Brownant Documentation

Release 0.1.4

Jiangge Zhang

October 24, 2013

CONTENTS

Brownant is a light and simple crawling framework.

CHAPTER

ONE

USER'S GUIDE

1.1 Introduction

Brownant is a light and simple crawling framework.

1.1.1 Installation

```
$ pip install brownant
```

1.1.2 Links

- Document
- Issue Track

1.1.3 Issues

If you want to report bugs or request features, please create issues on GitHub Issues.

1.1.4 Contributes

You can send a pull reueqst on GitHub.

1.2 Quick Start

There are some simple crawling applications written with Brownant.

1.2.1 The Minimal Demo

This demo could get the download link from the PyPI home page of given project.

```
# example.py
from brownant.app import BrownAnt
from brownant.site import Site
from lxml import html
from requests import Session

site = Site(name="pypi")
http = Session()


@site.route("pypi.python.org", "/pypi/<name>", defaults={"version": None})
@site.route("pypi.python.org", "/pypi/<name>/<version>")
def pypi_info(request, name, version):
    url = request.url.geturl()
    etree = html.fromstring(http.get(url).content)
    download_url = etree.xpath(".//div[@id='download-button']/a/@href")[0]

    return {"name": name, "version": version, "download_url": download_url}

app = BrownAnt()
app.mount_site(site)

if __name__ == "__main__":
    from pprint import pprint
    pprint(app.dispatch_url("https://pypi.python.org/pypi/Werkzeug/0.9.4"))
```

And run it, we will get the output:

```
$ python example.py
{'download_url': 'https://.../source/W/Werkzeug/Werkzeug-0.9.4.tar.gz',
 'name': u'Werkzeug',
 'version': u'0.9.4'}
```

1.2.2 The Declarative Demo

With the declarative usage, the workflow will be flexible and readable.

First, we define the "dinergate" in a site supported module:

```
# sites/pypi.py
from brownant.site import Site
from brownant.dinergate import Dinergate
from brownant.pipeline.network import TextResponseProperty
from brownant.pipeline.html import ElementTreeProperty, XPathTextProperty

site = Site(name="pypi")


@site.route("pypi.python.org", "/pypi/<name>/<version>")
class PythonPackageInfo(Dinergate):

    URL_TEMPLATE = "http://pypi.python.org/pypi/{self.name}/{self.version}"

    text_response = TextResponseProperty()
    etree = ElementTreeProperty()
    download_url = XPathTextProperty(
        xpath=".//div[@id='download-button']/a/@href",
        strip_spaces=True, pick_mode="first"
```

```
    )

    @property
    def info(self):
        return {"name": self.name, "version": self.version,
                "download_url": self.download_url}
```

And then we define an application instance and mount the site.

```
# app.py
from brownant.app import BrownAnt

app = BrownAnt()
app.mount_site("sites.pypi:site")


if __name__ == "__main__":
    from pprint import pprint
    pkg = app.dispatch_url("https://pypi.python.org/pypi/Werkzeug/0.9.4")
    pprint(pkg.info)
```

And run it, we will get the same output.

CHAPTER

TWO

API REFERENCE

2.1 Basic API

The basic API included the application framework and routing system (provided by `werkzeug.routing`) of Brownant.

2.1.1 brownant.app

class `brownant.app.`**BrownAnt**

The app which could manage whole crawler system.

add_url_rule(*host*, *rule_string*, *endpoint*, ***options*)

Add a url rule to the app instance.

The url rule is the same with Flask apps and other Werkzeug apps.

Parameters

- **host** – the matched hostname. e.g. "www.python.org"
- **rule_string** – the matched path pattern. e.g. "/news/<int:id>"
- **endpoint** – the endpoint name as a dispatching key such as the qualified name of the object.

dispatch_url(*url_string*)

Dispatch the URL string to the target endpoint function.

Parameters **url_string** – the origin URL string.

Returns the return value of calling dispatched function.

mount_site(*site*)

Mount a supported site to this app instance.

