第 2 章 Pythonではじめるクローリング・スクレイピング

ここからはPythonを使ってクローリング・スクレイピングを行います。Pythonの開発環境構築と基本文法から、クローリング・スクレイピングの一連の過程をスクリプトにするまでを解説します。

2.1 Pythonを使うメリット

クローリング・スクレイピングにPythonを使うメリットとして、**第1章**のまとめで述べたように汎用的なプログラミング言語であることを挙げました。さらに次のメリットがあります。

- 言語自体の特性
- 強力なライブラリ
- スクレイピング後の処理との親和性

2.1.1 言語自体の特性

1点目のメリットはPythonの言語そのものの特性です。

Pythonは教育用に使われることもある読みやすく書きやすい言語です。一度書かれたプログラムは、その後何度も他の人（未来の自分を含む）に読まれることになるため、読みやすさは重要です。

PythonはBattery Included（電池付属）と言われています。電池つき電化製品のように、豊富な標準ライブラリが付属しており、インストール後すぐに使いはじめられることの比喩です。ここでは、Pythonの標準ライブラリだけでクローリング・スクレイピングを行います。後の章でより強力なサードパーティライブラリも活用しますが、手軽に使いはじめられるPythonの良さがわかるはずです。

さらに、複数のWebサイトから高速にデータを取得するためには、非同期処理が有効です。PythonにはTwistedやTornadoなどの非同期処理のためのフレームワークが存在し、Python 3.4からはasyncioと呼ばれる非同期処理のための標準ライブラリもあります。非同期処理の分野においてはNode.jsやGo言語が有名ですが、Pythonでも手軽に扱えます。

2.1.2 強力なサードパーティライブラリの存在

2点目は豊富なサードパーティライブラリの存在です。Python Package Index（PyPI）*1には、世界中の開発者が数多くのライブラリを公開しており、簡単に使うことができます。特に、**第3章**で紹介するBeautiful SoupやlxmlはPythonの有名なスクレイピングライブラリですし、**第6章**で紹介するScrapyは強力なクローリング・スクレイピングフレームワークです。このような強力なライブラリによって巨人の肩の上に乗り、短いプログラムを書くだけで素早くクローリング・スクレイピングを行えます。

2.1.3 スクレイピング後の処理との親和性

3点目はクローリング・スクレイピングでデータを取得した後、データ分析などの処理を行う際にもPythonが強力な武器になる点です。データ分析においてもPythonには優秀なライブラリが揃っているため、1つの言語を修得するだけで大抵のことは実現できてしまいます。

Pythonでは数値計算や科学技術計算の分野で古くからNumPyやSciPyといったライブラリが有名で、これらをベースとしたデータ分析用のライブラリが存在します。例えばpandas（**5.3.2**参照）は、NumPyをベースとしてデータの前処理（欠損値や表記ゆれの処理）や集計を簡単に行えるライブラリです。またmatplotlib（**5.3.3**参照）は数値データをグラフで可視化できます。データ分析の分野ではR言語が有名ですが、これらのライブラリでPythonでも同様の分析が行えます。他にも様々なライブラリがあります。

2.2 Pythonのインストールと実行

Python 3をインストールし、仮想環境内で実行します。

2.2.1 Python 2とPython 3

Pythonには大きく分けて、**Python 2**系と**Python 3**系の2つのバージョンがあります。Pythonは後方互換性を重視する言語ですが、2008年にリリースされたバージョン3.0において後方互換性を崩す大きな変更が入りました。そのためライブラリの3系への対応が進まず、2系も並行して開発されてきた経緯があります。

近年では多くのライブラリが3系に対応し、Python 3を問題なく使える状況になってきています。3

*1 https://pypi.python.org/pypi

系は2系に比べて、よりわかりやすい文法への変更、Unicodeサポートの強化、標準ライブラリの整理、非同期I/Oのサポートなど様々な改善が行われています。2系のサポートは2020年で打ち切られる予定になっていることもあり、今から使いはじめるのであれば3系を使うのがオススメです。本書ではPython 3のみを使用し、Python 3.5.1（OS X）とPython 3.4.3（Ubuntu）を対象として解説します。

