some setting names were changed. Please check the documentation in each new repository to get familiar with the new usage.

古い場所	新しい場所
scrapy.commands.deploy	scrapyd-client (See other alternatives here: スパイダーのデプロイ)
scrapy.contrib.djangoitem	scrapy-djangoitem
scrapy.webservice	scrapy-jsonrpc

`scrapy.contrib_exp` and `scrapy.contrib` dissolutions

古い場所	新しい場所
scrapy.contrib_exp.downloadermiddleware.decompression	scrapy.downloadermiddlewares.decompression
scrapy.contrib_exp.iterators	scrapy.utils.iterators
scrapy.contrib.downloadermiddleware	scrapy.downloadermiddlewares
scrapy.contrib.exporter	scrapy.exporters
scrapy.contrib.linkextractors	scrapy.linkextractors
scrapy.contrib.loader	scrapy.loader
scrapy.contrib.loader.processor	scrapy.loader.processors
scrapy.contrib.pipeline	scrapy.pipelines
scrapy.contrib.spidermiddleware	scrapy.spidermiddlewares
scrapy.contrib.spiders	scrapy.spiders
• scrapy.contrib.closespider • scrapy.contrib.corestats • scrapy.contrib.debug • scrapy.contrib.feedexport • scrapy.contrib.httpcache • scrapy.contrib.logstats • scrapy.contrib.memdebug • scrapy.contrib.memusage • scrapy.contrib.spiderstate • scrapy.contrib.statsmailer • scrapy.contrib.throttle	scrapy.extensions.*

Plural renames and Modules unification

古い場所	新しい場所
scrapy.command	scrapy.commands
scrapy.dupefilter	scrapy.dupefilters
scrapy.linkextractor	scrapy.linkextractors
scrapy.spider	scrapy.spiders
scrapy.squeue	scrapy.squeues
scrapy.statscol	scrapy.statscollectors
scrapy.utils.decorator	scrapy.utils.decorators

Class renames

古い場所	新しい場所
scrapy.spidermanager.SpiderManager	scrapy.spiderloader.SpiderLoader

Settings renames

古い場所	新しい場所
SPIDER_MANAGER_CLASS	SPIDER_LOADER_CLASS

Changelog

New Features and Enhancements

- Python logging (issue 1060, issue 1235, issue 1236, issue 1240, issue 1259, issue 1278, issue 1286)
- FEED_EXPORT_FIELDS option (issue 1159, issue 1224)
- Dns cache size and timeout options (issue 1132)
- support namespace prefix in xmliter_lxml (issue 963)
- Reactor threadpool max size setting (issue 1123)
- Allow spiders to return dicts. (issue 1081)
- Add Response.urljoin() helper (issue 1086)
- look in ~/.config/scrapy.cfg for user config (issue 1098)
- handle TLS SNI (issue 1101)
- Selectorlist extract first (issue 624, issue 1145)
- Added JmesSelect (issue 1016)

- add gzip compression to filesystem http cache backend (issue 1020)
- CSS support in link extractors (issue 983)
- httpcache dont_cache meta #19 #689 (issue 821)
- add signal to be sent when request is dropped by the scheduler (issue 961)
- avoid download large response (issue 946)
- Allow to specify the quotechar in CSVFeedSpider (issue 882)
- Add referer to "Spider error processing" log message (issue 795)
- process robots.txt once (issue 896)
- GSoC Per-spider settings (issue 854)
- Add project name validation (issue 817)
- GSoC API cleanup (issue 816, issue 1128, issue 1147, issue 1148, issue 1156, issue 1185, issue 1187, issue 1258, issue 1268, issue 1276, issue 1285, issue 1284)
- Be more responsive with IO operations (issue 1074 and issue 1075)
- Do leveldb compaction for httpcache on closing (issue 1297)

非推奨と削除

- Deprecate htmlparser link extractor (issue 1205)
- remove deprecated code from FeedExporter (issue 1155)
- a leftover for.15 compatibility (issue 925)
- drop support for CONCURRENT_REQUESTS_PER_SPIDER (issue 895)
- Drop old engine code (issue 911)
- Deprecate SgmlLinkExtractor (issue 777)

Relocations

- Move exporters/__init__.py to exporters.py (issue 1242)
- Move base classes to their packages (issue 1218, issue 1233)
- Module relocation (issue 1181, issue 1210)
- rename SpiderManager to SpiderLoader (issue 1166)
- Remove djangoitem (issue 1177)
- remove scrapy deploy command (issue 1102)

- dissolve contrib_exp (issue 1134)
- Deleted bin folder from root, fixes #913 (issue 914)
- Remove jsonrpc based webservice (issue 859)
- Move Test cases under project root dir (issue 827, issue 841)
- Fix backward incompatibility for relocated paths in settings (issue 1267)

Documentation

- CrawlerProcess documentation (issue 1190)
- Favoring web scraping over screen scraping in the descriptions (issue 1188)
- Some improvements for Scrapy tutorial (issue 1180)
- Documenting Files Pipeline together with Images Pipeline (issue 1150)
- deployment docs tweaks (issue 1164)
- Added deployment section covering scrapyd-deploy and shub (issue 1124)
- Adding more settings to project template (issue 1073)
- some improvements to overview page (issue 1106)
- Updated link in docs/topics/architecture.rst (issue 647)
- DOC reorder topics (issue 1022)
- updating list of Request.meta special keys (issue 1071)
- DOC document download_timeout (issue 898)
- DOC simplify extension docs (issue 893)
- Leaks docs (issue 894)
- DOC document from_crawler method for item pipelines (issue 904)
- Spider_error doesn't support deferreds (issue 1292)
- Corrections & Sphinx related fixes (issue 1220, issue 1219, issue 1196, issue 1172, issue 1171, issue 1169, issue 1160, issue 1154, issue 1127, issue 1112, issue 1105, issue 1041, issue 1082, issue 1033, issue 944, issue 866, issue 864, issue 796, issue 1260, issue 1271, issue 1293, issue 1298)

Bugfixes

- Item multi inheritance fix (issue 353, issue 1228)
- ItemLoader.load_item: iterate over copy of fields (issue 722)

- Fix Unhandled error in Deferred (RobotsTxtMiddleware) (issue 1131, issue 1197)
- Force to read DOWNLOAD_TIMEOUT as int (issue 954)
- scrapy.utils.misc.load_object should print full traceback (issue 902)
- Fix bug for ".local" host name (issue 878)
- Fix for Enabled extensions, middlewares, pipelines info not printed anymore (issue 879)
- fix dont_merge_cookies bad behaviour when set to false on meta (issue 846)

