Beautiful Soup (HTML XML 解析)

[Python]

- http://www.crummy.com/software/BeautifulSoup/
- Document
  - http://www.crummy.com/software/BeautifulSoup/documentation.html
  - http://tdoc.info/beautifulsoup/ 日本語

インストール

PIP からインストール

```
# pip install BeautifulSoup
```

ダウンロードして解凍

```
piroto@eisai /cygdrive/c/Users/piroto/Downloads
$ tar -xvf beautifulsoup4-4.1.2.tar.gz
```

■インストール

```
$ cd  beautifulsoup4-4.1.2
$ python setup.py install
```

パーサーのインストール

- Beautiful Soup は Python の標準ライブラリに含まれている HTML パーサーをサポートしています。
- その他にもサードパーティー製のパーサーもサポートしています。
  - その一つが、lxml パーサーで、以下の様にインストールできます。

```
$ apt-get install python-lxml
$ easy_install lxml
$ pip install lxml
```

- 以下の様なパーサーがあります
  - Python 標準ライブラリのパーサー
  - lxml HTML パーサー
  - lxml XML パーサー
  - html5lib

Import

```
from BeautifulSoup import BeautifulSoup         # For processing HTML
from BeautifulSoup import BeautifulStoneSoup    # For processing XML
import BeautifulSoup                            # To get everything
```

No module named BeautifulSoup エラーとなる場合

```
from bs4 import BeautifulSoup # To get everything
```

解析

・文字列およびファイルハンドルによる文書解析

```
soup = BeautifulSoup(open("index.html"))
soup = BeautifulSoup("<html>data</html>")
```

・URL を指定して解析

```
import urllib2
from BeautifulSoup import BeautifulSoup

soup = BeautifulSoup(urllib2.urlopen('http://xxxxx.com'))
```

オブジェクト

・Beautiful Soup は複雑な HTML 文書を、Python オブジェクトのツリーに変換する
・以下の 4 種類のオブジェクトを扱うだけでよい

Tag

・XML、HTML のタグに一致する

```
tag = soup.b
```

Name

・すべてのタグは、.name でアクセスできる名前を持つ
・タグの名称を変更すると、内容も変更される

```
tag.name
```

Attributes

・タグはいくつかの属性をもつ。
・以下の様にアクセス可能

```
tag['class']
```

・.attrs で直接アクセスできる
・追加、削除、変更ができる

```
tag['class'] = 'verybold'
tag['id'] = 1
del tag['class']
del tag['id']
```

複数値属性

・HTML4 では、いくつかの属性で、複数の値を持つことができる
・もっとも知られているのが、class
・Beautiful Soup では、リストとして扱う

```
css_soup = BeautifulSoup('<p class="body strikeout"></p>')
```

```
css_soup.p['class']
# ["body", "strikeout"]
```

・XML では、複数値として扱わない

```
xml_soup = BeautifulSoup('<p class="body strikeout"></p>', 'xml')
xml_soup.p['class']
# u'body strikeout'
```

NavigableString

・文字列は、Beautiful Soup で、NavigableString を利用する
・Python の Unicode 文字列とほぼ同じ
・tree をナビゲートしたり検索したりする機能がサポートされている

```
tag.string
# u'Extremely bold'
type(tag.string)
# <class 'bs4.element.NavigableString'>
```

・unicode() で、Unicode 文字列に変換できる

```
unicode_string = unicode(tag.string)
unicode_string
# u'Extremely bold'
type(unicode_string)
# <type 'unicode'>
```

BeautifulSoup

・文書全体を表す
・Tag オブジェクトとして文書を扱う
・.name は、特別に、"[document]" となる

コメントと特殊な文字列

コメント

・Comment オブジェクトは、NavigableString の特殊型

```
markup = "<b><!--Hey, buddy. Want to buy a used parser?--></b>"
soup = BeautifulSoup(markup)
comment = soup.b.string
type(comment)
# <class 'bs4.element.Comment'>
```

tree のナビゲート

下がる

■タグ名を使ったナビゲート

```
soup.head
soup.title
soup.find_all('a')
```

■ .contents と .children

```
head_tag.contents
[<title>The Dormouse's story</title>]
```

```
for child in title_tag.children:
   print(child)
```