2.2.2 パッケージマネージャーによるPython 3のインストール

OS XではHomebrew（**1.2.1**参照）で、UbuntuではAPTでインストールします*2*3*4。

```
$ brew install python3  # OS Xの場合
```

```
$ sudo apt-get install -y python3 python3.4-venv  # Ubuntuの場合
```

インストールに成功すると、python3コマンドでバージョンを確認できます。

```
$ python3 -V
Python 3.5.1
```

2.2.3 仮想環境（venv）の使用

近年のプログラミング言語では**仮想環境（Virtual Environment）**と呼ばれる、ランタイムやライブラリを環境（用途）ごとに分離できる仕組みが多く使われています。

● 仮想環境とは

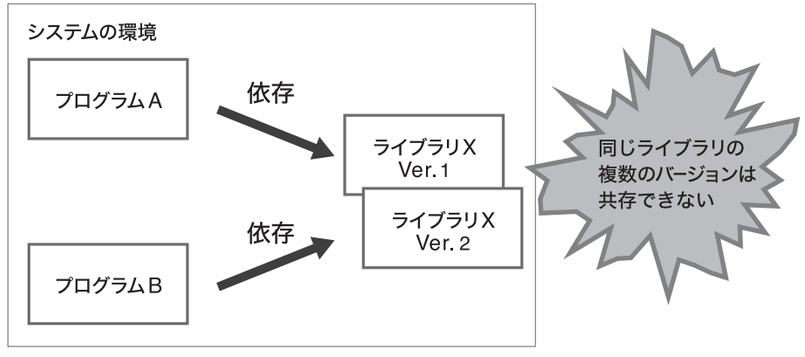
例えば、1台のコンピューターで2つの異なるプログラムAとBを書いているとしましょう。2つのプログラムはXというライブラリに依存しており、プログラムAではライブラリXのバージョン1に、プログラムBではライブラリXのバージョン2に依存しているとします。Pythonでは1つの環境に同じライブラリの複数バージョンをインストールできないため、インストールできるのはXのバージョン1か2のいずれかのみです。ライブラリXのバージョン1と2に互換性がない場合、プログラムAとBのどちらかは正常に動作しなくなってしまいます（**図2.1**）。

*2 本書で使用するプラットフォームでは、Python 2系と3系はコマンド名やライブラリのインストール場所が異なるので共存できます。3系をインストールしても、既存の2系が使えなくなることはありません。

*3 Pythonをインストールするためのツールとして、pyenv（https://github.com/yyuu/pyenv）やpythonz（https://github.com/saghul/pythonz）などが存在します。しかしPythonはメジャーバージョンが同じであれば基本的に後方互換があるので、マイナーバージョンが異なるPythonを使い分けたいといったこだわりがない限り、利用する必要はないでしょう。

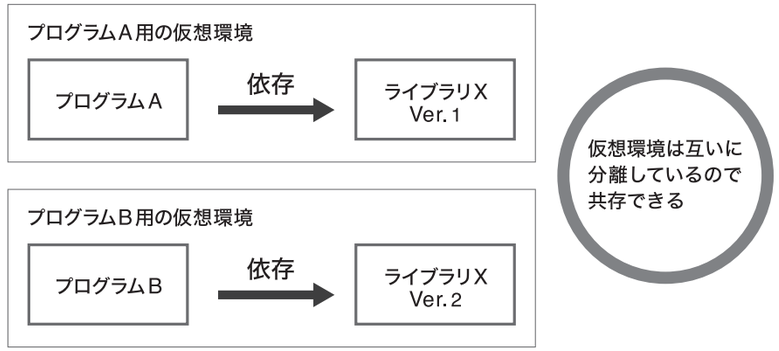
*4 仮想環境を利用するために、Ubuntuではvenvパッケージをインストールする必要があります。本書の解説には用いていませんがUbuntu 16.04では`python3.4-venv`パッケージの代わりに`python3-venv`パッケージをインストールします。