Python 3 In Progress Support

- disable scrapy.telnet if twisted.conch is not available (issue 1161)
- fix Python 3 syntax errors in ajaxcrawl.py (issue 1162)
- more python3 compatibility changes for urllib (issue 1121)
- assertItemsEqual was renamed to assertCountEqual in Python 3. (issue 1070)
- Import unittest.mock if available. (issue 1066)
- updated deprecated cgi.parse_qsl to use six's parse_qsl (issue 909)
- Prevent Python 3 port regressions (issue 830)
- PY3: use MutableMapping for python 3 (issue 810)
- PY3: use six.BytesIO and six.moves.cStringIO (issue 803)
- PY3: fix xmlrpclib and email imports (issue 801)
- PY3: use six for robotparser and urlparse (issue 800)
- PY3: use six.iterkeys, six.iteritems, and tempfile (issue 799)
- PY3: fix has_key and use six.moves.configparser (issue 798)
- PY3: use six.moves.cPickle (issue 797)
- PY3 make it possible to run some tests in Python3 (issue 776)

Tests

- remove unnecessary lines from py3-ignores (issue 1243)
- Fix remaining warnings from pytest while collecting tests (issue 1206)
- Add docs build to travis (issue 1234)
- TST don't collect tests from deprecated modules. (issue 1165)

- install service_identity package in tests to prevent warnings (issue 1168)
- Fix deprecated settings API in tests (issue 1152)
- Add test for webclient with POST method and no body given (issue 1089)
- py3-ignores.txt supports comments (issue 1044)
- modernize some of the asserts (issue 835)
- selector.__repr__ test (issue 779)

Code refactoring

- CSVFeedSpider cleanup: use iterate_spider_output (issue 1079)
- remove unnecessary check from scrapy.utils.spider.iter_spider_output (issue 1078)
- Pydispatch pep8 (issue 992)
- Removed unused 'load=False' parameter from walk_modules() (issue 871)
- For consistency, use `job_dir` helper in `SpiderState` extension. (issue 805)
- rename "sflo" local variables to less cryptic "log_observer" (issue 775)

7.1.31 Scrapy 0.24.6 (2015-04-20)

- encode invalid xpath with unicode_escape under PY2 (commit 07cb3e5)
- fix IPython shell scope issue and load IPython user config (commit 2c8e573)
- Fix small typo in the docs (commit d694019)
- Fix small typo (commit f92fa83)
- Converted sel.xpath() calls to response.xpath() in Extracting the data (commit c2c6d15)

7.1.32 Scrapy 0.24.5 (2015-02-25)

- Support new _getEndpoint Agent signatures on Twisted 15.0.0 (commit 540b9bc)
- DOC a couple more references are fixed (commit b4c454b)
- DOC fix a reference (commit e3c1260)
- t.i.b.ThreadedResolver is now a new-style class (commit 9e13f42)
- S3DownloadHandler: fix auth for requests with quoted paths/query params (commit cdb9a0b)

- fixed the variable types in mailsender documentation (commit bb3a848)
- Reset items_scraped instead of item_count (commit edb07a4)
- Tentative attention message about what document to read for contributions (commit 7ee6f7a)
- mitmproxy 0.10.1 needs netlib 0.10.1 too (commit 874fcdd)
- pin mitmproxy 0.10.1 as >0.11 does not work with tests (commit c6b21f0)
- Test the parse command locally instead of against an external url (commit c3a6628)
- Patches Twisted issue while closing the connection pool on HTTPDownloadHandler (commit d0bf957)
- Updates documentation on dynamic item classes. (commit eeb589a)
- Merge pull request #943 from Lazar-T/patch-3 (commit 5fdab02)
- typo (commit b0ae199)
- pywin32 is required by Twisted. closes #937 (commit 5cb0cfb)
- Update install.rst (commit 781286b)
- Merge pull request #928 from Lazar-T/patch-1 (commit b415d04)
- comma instead of fullstop (commit 627b9ba)
- Merge pull request #885 from jsma/patch-1 (commit de909ad)
- Update request-response.rst (commit 3f3263d)
- SgmlLinkExtractor - fix for parsing <area> tag with Unicode present (commit 49b40f0)

7.1.33 Scrapy 0.24.4 (2014-08-09)

- pem file is used by mockserver and required by scrapy bench (commit 5eddc68)
- scrapy bench needs scrapy.tests* (commit d6cb999)

7.1.34 Scrapy 0.24.3 (2014-08-09)

- no need to waste travis-ci time on py3 for 0.24 (commit 8e080c1)
- Update installation docs (commit 1d0c096)
- There is a trove classifier for Scrapy framework! (commit 4c701d7)
- update other places where w3lib version is mentioned (commit d109c13)

- Update w3lib requirement to 1.8.0 (commit 39d2ce5)
- Use w3lib.html.replace_entities() (remove_entities() is deprecated) (commit 180d3ad)
- set zip_safe=False (commit a51ee8b)
- do not ship tests package (commit ee3b371)
- scrapy.bat is not needed anymore (commit c3861cf)
- Modernize setup.py (commit 362e322)
- headers can not handle non-string values (commit 94a5c65)
- fix ftp test cases (commit a274a7f)
- The sum up of travis-ci builds are taking like 50min to complete (commit ae1e2cc)
- Update shell.rst typo (commit e49c96a)
- removes weird indentation in the shell results (commit 1ca489d)
- improved explanations, clarified blog post as source, added link for XPath string functions in the spec (commit 65c8f05)
- renamed UserTimeoutError and ServerTimeouterror #583 (commit 037f6ab)
- adding some xpath tips to selectors docs (commit 2d103e0)
- fix tests to account for https://github.com/scrapy/w3lib/pull/23 (commit f8d366a)
- get_func_args maximum recursion fix #728 (commit 81344ea)
- Updated input/ouput processor example according to #560. (commit f7c4ea8)
- Fixed Python syntax in tutorial. (commit db59ed9)
- Add test case for tunneling proxy (commit f090260)
- Bugfix for leaking Proxy-Authorization header to remote host when using tunneling (commit d8793af)
- Extract links from XHTML documents with MIME-Type "application/xml" (commit ed1f376)
- Merge pull request #793 from roysc/patch-1 (commit 91a1106)
- Fix typo in commands.rst (commit 743e1e2)
- better testcase for settings.overrides.setdefault (commit e22daaf)
- Using CRLF as line marker according to http 1.1 definition (commit 5ec430b)

7.1.35 Scrapy 0.24.2 (2014-07-08)

- Use a mutable mapping to proxy deprecated settings.overrides and settings.defaults attribute (commit e5e8133)
- there is not support for python3 yet (commit 3cd6146)
- Update python compatible version set to debian packages (commit fa5d76b)
- DOC fix formatting in release notes (commit c6a9e20)

7.1.36 Scrapy 0.24.1 (2014-06-27)

- Fix deprecated CrawlerSettings and increase backward compatibility with .defaults attribute (commit 8e3f20a)