■ .descendants

・.contents および .children は、タグの直接の子供しか考慮していない
・ネストした子供を順次取り出す

```
for child in head_tag.descendants:
    print(child)
# <title>The Dormouse's story</title>
# The Dormouse's story
```

■ .string

```
title_tag.string
# u'The Dormouse's story'
```

■ .strings と stripped_strings

strings

```
for string in soup.strings:
    print(repr(string))
# u"The Dormouse's story"
# u'\n\n'
# u"The Dormouse's story"
# u'\n\n'
```

stripped_strings

```
for string in soup.stripped_strings:
    print(repr(string))
# u"The Dormouse's story"
# u"The Dormouse's story"
```

上がる

■ .parent

```
title_tag = soup.title
title_tag
# <title>The Dormouse's story</title>
title_tag.parent
# <head><title>The Dormouse's story</title></head>
```

■ .paretns

```
link = soup.a
link
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
for parent in link.parents:
```

```
    if parent is None:
        print(parent)
    else:
        print(parent.name)
# p
# body
# html
# [document]
# None
```

横に移動

■ .next_sibling と .prebious_sibling

```
sibling_soup.b.next_sibling
# <c>text2</c>

sibling_soup.c.previous_sibling
# <b>text1</b>
```

■ .next_siblings と .prebious_siblings

```
for sibling in soup.a.next_siblings:
    print(repr(sibling))
for sibling in soup.find(id="link3").previous_siblings:
    print(repr(sibling))
```

後ろ向き前向き

■ .next_element と .previous_element

■ .next_elements と .previous_elements

ツリーの検索

■ string

```
soup.find_all('b')
```

■正規表現

```
import re
for tag in soup.find_all(re.compile("^b")):
    print(tag.name)
# body
# b
```

■リスト

```
soup.find_all(["a", "b"])
```

関数

■ findAll()

タグ名を渡す

```
soup.findAll('b')
```

正規表現を使う

```
import re
tagsStartingWithB = soup.findAll(re.compile('^b'))
```

リストまたは辞書を渡す (結果は同一だが後者が高速)

```
soup.findAll(['title', 'p'])
oup.findAll({'title' : True, 'p' : True})
```

True を渡すとすべてのタグを返す

```
allTags = soup.findAll(True)
```

一つの引数をもち真偽値を返す呼び出し可能オブジェクトを渡すとフィルタできる

```
soup.findAll(lambda tag: len(tag.attrs) == 2)
```

■ find()

・結果が一つしかないと分かっている場合など、最後までスキャンする必要はない利用

```
find(name, attrs, recursive, string, **kwargs)
```

・以下はほぼ同等

```
soup.find_all('title', limit=1)
# [<title>The Dormouse's story</title>]

soup.find('title')
# <title>The Dormouse's story</title>
```

CSS クラスで検索

```
soup.find("b", { "class" : "lime" })
# <b class="lime">Lime</b>
```

■ find_parents() と find_parent()

■ find_next_siblings() と find_next_sibling()

・あるオブジェクトの nextSibling メンバー変数を辿り、指定した Tag あるいは NavigableText を集めてきます。

```
paraText = soup.find(text='This is paragraph ')
paraText.findNextSiblings('b')
# [<b>one</b>]

paraText.findNextSibling(text = lambda(text): len(text) == 1)
# u'.'
```

■ find_previous_siblings() と find_previous_sibling()

■ find_all_next() と find_next()

■ find_all_previous() と find_previous()

CSS selectors

■タグ

```
soup.select("title")
```

■配下のタグ

```
soup.select("body a")
```

■直下のタグ

```
soup.select("head > title")
```

■ CSS class

```
soup.select(".sister")
```

■ ID

```
soup.select("#link1")
soup.select("a#link2")
```

■属性

```
soup.select('a[href]')
```

■属性値

```
soup.select('a[href="http://example.com/elsie"]')
```

ツリーの操作

タグ名と属性の変更

```
tag.name = "blockquote"
tag['class'] = 'verybold'
tag['id'] = 1
del tag['class']
del tag['id']
```