▼ 図2.1 仮想環境がない時



仮想環境を使うと、このような事態を避けられます。プログラムA用の仮想環境にXのバージョン1をインストールし、プログラムB用の仮想環境にXのバージョン2をインストールすることで、互いに干渉することなくプログラムを開発できます（**図2.2**）。

▼ 図2.2 仮想環境がある時



Pythonでは**venv**という標準モジュールで仮想環境を利用できます*5。

仮想環境内ではpython3コマンドのようにバージョン番号がついたコマンドを使わなくても、pythonコマンドで仮想環境に紐付けられたPythonランタイムを起動できるメリットもあります。また、仮想

*5 Pythonでは仮想環境を使うためにvirtualenv（https://virtualenv.pypa.io/en/latest/）というサードパーティのツールが広く使われてきました。Python 3.3以降はvenvを使えばvirtualenvを使う必要はありません。

環境は通常の開発で広く使われているため、覚えておいて損はないでしょう。

● 仮想環境の使い方

次のコマンドで、仮想環境を作成します*6*7。-mオプションは指定したモジュールをスクリプトとして実行することを意味します。

```
$ python3 -m venv scraping
```

カレントディレクトリにscrapingディレクトリが作成されます。このディレクトリの名前が仮想環境の名前になります。ディレクトリの名前は自由に設定できます*8。

```
$ ls scraping/
bin        include    lib        pyvenv.cfg
```

仮想環境に入るためには、.（ドット）コマンドでactivateスクリプトを実行します。

```
$ . scraping/bin/activate  # ドットと引数の間にスペースを入れる。
```

.コマンドは引数で指定したファイルを読み込み、現在のシェルで実行するコマンドです。BashやZshなどではsourceコマンドも同じ意味を持ちます。

仮想環境に入ると、シェルのプロンプトの先頭に(scraping)と表示されるようになります。

```
(scraping) $
```

仮想環境内ではpythonコマンドでPython 3が実行できます。

```
(scraping) $ python -V
Python 3.5.1
```

whichコマンドでpythonコマンドのパスを確認すると、scraping/binディレクトリ内のpythonコマンドを指していることがわかります。

```
(scraping) $ which python
/path/to/scraping/bin/python
```

*6 WindowsのVirtualBox上で仮想マシンを実行している場合、共有フォルダとしてマウントされているディレクトリ（Appendixの手順のデフォルトでは/vagrant/）内に仮想環境を作成するとシンボリックリンクの作成に失敗します。仮想環境はホームディレクトリ（~）など、共有フォルダ以外のディレクトリに作成してください。

*7 venvモジュール導入当初は、pyvenvというコマンドで仮想環境を作成していましたが、このコマンドはPython 3.6以降ではDeprecatedになる予定です。

*8 仮想環境を作成した後にディレクトリをリネームしたり移動したりすると、正常に動作しなくなるので注意してください。

deactivateコマンドで仮想環境から抜けることができます。仮想環境から抜けるとシェルのプロンプトから(scraping)の文字がなくなり、元に戻ります。

```
(scraping) $ deactivate
```

仮想環境自体が不要になった場合はディレクトリをまるごと削除します。

以降の説明では、特に断りのない限り仮想環境内でコマンドを実行します。書かれている通りに実行しても結果が異なったり予期せぬエラーが発生する場合は、まず仮想環境内でPython 3を使用しているか確認してください。

2.2.4 インタラクティブシェルの使用

pythonコマンドを引数なしで実行すると、インタラクティブシェルが起動します。Pythonのコードを対話的に実行できるので、ライブラリの使い方の確認などに便利です。

```
(scraping) $ python
Python 3.5.1 (default, Apr 18 2016, 03:49:24)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1～3行目にはバージョンなどの情報と簡単な説明が表示され、4行目に>>>が表示されて入力を受け付ける状態になります。この>>>をプロンプトと呼びます。プロンプトのある行にPythonの式を入力してEnterを押すと、次の行にその式の値が表示されます。