7.1.37 Scrapy 0.24.0 (2014-06-26)

Enhancements

- Improve Scrapy top-level namespace (issue 494, issue 684)
- Add selector shortcuts to responses (issue 554, issue 690)
- Add new lxml based LinkExtractor to replace unmantained SgmlLinkExtractor (issue 559, issue 761, issue 763)
- Cleanup settings API - part of per-spider settings **GSoC project** (issue 737)
- Add UTF8 encoding header to templates (issue 688, issue 762)
- Telnet console now binds to 127.0.0.1 by default (issue 699)
- Update debian/ubuntu install instructions (issue 509, issue 549)
- Disable smart strings in lxml XPath evaluations (issue 535)
- Restore filesystem based cache as default for http cache middleware (issue 541, issue 500, issue 571)
- Expose current crawler in Scrapy shell (issue 557)
- Improve testsuite comparing CSV and XML exporters (issue 570)
- New `offsite/filtered` and `offsite/domains` stats (issue 566)
- Support process_links as generator in CrawlSpider (issue 555)
- Verbose logging and new stats counters for DupeFilter (issue 553)
- Add a mimetype parameter to `MailSender.send()` (issue 602)
- Generalize file pipeline log messages (issue 622)

- Replace unencodeable codepoints with html entities in SGMLLinkExtractor (issue 565)
- Converted SEP documents to rst format (issue 629, issue 630, issue 638, issue 632, issue 636, issue 640, issue 635, issue 634, issue 639, issue 637, issue 631, issue 633, issue 641, issue 642)
- Tests and docs for clickdata's nr index in FormRequest (issue 646, issue 645)
- Allow to disable a downloader handler just like any other component (issue 650)
- Log when a request is discarded after too many redirections (issue 654)
- Log error responses if they are not handled by spider callbacks (issue 612, issue 656)
- Add content-type check to http compression mw (issue 193, issue 660)
- Run pypy tests using latest pypi from ppa (issue 674)
- Run test suite using pytest instead of trial (issue 679)
- Build docs and check for dead links in tox environment (issue 687)
- Make scrapy.version_info a tuple of integers (issue 681, issue 692)
- Infer exporter's output format from filename extensions (issue 546, issue 659, issue 760)
- Support case-insensitive domains in `url_is_from_any_domain()` (issue 693)
- Remove pep8 warnings in project and spider templates (issue 698)
- Tests and docs for `request_fingerprint` function (issue 597)
- Update SEP-19 for GSoC project `per-spider settings` (issue 705)
- Set exit code to non-zero when contracts fails (issue 727)
- Add a setting to control what class is instanciated as Downloader component (issue 738)
- Pass response in `item_dropped` signal (issue 724)
- Improve `scrapy check` contracts command (issue 733, issue 752)
- Document `spider.closed()` shortcut (issue 719)
- Document `request_scheduled` signal (issue 746)
- Add a note about reporting security issues (issue 697)
- Add LevelDB http cache storage backend (issue 626, issue 500)
- Sort spider list output of `scrapy list` command (issue 742)
- Multiple documentation enhancemens and fixes (issue 575, issue 587, issue 590, issue 596, issue 610, issue 617, issue 618, issue 627, issue 613, issue 643, issue 654, issue 675, issue 663, issue 711, issue 714)

Bugfixes

- Encode unicode URL value when creating Links in RegexLinkExtractor (issue 561)
- Ignore None values in ItemLoader processors (issue 556)
- Fix link text when there is an inner tag in SGMLLinkExtractor and HtmlParserLinkExtractor (issue 485, issue 574)
- Fix wrong checks on subclassing of deprecated classes (issue 581, issue 584)
- Handle errors caused by inspect.stack() failures (issue 582)
- Fix a reference to unexistent engine attribute (issue 593, issue 594)
- Fix dynamic itemclass example usage of type() (issue 603)
- Use lucasdemarchi/codespell to fix typos (issue 628)
- Fix default value of attrs argument in SgmlLinkExtractor to be tuple (issue 661)
- Fix XXE flaw in sitemap reader (issue 676)
- Fix engine to support filtered start requests (issue 707)
- Fix offsite middleware case on urls with no hostnames (issue 745)
- Testsuite doesn't require PIL anymore (issue 585)

7.1.38 Scrapy 0.22.2 (released 2014-02-14)

- fix a reference to unexistent engine.slots. closes #593 (commit 13c099a)
- downloaderMW doc typo (spiderMW doc copy remnant) (commit 8ae11bf)
- Correct typos (commit 1346037)

7.1.39 Scrapy 0.22.1 (released 2014-02-08)

- localhost666 can resolve under certain circumstances (commit 2ec2279)
- test inspect.stack failure (commit cc3eda3)
- Handle cases when inspect.stack() fails (commit 8cb44f9)
- Fix wrong checks on subclassing of deprecated classes. closes #581 (commit 46d98d6)
- Docs: 4-space indent for final spider example (commit 13846de)

- Fix HtmlParserLinkExtractor and tests after #485 merge (commit 368a946)
- BaseSgmlLinkExtractor: Fixed the missing space when the link has an inner tag (commit b566388)
- BaseSgmlLinkExtractor: Added unit test of a link with an inner tag (commit c1cb418)
- BaseSgmlLinkExtractor: Fixed unknown_endtag() so that it only set current_link=None when the end tag match the opening tag (commit 7e4d627)
- Fix tests for Travis-CI build (commit 76c7e20)
- replace unencodeable codepoints with html entities. fixes #562 and #285 (commit 5f87b17)
- RegexLinkExtractor: encode URL unicode value when creating Links (commit d0ee545)
- Updated the tutorial crawl output with latest output. (commit 8da65de)
- Updated shell docs with the crawler reference and fixed the actual shell output. (commit 875b9ab)
- PEP8 minor edits. (commit f89efaf)
- Expose current crawler in the scrapy shell. (commit 5349cec)
- Unused re import and PEP8 minor edits. (commit 387f414)
- Ignore None's values when using the ItemLoader. (commit 0632546)
- DOC Fixed HTTPCACHE_STORAGE typo in the default value which is now Filesystem instead Dbm. (commit cde9a8c)
- show ubuntu setup instructions as literal code (commit fb5c9c5)
- Update Ubuntu installation instructions (commit 70fb105)
- Merge pull request #550 from stray-leone/patch-1 (commit 6f70b6a)
- modify the version of scrapy ubuntu package (commit 725900d)
- fix 0.22.0 release date (commit af0219a)
- fix typos in news.rst and remove (not released yet) header (commit b7f58f4)

7.1.40 Scrapy 0.22.0 (released 2014-01-17)

Enhancements

- [**Backward incompatible**] Switched HTTPCacheMiddleware backend to filesystem (issue 541) To restore old backend set `HTTPCACHE_STORAGE` to `scrapy.contrib.httpcache.DbmCacheStorage`
- Proxy https:// urls using CONNECT method (issue 392, issue 397)

- Add a middleware to crawl ajax crawleable pages as defined by google (issue 343)
- Rename scrapy.spider.BaseSpider to scrapy.spider.Spider (issue 510, issue 519)
- Selectors register EXSLT namespaces by default (issue 472)
- Unify item loaders similar to selectors renaming (issue 461)
- Make `RFPDupeFilter` class easily subclassable (issue 533)
- Improve test coverage and forthcoming Python 3 support (issue 525)
- Promote startup info on settings and middleware to INFO level (issue 520)
- Support partials in `get_func_args` util (issue 506, issue:*504*)
- Allow running indiviual tests via tox (issue 503)
- Update extensions ignored by link extractors (issue 498)
- Add middleware methods to get files/images/thumbs paths (issue 490)
- Improve offsite middleware tests (issue 478)
- Add a way to skip default Referer header set by RefererMiddleware (issue 475)
- Do not send `x-gzip` in default `Accept-Encoding` header (issue 469)
- Support defining http error handling using settings (issue 466)
- Use modern python idioms wherever you find legacies (issue 497)
- Improve and correct documentation (issue 527, issue 524, issue 521, issue 517, issue 512, issue 505, issue 502, issue 489, issue 465, issue 460, issue 425, issue 536)