.string **の変更**

```
tag.string = "New link text."
```

append()

```
soup = BeautifulSoup("<a>Foo</a>")
soup.a.append("Bar")
```

BeautifulSoup.new_string() と .new_tag()

■ new_string()

```
soup = BeautifulSoup("<b></b>")
tag = soup.b
tag.append("Hello")
new_string = soup.new_string(" there")
tag.append(new_string)
tag
# <b>Hello there.</b>
tag.contents
# [u'Hello', u' there']
```

■ new_tag()

```
soup = BeautifulSoup("<b></b>")
original_tag = soup.b

new_tag = soup.new_tag("a", href="http://www.example.com")
original_tag.append(new_tag)
original_tag
# <b><a href="http://www.example.com"></a></b>


new_tag.string = "Link text."
original_tag
# <b><a href="http://www.example.com">Link text.</a></b>
```

insert()

```
markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
soup = BeautifulSoup(markup)
tag = soup.a

tag.insert(1, "but did not endorse ")
tag
# <a href="http://example.com/">I linked to but did not endorse  <i>example.com</i></a>
tag.contents
# [u'I linked to ', u'but did not endorse', <i>example.com</i>]
```

insert_before() と insert_after()

clear()

```
markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
soup = BeautifulSoup(markup)
tag = soup.a

tag.clear()
tag
# <a href="http://example.com/"></a>
```

extract()

・タグもしくは文字列をツリーから削除

```
markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
soup = BeautifulSoup(markup)
a_tag = soup.a
```

```
i_tag = soup.i.extract()

a_tag
# <a href="http://example.com/">I linked to</a>

i_tag
# <i>example.com</i>
```

decompose()

・タグをツリーから取り除く

```
markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
soup = BeautifulSoup(markup)
a_tag = soup.a

soup.i.decompose()

a_tag
# <a href="http://example.com/">I linked to</a>
```

replace_with()

・タグおよび文字列をツリーから取り除き、別のタグおよび文字列に置き換える

wrap()

・要素をタグでラップする

```
soup = BeautifulSoup("<p>I wish I was bold.</p>")
soup.p.string.wrap(soup.new_tag("b"))
# <b>I wish I was bold.</b>
```

unwrap()

・タグをはがす

```
markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'
soup = BeautifulSoup(markup)
a_tag = soup.a

a_tag.i.unwrap()
a_tag
# <a href="http://example.com/">I linked to example.com</a>
```

出力

Pretty-printing

■ prettify()

Non-pretty printing

■ unicode() **もしくは** str() **を使う**

XML

```
from BeautifulSoup import BeautifulStoneSoup

def get_translation(itemid):
    '''
    ItemId から、訳語を取得する
    '''
        url   =   r'http://btonic.est.co.jp/NetDic/NetDicV09 .asmx/GetDicItemLite?Dic=EJdict&Item={0
}&Loc=&Prof=XHTML'
    url = url.format(itemid)

    f = urllib2.urlopen(url)

    soup = BeautifulStoneSoup(f.read())
    #print soup.prettify()

    try:
        ret =  soup.body('div')[1].text
    except:
        ret = ''
    return ret
```

例

ページを解析し指定タグのテキストのみ取り出す

```
soup = BeautifulSoup(urllib2.urlopen(url))
for p in soup('p'):
    print p.text
```

タグの文字列を取得する

```
soup = BeautifulSoup(urllib2.urlopen(url))
''.join(soup.findAll(text=True))
```

必要なタグのみ解析する

```
tags = SoupStrainer(['title','img','h1','h2','h3','h4','h5','h6','p','meta','link'])
soup = BeautifulSoup(urllib2.urlopen(url), parseOnlyThese=tags)
```

タグに囲まれた文字列値がある場合出力

```
soup = BeautifulSoup(urllib2.urlopen(url))
if hasattr(soup, "title") and hasattr(soup.title, "string"):
    print soup.title.string
```

Image タグから リンクを抜き出す

```
soup = BeautifulSoup(urllib2.urlopen(url))
for tag in soup.findAll():
    if tag.name == 'img':
        print tag['src']
```