```
>>> 1 + 1
2
```

プロンプトが表示されている時にCtrl＋Dを押すか、exit()と入力してEnterを押すとインタラクティブシェルを終了できます。

入力中の場合は、Ctrl＋Cを押すと入力中の文字列がリセットされて、プロンプトが表示されます。通常のシェルと同じように、上下の矢印キーで以前の入力を表示したり、Ctrl＋Rで以前の入力からインクリメンタルサーチすることも可能です。

Pythonのオンラインマニュアルでもインタラクティブシェル形式の解説が多くあります。新しい機能を使うときなどはインタラクティブシェルで試す習慣をつけると良いでしょう。

2.3 Pythonの基礎知識

本書を読み進め、実際にクローリング・スクレイピングする上で必要な、Pythonの文法をはじめとした基礎知識を解説します。最初は流し読みして、後で不明点があったときに戻ってきても良いでしょう。

2.3.1 スクリプトファイルの実行と構成

Pythonスクリプトファイルの実行方法と構成を解説します。

● Pythonスクリプトファイルの実行

Pythonのスクリプトは.pyという拡張子のファイルに保存します。テキストエディターを使って、**リスト2.1**の中身を持つファイルをgreet.pyという名前で作成してください。ファイルのエンコーディングはUTF-8を使います。

▼ **リスト2.1　greet.py — Pythonスクリプトの例**

```
import sys


def greet(name):
    print('Hello, {0}!'.format(name))

if len(sys.argv) > 1:
    name = sys.argv[1]
    greet(name)
else:
    greet('world')
```

Pythonのスクリプトファイルを実行するには、ファイルパスをpythonコマンドの引数に渡します。ファイルを保存したら、仮想環境内で次のコマンドを実行して、「Hello, world!」と表示されることを確認します。

```
(scraping) $ python greet.py
Hello, world!
```

この例では引数に文字列を与えて実行すると、表示が変わります。

```
(scraping) $ python greet.py Guido
Hello, Guido!
```


● Pythonスクリプトの構成

Pythonを使ったことがない方でも、他の言語の経験があればある程度**リスト2.1**のコードの意味するところが理解できるのではないでしょうか。このコードにコメントをつけると**リスト2.2**のようになります。行の#以降はコメントです。

▼ **リスト2.2　greet_with_comments.py — Pythonスクリプトの例（コメントつき）**

```
import sys  # import文でsysモジュールを読み込む。

# def文でgreet()関数を定義する。インデントされている行が関数の中身を表す。
def greet(name):
    print('Hello, {0}!'.format(name))  # 組み込み関数print()は文字列を出力する。

# if文でもインデントが範囲を表す。sys.argvはコマンドライン引数のリストを表す変数。
if len(sys.argv) > 1:
    # if文の条件が真のとき
    name = sys.argv[1]  # 変数は定義せずに代入できる。
    greet(name)  # greet()関数を呼び出す。
else:
    # if文の条件が偽のとき
    greet('world')  # greet()関数を呼び出す。
```

Pythonのスクリプトは上から順に実行されます。関数は事前に定義されている必要があります。基本的に1行に1文だけを書き、行末にセミコロンなどの記号は不要です。

Pythonでは**インデント**が大きな意味を持ちます。次のようにブロックをインデントで表します。

```
if a == 1:
    print('a is 1')
else:
    print('a is not 1')
```

C言語やJavaScriptでは、if文などでブロックを表すときに{}（波括弧）を使い、多くの場合は読みやすいようにブロック内をインデントします。

```
if (a == 1) {
    printf("a is 1\n");
} else {
    printf("a is not 1\n");
}
```

しかし、ブロック内をインデントしなくても意味は変わりません。