Fixes

- Update Selector class imports in CrawlSpider template (issue 484)
- Fix unexistent reference to `engine.slots` (issue 464)
- Do not try to call `body_as_unicode()` on a non-TextResponse instance (issue 462)
- Warn when subclassing XPathItemLoader, previously it only warned on instantiation. (issue 523)
- Warn when subclassing XPathSelector, previously it only warned on instantiation. (issue 537)
- Multiple fixes to memory stats (issue 531, issue 530, issue 529)
- Fix overriding url in `FormRequest.from_response()` (issue 507)
- Fix tests runner under pip 1.5 (issue 513)

- Fix logging error when spider name is unicode (issue 479)

7.1.41 Scrapy 0.20.2 (released 2013-12-09)

- Update CrawlSpider Template with Selector changes (commit 6d1457d)
- fix method name in tutorial. closes GH-480 (commit b4fc359

7.1.42 Scrapy 0.20.1 (released 2013-11-28)

- include_package_data is required to build wheels from published sources (commit 5ba1ad5)
- process_parallel was leaking the failures on its internal deferreds. closes #458 (commit 419a780)

7.1.43 Scrapy 0.20.0 (released 2013-11-08)

Enhancements

- New Selector's API including CSS selectors (issue 395 and issue 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)
- `Selector.remove_namespaces()` now remove namespaces from element's attributes. (issue 416)
- Paved the road for Python 3.3+ (issue 435, issue 436, issue 431, issue 452)
- New item exporter using native python types with nesting support (issue 366)
- Tune HTTP1.1 pool size so it matches concurrency defined by settings (commit b43b5f575)
- scrapy.mail.MailSender now can connect over TLS or upgrade using STARTTLS (issue 327)
- New FilesPipeline with functionality factored out from ImagesPipeline (issue 370, issue 409)
- Recommend Pillow instead of PIL for image handling (issue 317)
- Added debian packages for Ubuntu quantal and raring (commit 86230c0)
- Mock server (used for tests) can listen for HTTPS requests (issue 410)
- Remove multi spider support from multiple core components (issue 422, issue 421, issue 420, issue 419, issue 423, issue 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)
- Pylinted, pep8 and removed old-style exceptions from source (issue 430, issue 432)
- Use importlib for parametric imports (issue 445)
- Handle a regression introduced in Python 2.7.5 that affects XmlItemExporter (issue 372)
- Bugfix crawling shutdown on SIGINT (issue 450)
- Do not submit `reset` type inputs in FormRequest.from_response (commit b326b87)
- Do not silence download errors when request errback raises an exception (commit 684cfc0)

Bugfixes

- Fix tests under Django 1.6 (commit b6bed44c)
- Lot of bugfixes to retry middleware under disconnections using HTTP 1.1 download handler
- Fix inconsistencies among Twisted releases (issue 406)
- Fix scrapy shell bugs (issue 418, issue 407)
- Fix invalid variable name in setup.py (issue 429)
- Fix tutorial references (issue 387)
- Improve request-response docs (issue 391)
- Improve best practices docs (issue 399, issue 400, issue 401, issue 402)
- Improve django integration docs (issue 404)
- Document `bindaddress` request meta (commit 37c24e01d7)
- Improve `Request` class documentation (issue 226)

Other

- Dropped Python 2.6 support (issue 448)
- Add cssselect python package as install dependency
- Drop libxml2 and multi selector's backend support, lxml 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)

Thanks

Thanks to everyone who contribute to this release!

List of contributors sorted by number of commits:

```
69 Daniel Gra^^c3^^b1a <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@...>
```

7.1.44 Scrapy 0.18.4 (released 2013-10-10)

- IPython refuses to update the namespace. fix #396 (commit 3d32c4f)
- Fix AlreadyCalledError replacing a request in shell command. closes #407 (commit b1d8919)
- Fix start_requests laziness and early hangs (commit 89faf52)

7.1.45 Scrapy 0.18.3 (released 2013-10-03)

- fix regression on lazy evaluation of start requests (commit 12693a5)
- forms: do not submit reset inputs (commit e429f63)
- increase unittest timeouts to decrease travis false positive failures (commit 912202e)

- backport master fixes to json exporter (commit cfc2d46)
- Fix permission and set umask before generating sdist tarball (commit 06149e0)

7.1.46 Scrapy 0.18.2 (released 2013-09-03)

- Backport `scrapy check` command fixes and backward compatible multi crawler process(issue 339)

7.1.47 Scrapy 0.18.1 (released 2013-08-27)

- remove extra import added by cherry picked changes (commit d20304e)
- fix crawling tests under twisted pre 11.0.0 (commit 1994f38)
- py26 can not format zero length fields {} (commit abf756f)
- test PotentiaDataLoss errors on unbound responses (commit b15470d)
- Treat responses without content-length or Transfer-Encoding as good responses (commit c4bf324)
- do no include ResponseFailed if http11 handler is not enabled (commit 6cbe684)
- New HTTP client wraps connection losts in ResponseFailed exception. fix #373 (commit 1a20bba)
- limit travis-ci build matrix (commit 3b01bb8)
- Merge pull request #375 from peterarenot/patch-1 (commit fa766d7)
- Fixed so it refers to the correct folder (commit 3283809)
- added quantal & raring to support ubuntu releases (commit 1411923)
- fix retry middleware which didn't retry certain connection errors after the upgrade to http1 client, closes GH-373 (commit bb35ed0)
- fix XmlItemExporter in Python 2.7.4 and 2.7.5 (commit de3e451)
- minor updates to 0.18 release notes (commit c45e5f1)
- fix contributters list format (commit 0b60031)

7.1.48 Scrapy 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)

- Use lxml recover option to parse sitemaps (issue 347)
- Bugfix cookie merging by hostname and not by netloc (issue 352)
- Support disabling `HttpCompressionMiddleware` using a flag setting (issue 359)
- Support xml namespaces using `iternodes` parser in `XMLFeedSpider` (issue 12)
- Support `dont_cache` request meta flag (issue 19)
- Bugfix `scrapy.utils.gz.gunzip` broken by changes in python 2.7.4 (commit 4dc76e)
- Bugfix url encoding on `SgmlLinkExtractor` (issue 24)
- Bugfix `TakeFirst` processor shouldn't discard zero (0) value (issue 59)
- Support nested items in xml exporter (issue 66)
- Improve cookies handling performance (issue 77)
- Log dupe filtered requests once (issue 105)
- Split redirection middleware into status and meta based middlewares (issue 78)
- Use HTTP1.1 as default downloader handler (issue 109 and issue 318)
- Support xpath form selection on `FormRequest.from_response` (issue 185)
- Bugfix unicode decoding error on `SgmlLinkExtractor` (issue 199)
- Bugfix signal dispatching on pypi interpreter (issue 205)
- Improve request delay and concurrency handling (issue 206)
- Add RFC2616 cache policy to `HttpCacheMiddleware` (issue 212)
- Allow customization of messages logged by engine (issue 214)
- Multiples improvements to `DjangoItem` (issue 217, issue 218, issue 221)
- Extend Scrapy commands using setuptools entry points (issue 260)
- Allow spider `allowed_domains` value to be set/tuple (issue 261)
- Support `settings.getdict` (issue 269)
- Simplify internal `scrapy.core.scraper` slot handling (issue 271)
- Added `Item.copy` (issue 290)
- Collect idle downloader slots (issue 297)
- Add `ftp://` scheme downloader handler (issue 329)

- Added downloader benchmark webserver and spider tools **ベンチマーキング**
- Moved persistent (on disk) queues to a separate project (queuelib) which scrapy now depends on
- Add scrapy commands using external libraries (issue 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 **セレクター**.
- 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 Gra^^c3^^b1a <dangra@...>
 20 Nicol^^c3^^a1s Ram^^c3^^adrez <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@...>
```

(次のページに続く)

(前のページからの続き)

```
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 Nenad^^c3^^a1l <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@...>
```

7.1.49 Scrapy 0.16.5 (released 2013-05-30)

- obey request method when scrapy deploy is redirected to a new endpoint (commit 8c4fcee)
- fix inaccurate downloader middleware documentation. refs #280 (commit 40667cb)
- doc: remove links to diveintopython.org, which is no longer available. closes #246 (commit bd58bfa)
- Find form nodes in invalid html5 documents (commit e3d6945)
- Fix typo labeling attrs type bool instead of list (commit a274276)

7.1.50 Scrapy 0.16.4 (released 2013-01-23)

- fixes spelling errors in documentation (commit 6d2b3aa)
- add doc about disabling an extension. refs #132 (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)
- lint and improve images pipeline error logging (commit 56b45fc)
- fixed doc typos (commit 243be84)

- add documentation topics: Broad Crawls & Common Practies (commit 1fbb715)
- fix bug in scrapy parse command when spider is not specified explicitly. closes #209 (commit c72e682)
- Update docs/topics/commands.rst (commit 28eac7a)

7.1.51 Scrapy 0.16.3 (released 2012-12-07)

- Remove concurrency limitation when using download delays and still ensure inter-request delays are enforced (commit 487b9b5)
- add error details when image pipeline fails (commit 8232569)
- improve mac os compatibility (commit 8dcf8aa)
- setup.py: use README.rst to populate long_description (commit 7b5310d)
- doc: removed obsolete references to ClientForm (commit 80f9bb6)
- correct docs for default storage backend (commit 2aa491b)
- doc: removed broken proxyhub link from FAQ (commit bdf61c4)
- Fixed docs typo in SpiderOpenCloseLogging example (commit 7184094)

7.1.52 Scrapy 0.16.2 (released 2012-11-09)

- scrapy contracts: python2.6 compat (commit a4a9199)
- scrapy contracts verbose option (commit ec41673)
- proper unittest-like output for scrapy contracts (commit 86635e4)
- added open_in_browser to debugging doc (commit c9b690d)
- removed reference to global scrapy stats from settings doc (commit dd55067)
- Fix SpiderState bug in Windows platforms (commit 58998f4)

7.1.53 Scrapy 0.16.1 (released 2012-10-26)

- fixed LogStats extension, which got broken after a wrong merge before the 0.16 release (commit 8c780fd)
- better backward compatibility for scrapy.conf.settings (commit 3403089)
- extended documentation on how to access crawler stats from extensions (commit c4da0b5)

- removed .hgtags (no longer needed now that scrapy uses git) (commit d52c188)
- fix dashes under rst headers (commit fa4f7f9)
- set release date for 0.16.0 in news (commit e292246)

7.1.54 Scrapy 0.16.0 (released 2012-10-18)

Scrapy changes:

- added **スパイダー コントラクト**, a mechanism for testing spiders in a formal/reproducible way
- added options `-o` and `-t` to the *`runspider`* command
- documented *AutoThrottle* **拡張機能** 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, backward 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 **コア** *API*
- `lxml` is now the default selectors backend instead of `libxml2`
- ported FormRequest.from_response() to use lxml instead of 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)
- StackTraceDump extension: also dump trackref live references (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, and ported Scrapy code to use that module
- dropped support for Python 2.5. See https://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 library
- promoted topics-djangoitem to main contrib
- LogFormatter method now return dicts(instead of strings) to support lazy formatting (issue 164, 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 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

7.1.55 Scrapy 0.14.4

- added precise to supported ubuntu distros (commit b7e46df)

- fixed bug in json-rpc webservice reported in https://groups.google.com/forum/#!topic/scrapy-users/qgVBmFybNAQ/discussion. also removed no longer supported 'run' command from extras/scrapy-ws.py (commit 340fbdb)
- meta tag attributes for content-type http equiv can be in any order. #123 (commit 0cb68af)
- replace "import Image" by more standard "from PIL import Image". closes #88 (commit 4d17048)
- return trial status as bin/runtests.sh exit value. #118 (commit b7b2e7f)

7.1.56 Scrapy 0.14.3

- forgot to include pydispatch license. #118 (commit fd85f9c)
- include egg files used by testsuite in source distribution. #118 (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)
- added note to docs/topics/firebug.rst about google directory being shut down (commit 668e352)
- dont discard slot when empty, just save in another dict in order to recycle if needed again. (commit 8e9f607)
- do not fail handling unicode xpaths in libxml2 backed selectors (commit b830e95)
- fixed minor mistake in Request objects documentation (commit bf3c9ee)
- fixed minor defect in link extractors documentation (commit ba14f38)
- removed some obsolete remaining code related to sqlite support in scrapy (commit 0665175)

7.1.57 Scrapy 0.14.2

- move buffer pointing to start of file before computing checksum. refs #92 (commit 6a5bef2)
- Compute image checksum before persisting images. closes #92 (commit 9817df1)
- remove leaking references in cached failures (commit 673a120)
- fixed bug in MemoryUsage extension: get_engine_status() takes exactly 1 argument (0 given) (commit 11133e9)
- fixed struct.error on http compression middleware. closes #87 (commit 1423140)
- ajax crawling wasn't expanding for unicode urls (commit 0de3fb4)
- Catch start_requests iterator errors. refs #83 (commit 454a21d)
- Speed-up libxml2 XPathSelector (commit 2fbd662)

- updated versioning doc according to recent changes (commit 0a070f5)
- scrapyd: fixed documentation link (commit 2b4e4c3)
- extras/makedeb.py: no longer obtaining version from git (commit caffe0e)

7.1.58 Scrapy 0.14.1

- extras/makedeb.py: no longer obtaining version from git (commit caffe0e)
- bumped version to 0.14.1 (commit 6cb9e1c)
- fixed reference to tutorial directory (commit 4b86bd6)
- doc: removed duplicated callback argument from Request.replace() (commit 1aeccdd)
- fixed formatting of scrapyd doc (commit 8bf19e6)
- Dump stacks for all running threads and fix engine status dumped by StackTraceDump extension (commit 14a8e6e)
- added comment about why we disable ssl on boto images upload (commit 5223575)
- SSL handshaking hangs when doing too many parallel connections to S3 (commit 63d583d)
- change tutorial to follow changes on dmoz site (commit bcb3198)
- Avoid _disconnectedDeferred AttributeError exception in Twisted>=11.1.0 (commit 98f3f87)
- allow spider to set autothrottle max concurrency (commit 175a4b5)

7.1.59 Scrapy 0.14

New features and settings

- Support for AJAX crawleable urls
- New persistent scheduler that stores requests on disk, allowing to suspend and resume crawls (r2737)
- 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, r2783)
- New `ChunkedTransferMiddleware` (enabled by default) to support chunked transfer encoding (r2769)
- Add boto 2.0 support for S3 downloader handler (r2763)
- Added marshal to formats supported by feed exports (r2744)

- In request errbacks, offending requests are now received in `failure.request` attribute (r2738)
- Big downloader refactoring to support per domain/ip concurrency limits (r2732)
  - `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)
- Moved Amazon AWS-related components/extensions (SQS spider queue, SimpleDB stats collector) to a separate project: [scaws](https://github.com/scrapinghub/scaws) (r2706, r2714)
- Moved spider queues to scrapyd: `scrapy.spiderqueue` -> `scrapyd.spiderqueue` (r2708)
- Moved sqlite utils to scrapyd: `scrapy.utils.sqlite` -> `scrapyd.sqlite` (r2781)
- Real support for returning iterators on `start_requests()` method. The iterator is now consumed during the crawl when the spider is getting idle (r2704)
- Added *REDIRECT_ENABLED* setting to quickly enable/disable the redirect middleware (r2697)
- Added *RETRY_ENABLED* setting to quickly enable/disable the retry middleware (r2694)
- Added `CloseSpider` exception to manually close spiders (r2691)
- Improved encoding detection by adding support for HTML5 meta charset declaration (r2690)