```
if (a == 1) {
printf("a is 1\n");
} else {
printf("a is not 1\n");
}
```

これでは、プログラムが非常に読みにくくなってしまいます。Pythonではこのように読みにくくなることを避けるため、正しくインデントされていないブロックがあるとIndentationErrorというエラーになり実行できません。構文として正しいインデントを強制することで、誰が書いても読みやすいプログラムになるのです。インデントでブロックを表す言語を使うのがはじめての場合は気になるかもしれませんが、慣れれば自然と書けるようになるので安心してください。

Pythonでのインデントは一般的にスペース4つを使います。多くのテキストエディターでは、ソフトタブと呼ばれる設定を有効にすることで、Tabキーを押したときにスペース4つを挿入できます。なお、インデントを増やす直前の行の末尾には:(コロン)を置くことを忘れないでください。

2.3.2 基本的なデータ構造

Pythonでは数値、文字列、リスト、辞書などの基本的なデータ構造を手軽に扱えます。

● 数値

整数と実数の基本的な四則演算が行えます。pythonコマンドでインタラクティブシェルを起動して、次と同じように打ち込んでいくとわかりやすいでしょう。

```
>>> type(1)  # 整数はintクラス。type()関数でクラスを確認できる。
<class 'int'>
>>> type(1.0)  # 実数はfloatクラス。
<class 'float'>
>>> 1 + 2
3
>>> 2 - 1
1
>>> 2 * 3
6
>>> 5 / 2  # / 演算子による除算結果は実数になる。
2.5
>>> 5 // 2  # // 演算子による除算結果は切り捨てられた整数になる。
2
>>> 5 % 2  # % 演算子で除算の余りを取得する。
1
>>> 1 + 2 * 3  # 演算子には優先順位があり、* のほうが + よりも優先される。
7
```



```
>>> (1 + 2) * 3  # () の中身のほうが優先順位が高い。
9
```

● 文字列

Unicode文字列を表すstrクラスと、バイト列を表すbytesクラスがあります。文字列操作は基本的にstrクラスで行い、ファイルやネットワーク越しのデータの読み書きなど、Python以外との境界でbytesクラスに変換します。

```
>>> type('abc')  # 文字列はstrクラス。
<class 'str'>
>>> 'abc'  # 文字列は'または"で囲う。'...'内では"が、"..."内では'がエスケープせずに使える点を除いて違いはない。
'abc'
>>> 'あいうえお'
'あいうえお'
>>> "1970's"  # 'を含む文字列は"で囲うとわかりやすい。
"1970's"
>>> 'abc\n123'  # \nは改行文字を表す。他にも\t(タブ文字)などのエスケープシーケンスがある。
'abc\n123'
>>> print('abc\n123')  # print()関数を使うと、クオートやエスケープなしにそのまま表示される。
abc
123
```

```
>>> len('abc')  # 組み込み関数len()で文字列の長さを取得する。
3
>>> len('あいうえお')  # 文字列の長さはUnicode文字単位で数える。
5
>>> 'abcdef'[0]  # [0]で0番目の文字を表す1文字の文字列を得る。
'a'
>>> 'abcdef'[-1]  # [-1]のように負のインデックスを指定すると、後ろから数える。
'f'
# 開始インデックスと終了インデックスを指定して範囲の部分文字列を得る。これをスライスと呼ぶ。
>>> 'abcdef'[1:3]
'bc'
>>> 'abcdef'[:3]  # 開始インデックスを省略すると、先頭から終了インデックスまでの部分文字列を得る。
'abc'
>>> 'abcdef'[1:]  # 終了インデックスを省略すると、開始インデックスから末尾までの部分文字列を得る。
'bcdef'
>>> 'abc' + 'def'  # +演算子で文字列同士を結合する。
'abcdef'
# strクラスのformat()メソッドで{}の部分を引数の値に置き換えた文字列を取得する。
>>> '{0}, Python {1}!'.format('Hello', 3)  # {}内の番号は、引数のインデックスを表す。
'Hello, Python 3!'
```

```
# strクラスのencode()メソッドでbytesオブジェクトに変換。第1引数でエンコーディングの名前を指定。
>>> 'ABCあいう'.encode('utf-8')
b'ABC\xe3\x81\x82\xe3\x81\x84\xe3\x81\x86'
```

第 6 章 フレームワーク Scrapy

ここまでは、個々のライブラリを組み合わせてクローリング・スクレイピングを行ってきました。様々なWebサイトを対象にクローラーのプログラムを書いていると、同じような処理を繰り返し書いていることに気づくかもしれません。この手間を省くために使えるのが、クローリング・スクレイピングのためのフレームワーク、Scrapyです。Scrapyを使うと、どんなWebサイトでも使える共通処理をフレームワークに任せて、ユーザーは個々のWebサイトごとに異なる処理だけを書けばよくなります。

短いコードで効率的にクローリング・スクレイピングできるので、様々なサイトからデータを抜き出したい場合や、継続的にクロールを行いたい場合には、学習コストを払うだけの価値があるでしょう。

6.1 Scrapyの概要

Scrapy[*1]はクローリング・スクレイピングのためのPythonのフレームワークです。豊富な機能が備わっており、ユーザーはページからデータを抜き出すという本質的な作業に集中できます。

- Webページからのリンクの抽出
- robots.txtの取得と拒否されているページのクロール防止
- XML Sitemapの取得とリンクの抽出
- ドメインごと／IPアドレスごとのクロール時間間隔の調整
- 複数のクロール先の並行処理
- 重複するURLのクロール防止
- エラー時の回数制限付きのリトライ
- クローラーのデーモン化とジョブの管理

これまでは個々の機能を持つライブラリを自分の書いたプログラムから呼び出して利用してきました。フレームワークであるScrapyでは、流儀にしたがってプログラムを書き、それらをScrapyが呼び出して実行します。新しく流儀を覚える必要はありますが、一度覚えてしまえば面倒な処理をフレームワークに任せられるため、手軽にクローリング・スクレイピングができます。

*1 https://scrapy.org/ 本書ではバージョン1.1.0を使用します。

Scrapyはイベント駆動型のネットワークプログラミングのフレームワークであるTwsited[*2]をベースにしており、Webサイトからのダウンロード処理は非同期に実行されます。このため、ダウンロードを待つ間にもスクレイピングなど別の処理を実行でき、効率よくクローリング・スクレイピングできます。Scrapyのような、スクレイピングのための多機能なフレームワークは他の言語ではあまり見かけず[*3]、これらの用途にPythonを使う大きな理由の1つだと言えます。

Scrapyは長らくPython 3では動作しませんでしたが、2016年5月にリリースされたバージョン1.1からPython 3に対応しました。

6.1.1 Scrapyのインストール

Scrapyをインストールします。UbuntuではOpenSSLの開発用パッケージが必要なので一緒にインストールします。Scrapyはlxmlに依存しているので、libxml2とlibxsltの開発用パッケージも必要です（**3.3.2**参照）[*4]。

```
(scraping) $ sudo apt-get install -y libssl-dev libffi-dev  # Ubuntuの場合
```

```
(scraping) $ pip install scrapy
```

インストールに成功すると`scrapy`コマンドが使えるようになります。

```
(scraping) $ scrapy version
Scrapy 1.1.0
```

6.1.2 Spiderの実行

Scrapyを使うのに、主に作成するのがSpiderというクラスです。対象のWebサイトごとにSpiderを作成し、クローリングの設定や、スクレイピングの処理を記述します。

まずは簡単なSpiderを実行してみましょう。**リスト6.1**は、Scrapyのサイトに掲載されているサンプルコード[*5]にコメントを加えたものです。

*2 https://twistedmatrix.com/

*3 近いものとしてRubyのAnemone（http://anemone.rubyforge.org/）がありますが、Scrapyよりも薄いフレームワークで、残念ながら2012年で開発が止まっています。

*4 ScrapyのようなC拡張ライブラリに依存するライブラリをインストールする場合、依存ライブラリのビルドに失敗してもインストール済みになってしまい、正常に動作しないことがあります。その場合、ビルド失敗の原因を取り除き、一度アンインストールするとインストールし直せます。また、インストール中に問題が発生する場合は`pip install -U pip`でpip自体をアップグレードすると問題が解決することもあります。

*5 https://scrapy.org/のページ中央

▼ リスト6.1　myspider.py — Scrapinghub社のブログから投稿のタイトルを取得するSpider