- Refactored close spider behavior to wait for all downloads to finish and be processed by spiders, before closing the spider (r2688)
- Added `SitemapSpider` (see documentation in Spiders page) (r2658)
- Added `LogStats` extension for periodically logging basic stats (like crawled pages and scraped items) (r2657)
- Make handling of gzipped responses more robust (#319, r2643). 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)
- Added new command to edit spiders: `scrapy edit` (r2636) and `-e` flag to `genspider` command that uses it (r2653)
- Changed default representation of items to pretty-printed dicts. (r2631). This improves default logging by making log more readable in the default case, for both Scraped and Dropped lines.
- Added *spider_error* signal (r2628)
- Added *COOKIES_ENABLED* setting (r2625)

- 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)
- Added new DBM HTTP cache storage backend (r2576)
- Added `listjobs.json` API to Scrapyd (r2571)
- `CsvItemExporter`: added `join_multivalued` parameter (r2578)
- Added namespace support to `xmliter_lxml` (r2552)
- Improved cookies middleware by making `COOKIES_DEBUG` nicer and documenting it (r2579)
- 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: (r263
  - 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 (several functions from `scrapy.utils.{http,markup,multipart,response,url}`, done in r2584)
  - scrapely (was `scrapy.contrib.ibl`, done in r2586)
- Removed unused function: `scrapy.utils.request.request_info()` (r2577)
- 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)
- Removed experimental crawlspider v2 (r2632)
- 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)
- Removed support for passing urls to `scrapy crawl` command (use `scrapy parse` instead) (r2704)

- Removed deprecated Execution Queue (r2704)
- Removed (undocumented) spider context extension (from scrapy.contrib.spidercontext) (r2780)
- removed `CONCURRENT_SPIDERS` setting (use scrapyd maxproc instead) (r2789)
- Renamed attributes of core components: downloader.sites -> downloader.slots, scraper.sites -> scraper.slots (r2717, r2718)
- Renamed setting `CLOSESPIDER_ITEMPASSED` to *CLOSESPIDER_ITEMCOUNT* (r2655). Backward compatibility kept.

7.1.60 Scrapy 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)

7.1.61 Scrapy 0.10

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features and improvements

- Scrapy クローラーを実稼働環境に展開するための `scrapyd` と呼ばれる新しい Scrapy サービス (#218)(ドキュメントあり)
- 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)`
  - backward compatibility kept (with deprecation warning)
- moved `scrapy.core.signals` module to `scrapy.signals`
  - backward compatibility kept (with deprecation warning)
- moved `scrapy.core.exceptions` module to `scrapy.exceptions`
  - backward 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)

7.1.62 Scrapy 0.9

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

New features and improvements

- scrapy.mail に SMTP-AUTH サポートを追加。
- New settings added: `MAIL_USER`, `MAIL_PASS` (r2065 | #149)
- Added new scrapy-ctl view command - To view URL in the browser, as seen by Scrapy (r2039)
- Added web service for controlling Scrapy process (this also deprecates the web console. (r2053 | #167)
- Support for running Scrapy as a service, for production systems (r1988, r2054, r2055, r2056, r2057 | #168)
- Added wrapper induction library (documentation only available in source code for now). (r2011)
- Simplified and improved response encoding support (r1961, r1969)
- Added `LOG_ENCODING` setting (r1956, documentation available)
- Added `RANDOMIZE_DOWNLOAD_DELAY` setting (enabled by default) (r1923, doc available)
- `MailSender` is no longer IO-blocking (r1955 | #146)
- Linkextractors and new Crawlspider now handle relative base tag urls (r1960 | #148)
- Several improvements to Item Loaders and processors (r2022, r2023, r2024, r2025, r2026, r2027, r2028, r2029, r2030)
- Added support for adding variables to telnet console (r2047 | #165)
- Support for requests without callbacks (r2050 | #166)

API changes

- Change `Spider.domain_name` to `Spider.name` (SEP-012, r1975)
- `Response.encoding` is now the detected encoding (r1961)
- `HttpErrorMiddleware` now returns None or raises an exception (r2006 | #157)
- `scrapy.command` modules relocation (r2035, r2036, r2037)
- Added `ExecutionQueue` for feeding spiders to scrape (r2034)
- Removed `ExecutionEngine` singleton (r2039)
- Ported `S3ImagesStore` (images pipeline) to use boto and threads (r2033)

- Moved module: `scrapy.management.telnet` to `scrapy.telnet` (r2047)

Changes to default settings

- Changed default `SCHEDULER_ORDER` to `DFO` (r1939)

7.1.63 Scrapy 0.8

The numbers like #NNN reference tickets in the old issue tracker (Trac) which is no longer available.

新機能

- Added DEFAULT_RESPONSE_ENCODING setting (r1809)
- Added `dont_click` argument to `FormRequest.from_response()` method (r1813, r1816)
- Added `clickdata` argument to `FormRequest.from_response()` method (r1802, r1803)
- Added support for HTTP proxies (`HttpProxyMiddleware`) (r1781, r1785)
- Offsite spider middleware now logs messages when filtering out requests (r1841)

後方互換性のない変更

- Changed `scrapy.utils.response.get_meta_refresh()` signature (r1804)
- Removed deprecated `scrapy.item.ScrapedItem` class - use `scrapy.item.Item instead` (r1838)
- Removed deprecated `scrapy.xpath` module - use `scrapy.selector` instead. (r1836)
- Removed deprecated `core.signals.domain_open` signal - use `core.signals.domain_opened` instead (r1822)
- `log.msg()` now receives a `spider` argument (r1822)
  - 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 | #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 | #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)
  - 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)
- Renamed setting: `REQUESTS_PER_DOMAIN` to `CONCURRENT_REQUESTS_PER_SPIDER` (r1830, r1844)
- Renamed setting: `CONCURRENT_DOMAINS` to `CONCURRENT_SPIDERS` (r1830)
- Refactored HTTP Cache middleware
- HTTP Cache middleware has been heavilty refactored, retaining the same functionality except for the domain sectorization which was removed. (r1843)
- Renamed exception: `DontCloseDomain` to `DontCloseSpider` (r1859 | #120)
- Renamed extension: `DelayedCloseDomain` to `SpiderCloseDelay` (r1861 | #121)
- Removed obsolete `scrapy.utils.markup.remove_escape_chars` function - use `scrapy.utils.markup.replace_escape_chars` instead (r1865)

7.1.64 Scrapy 0.7

Scrapy の最初のリリース。

7.2 Scrapy への貢献

重要: https://docs.scrapy.org/en/master/contributing.html で、この文書の最新バージョンを読んでいることを再確認してください

Scrapy に貢献する方法はたくさんあります。 それらのいくつかを次に示します:

- Scrapy についてのブログ。Scrapy の使用方法を世界中に伝えましょう。これは、より多くの例により新規参入者を助け、Scrapy プロジェクトの可視性を高めるのに役立ちます。
- 以下の「バグの報告」(*Reporting bugs*) で詳述されているガイドラインに従うようにして、バグを報告し、issue tracker で機能をリクエストします。
- 新しい機能やバグ修正のためのパッチを上げてください。パッチの作成方法と送信方法の詳細については、下記の **パッチを書く** と 「パッチの送信」(*Submitting patch*) をお読みください。
- Scrapy subreddit に参加して、Scrapy の改善方法に関するアイデアを共有してください。私たちは常に提案を受け入れています。
- Stack Overflow で Scrapy の質問に答えてください。

7.2.1 バグの報告

注釈: セキュリティの問題は **scrapy-security@googlegroups.com にのみ** 報告してください。これは信頼できる Scrapy 開発者のみが利用できるプライベートメーリングリストであり、そのアーカイブは公開されていません。

よく書かれたバグレポートは非常に役立ちます。よって、新しいバグを報告するときは以下のガイドラインに留意してください。

- 最初に *FAQ(よくある質問と回答)* をチェックして、よく知られている質問で問題が解決されていないかどうかを確認してください
- あなたが Scrapy の使用に関する一般的な質問がある場合は、 Stack Overflow (scrapy タグ付け) で質問してください。
- open issues をチェックして、問題がすでに報告されているかどうかを確認します。報告されている場合は、そのレポートを閉じないで、チケットの履歴とコメントを確認してください。追加の有用な情報がある場合は、コメントを残すか、修正を含めて **プル・リクエスト** を送信することを検討してください。
- scrapy-users リストと Scrapy subreddit を検索して、そこで議論されているか、表示されているものがバグかどうかわからないかどうかを確認します。 `#scrapy` IRC チャンネルで質問することもできます。

- **完全で再現可能な特定のバグレポート** を作成します。テストケースが小さければ小さいほど良いです。他の開発者にはバグを再現するプロジェクトがないため、再現に必要なすべての関連ファイルを含めてください。たとえば、問題を示す「最小限の、完全な、検証可能な例」を作成する StackOverflow のガイド (Minimal, Complete, and Verifiable example) を参照してください。
- 完全で再現可能な例を提供する最も素晴らしい方法は、Scrapy テスト・スイートに失敗したテスト・ケースを追加するプル・リクエストを送信することです (**パッチの送信** 参照)。これは、自分で問題を解決する意図がない場合でも役立ちます。
- `scrapy version -v` の出力を含めると、バグに取り組んでいる開発者は、それが発生したバージョンとプラットフォームを正確に知ることができます。これは、バグを再現したり、すでに修正されているかどうかを知るのに非常に役立ちます。

7.2.2 パッチを書く

パッチが適切に作成されるほど、パッチが受け入れられる可能性が高くなり、マージが早くなります。

適切なパッチは以下のようにすべきです:

- 特定の変更に必要な最小限のコードが含まれています。小さなパッチは、確認とマージが簡単です。したがって、複数の変更 (またはバグ修正) を行っている場合は、変更ごとに 1 つのパッチを提出することを検討してください。複数の変更を 1 つのパッチにまとめないでください。大きな変更については、パッチ・キュー (patch queue) の使用を検討してください。
- すべての単体テストに合格させます。以下の「テストの実行」(*Running tests*) を参照してください。
- 修正されたバグまたは追加された新しい機能をチェックする 1 つ (または複数) のテスト・ケースを含めます。以下の「テストの作成」(*Writing tests*) を参照してください。
- パブリック (文書化された)API を追加または変更する場合は、同じパッチに文書ントの変更を含めてください。以下の「文書ポリシー」(*Documentation policies*) を参照してください。
- プライベート API を追加する場合は、`docs/conf.py` の ``coverage_ignore_pyobjects` 変数に正規表現を追加して、ドキュメント・カバレッジ・チェックから新しいプライベート API を除外してください。

  あなたのプライベート API が適切にスキップされているかどうかを確認するには、次のようにドキュメント・カバレッジ・レポートを生成します:

```
tox -e docs-coverage
```

7.2.3 パッチの送信

パッチを送信する最良の方法は、GitHub でプルリクエスト (pull request) を発行することです。オプションで最初に新しい問題を作成します。

修正された機能または新しい機能 (それが何であるか、なぜ必要なのかなど) を忘れずに説明してください。含める情報が多いほど、コア開発者がパッチを理解し、受け入れやすくなります。

パッチを作成する前に新しい機能 (またはバグ修正) について議論することもできますが、引数を説明し、主題にいくつかの追加の考えを入れたことを示す準備ができているパッチを用意しておくことは常に良いことです。適切な出発点は、GitHub でプル・リクエストを送信することです。それはあなたのアイデアを説明するのに十分単純であり、アイデアが検証され有用であることが証明された後、ドキュメント/テストを後で残すことができます。または、 Scrapy subreddit で会話を開始して、最初にアイデアについて話し合うこともできます。

しばしば、解決したい問題に対する既存のプル・リクエストが存在することがありますが、何らかの理由で停止します。多くの場合、プル・リクエストは正しい方向にありますが、変更は Scrapy メンテナによってリクエストされ、元のプル・リクエストの作成者にはそれらに対処する時間がありませんでした。この場合、このプルリ・クエストを選択することを検討してください。元のプル・リクエストからのすべてのコミットと、発生した問題に対処するための追加の変更を含む新しいプル・リクエストを開きます。そうすることは非常に役立ちます。元の作者のコミットが認識されるやいなや、彼/彼女のそのコミットを取り込むことは失礼とはみなされません。

既存のプルリクエストをローカル・ブランチにプルするには、 `git fetch upstream pull/$PR_NUMBER/head:$BRANCH_NAME_TO_CREATE` を実行します (「upstream」を scrapy リポジトリのリモート名に、「$ PR_NUMBER」をプル・リクエストの ID、びローカルに作成するブランチの名前を含む `$BRANCH_NAME_TO_CREATE`)。 https://help.github.com/articles/checking-out-pull-requests-locally/#modifying-an-inactive-pull-request-locally も参照してください。

GitHub プル・リクエストを作成するときは、タイトルを短く、しかし説明的なものにしてください。例えば、バグ #411:"Scrapy hangs if an exception raises in start_requests" の場合 "Fix for #411" より "Fix hanging when exception occurs in start_requests (#411)" が好ましいです。完全なタイトルを使用すると、issue tracker を簡単に確認できます。

最後に、機能的な変更とは別のコミットで、審美的な変更 (**PEP 8** 対応、未使用のインポートの削除など) を維持するようにしてください。これにより、プル・リクエストの確認が容易になり、マージされる可能性が高くなります。

7.2.4 コーディング・スタイル

Scrapy に含めるコードを記述するときは、これらのコーディング規則に従ってください:

- 特に指定がない限り、 **PEP 8** に従ってください。
- コードの可読性を向上させるためであれば、80 文字より長い行を使用しても構いません。

- 貢献するコードにあなたの名前を入れないでください。git は、コードの作成者を識別するのに十分なメタ・データを提供します。セットアップ手順については、 https://help.github.com/articles/setting-your-username-in-git/ を参照してください。

7.2.5 文書ポリシー

API メンバー (クラス、メソッドなど) のリファレンス・ドキュメントについては、docstrings を使用し、Sphinx ドキュメントが autodoc 拡張を使用してドキュメント文字列をプルすることを確認してください。 API リファレンス・ドキュメントは docstring の規則 (PEP 257) に準拠し、IDE フレンドリである必要があります。それは短くて要点を示して短い例を提供します。

チュートリアルやトピックなどの他の種類のドキュメントは、 `docs/` ディレクトリ内のファイルでカバーする必要があります。これには、API メンバーに固有のドキュメントが含まれますが、それは API リファレンス・ドキュメントを超える解説内容のものです。

いずれにせよ、docstring で何かが網羅されている場合、 `docs/` ディレクトリ内のファイルに docstring を複製するのではなく、autodoc 拡張を使用して docstring をドキュメントに取り込みます。

7.2.6 テスト

テストは Twisted unit-testing framework を使用して実装され、テストの実行には tox が必要です。

テストを実行する

最新の十分な tox インストールがあることを確認してください:

```
tox --version
```

バージョンが 1.7.0 より古い場合は、最初に更新してください:

```
pip install -U tox
```

すべてのテストを実行するには、Scrapy ソースコードのルートディレクトリに移動して実行します:

```
tox
```

特定のテスト (たとえば `tests/test_loader.py`) を実行するには、次を使用します:

```
tox -- tests/test_loader.py
```

特定の tox 環境でテストを実行するには、 `tox.ini` からの環境名で `-e <name>` を使用します。たとえば、Python 3.6 でテストを実行するときは次の通りです:

```
tox -e py36
```

環境のコンマ区切りリストを指定し、 tox ' s parallel mode を使用して、複数の環境でテストを並行して実行することもできます:

```
tox -e py27,py36 -p auto
```

コマンドライン・オプションを pytest に渡すには、tox の呼び出しで -- の後に追加します。 -- を使用すると、 tox.ini で定義されているデフォルトの位置引数がオーバーライドされるため、これらのデフォルトの位置引数 (scrapy tests) も -- の後に含める必要があります:

```
tox -- scrapy tests -x  # stop after first failure
```

pytest-xdist プラグインを使用することもできます。たとえば、すべての CPU コアを使用して Python 3.6 tox 環境ですべてのテストを実行するには次のようにします:

```
tox -e py36 -- scrapy tests -n auto
```

カバレッジ・レポートを表示するには、coverage をインストール (pip install coverage) して実行します:

```
coverage report
```

html または xml レポートなどのオプションについては、 coverage --help の出力を参照してください。

テストを書く

すべての機能 (新機能やバグ修正を含む) に期待どおりに動作することを確認するテストケースを含める必要があります。パッチをより早く受け入れてもらいたい場合は、パッチのテストを含めてください。

Scrapy はユニット・テストを使用します。ユニット・テストは tests/ ディレクトリにあります。それらのモジュール名は通常、テストしているモジュールの完全なパスに似ています。たとえば、アイテム・ローダーのコードは次の通りです:

```
scrapy.loader
```

そして、それらのユニットテストは以下です:

```
tests/test_loader.py
```

7.3 バージョン管理と API の安定性

7.3.1 バージョン管理

Scrapy のバージョンは 3 つの数字から成ります。 *A.B.C*

- *A* はメジャーバージョンです。これはめったに変更されず、非常に大きな変更を示します。
- *B* はリリース番号です。 これには、下位互換性を損なう可能性がある、機能や事柄を含む多くの変更が含まれますが、これらのケースを最小限に抑えるよう努めています。
- *C* はバグ修正リリース番号です。

後方非互換性は **リリースノート** で明示的に言及してあります。アップグレードする前に特別な注意が必要な場合があります。

開発リリースは 3 番号のバージョンには従わず、通常、接尾辞付きの `dev` バージョンとしてリリースされます。例： `1.3dev`

注釈: Scrapy 0.* シリーズでは、Scrapy は奇数番号のバージョンは開発リリース (odd-numbered versions for development releases) でしたた。これは Scrapy 1.0 以降には適用されません。

Scrapy 1.0 からは、すべてのリリースは本番環境対応と見なすべきです。

例えば:

- (製品で使っても安全な) *1.1* シリーズの *1.1.1* は最初のバグ修正リリース

7.3.2 API の安定性

API の安定性は、 *1.0* リリースの主要な目標の 1 つでした。

単一のダッシュ (_) で始まるメソッドまたは関数はプライベートであり、安定版として信頼してはいけません。

また、安定版は完全を意味するものではないことに注意してください。安定版 API は新しいメソッドや機能を拡張できますが、既存のメソッドは依然として同じように機能し続ける必要があります。

リリース・ノート 直近の Scrapy バージョンの変更点をご覧ください。

***Scrapy* への貢献** Scrapy プロジェクトに貢献する方法を学びます。

バージョン管理と *API* の安定性 Scrapy のバージョン管理と API の安定性を理解します。

Python モジュール索引

S

索引