```
import scrapy


class BlogSpider(scrapy.Spider):
    name = 'blogspider'  # Spiderの名前。
    # クロールを開始するURLのリスト。
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        """
        トップページからカテゴリページへのリンクを抜き出してたどる。
        """
        for url in response.css('ul li a::attr("href")').re('.*/category/.*'):
            yield scrapy.Request(response.urljoin(url), self.parse_titles)

    def parse_titles(self, response):
        """
        カテゴリページからそのカテゴリの投稿のタイトルをすべて抜き出す。
        """
        for post_title in response.css('div.entries > ul > li a::text').extract():
            yield {'title': post_title}
```

このSpiderは、Scrapyをメンテナンスしている Scrapinghub社のブログをクロールします。一覧・詳細パターンのWebサイトを処理するSpiderで、ブログのトップページ（一覧ページ）からカテゴリページ（詳細ページ）へのリンクを抽出してたどり、カテゴリページからそのカテゴリに含まれるすべての投稿のタイトルを取得します。リンクをたどる流れを図で表します。

```
/（トップページ）
 ├──→/category/autoscraping/（カテゴリページ）
 ├──→/category/professional-services/
 ├──→/category/tools/
 └──→...
```

scrapyコマンドのrunspiderサブコマンド（以降ではscrapy runspiderコマンドと表記します）の引数にファイルパスを指定して実行します。ログが表示され、数秒程度でクロールが完了します。

```
(scraping) $ scrapy runspider myspider.py -o items.jl
2016-05-25 19:00:16 [scrapy] INFO: Scrapy 1.1.0 started (bot: scrapybot)
2016-05-25 19:00:16 [scrapy] INFO: Overridden settings: {'FEED_URI': 'items.jl', 'FEED_FORMAT': 'jl'}
...
2016-05-25 19:00:16 [scrapy] INFO: Spider opened
2016-05-25 19:00:16 [scrapy] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2016-05-25 19:00:17 [scrapy] DEBUG: Crawled (200) <GET https://blog.scrapinghub.com> (referer: None)
2016-05-25 19:00:18 [scrapy] DEBUG: Crawled (200) <GET https://blog.scrapinghub.com/category/ ↵
autoscraping/> (referer: https://blog.scrapinghub.com)
```



```
2016-05-25 19:00:18 [scrapy] DEBUG: Scraped from <200 https://blog.scrapinghub.com/category/ ↵
autoscraping/>
{'title': 'Announcing Portia, the Open Source Visual Web\xa0Scraper! on'}
2016-05-25 19:00:18 [scrapy] DEBUG: Scraped from <200 https://blog.scrapinghub.com/category/ ↵
autoscraping/>
{'title': 'Introducing Dash on'}
...
```

作成されたitems.jlファイルの中身を見ると、投稿のタイトルがJSON Lines形式で取得できていることがわかります。**JSON Lines**[*6]形式とは各行にJSONオブジェクトを持つテキスト形式で、ファイルへの追記が容易な点が特徴です。

```
(scraping) $ cat items.jl
{"title": "Announcing Portia, the Open Source Visual Web\u00a0Scraper! on"}
{"title": "Introducing Dash on"}
{"title": "Spiders activity graphs on"}
{"title": "Autoscraping casts a wider\u00a0net on"}
{"title": "Scrapy Tips from the Pros May 2016\u00a0Edition on"}
...
```

ここでは完成したものを実行するだけでしたが、以降でSpiderを少しずつ作りながら、その動作を解説していきます。

6.2 Spiderの作成と実行

Yahoo!ニュースを対象に、基礎的なSpiderを作成します。

- Yahoo!ニュース
  http://news.yahoo.co.jp/

作成するSpiderは、Yahoo!ニュースのトップページに表示されているトピックスの一覧（**図6.1**）から個別のトピックスへのリンクをたどり、トピックスのタイトルと本文を抽出するというものです。

*6　http://jsonlines.org/