Parameters **site** – the site instance be mounted.

parse_url(*url_string*)

Parse the URL string with the url map of this app instance.

Parameters **url_string** – the origin URL string.

Returns the tuple as *(url, url_adapter, query_args)*, the url is parsed by the standard library *urlparse*, the url_adapter is from the werkzeug bound URL map, the query_args is a multidict from the werkzeug.

validate_url(*url*)
Validate the `ParseResult` object.

This method will make sure the `parse_url()` could work as expected even meet a unexpected URL string.

Parameters **url** (`ParseResult`) – the parsed url.

2.1.2 brownant.request

class `brownant.request.`**Request**(*url*, *args*)
The crawling request object.

Parameters

- **url** (`urllib.parse.ParseResult`) – the raw URL inputted from the dispatching app.
- **args** (`werkzeug.datastructures.MultiDict`) – the query arguments decoded from query string of the URL.

2.1.3 brownant.site

class `brownant.site.`**Site**(*name*)
The site supported object which could be mounted to app instance.

Parameters **name** – the name of the supported site.

play_actions(*target*)
Play record actions on the target object.

Parameters **target** (`BrownAnt`) – the target which recive all record actions, is a brown ant app instance normally.

record_action(*method_name*, **args*, ***kwargs*)
Record the method-calling action.

The actions expect to be played on an target object.

Parameters

- **method_name** – the name of called method.
- **args** – the general arguments for calling method.
- **kwargs** – the keyword arguments for calling method.

route(*host*, *rule*, ***options*)
The decorator to register wrapped function as the brown ant app.

All optional parameters of this method are compatible with the `add_url_rule()`.

Registered functions or classes must be import-able with its qualified name. It is different from the `Flask`, but like a lazy-loading mode. Registered objects only be loaded before the first using.

The right way:

```
@site.route("www.example.com", "/item/<int:item_id>")
def spam(request, item_id):
    pass
```

The wrong way:

```
def egg():
    # the function could not be imported by its qualified name
    @site.route("www.example.com", "/item/<int:item_id>")
    def spam(request, item_id):
        pass

egg()
```

Parameters

- **host** – the limited host name.
- **rule** – the URL path rule as string.
- **options** – the options to be forwarded to the `werkzeug.routing.Rule` object.

2.1.4 brownant.exceptions

exception `brownant.exceptions.`**`BrownAntException`**
The base exception of the brown ant framework.

exception `brownant.exceptions.`**`NotSupported`**
Bases: `brownant.exceptions.BrownAntException`

The given URL or other identity is from a platform which not support.

This exception means any url rules of the app which matched the URL could not be found.

2.1.5 brownant.utils

`brownant.utils.`**`to_bytes_safe`**(*text*, *encoding='utf-8'*)
Convert the input value into bytes type.

If the input value is string type and could be encode as UTF-8 bytes, the encoded value will be returned. Otherwise, the encoding has failed, the origin value will be returned as well.

Parameters

- **text** – the input value which could be string or bytes.
- **encoding** – the expected encoding be used while converting the string input into bytes.

Return type `bytes`

2.2 Declarative API

The declarative API is around the "dinergate" and "pipeline property".

2.2.1 brownant.dinergate

class `brownant.dinergate.`**`Dinergate`**(*request*, *http_client=None*, ***kwargs*)
The simple classify crawler.

In order to work with unnamed properties such as the instances of `PipelineProperty`, the meta class `DinergateType` will scan subclasses of this class and name all unnamed members which are instances of `cached_property`.

Parameters

- **request** (`Request`) – the standard parameter passed by app.
- **http_client** (`requests.Session`) – the session instance of python-requests.
- **kwargs** – other arguments from the URL pattern.

URL_TEMPLATE = None
the URL template string for generating crawled target. the *self* could be referenced in the template. (e.g. *"http://www.example.com/items/{self.item_id}?page={self.page}"*)

url
The fetching target URL.

The default behavior of this property is build URL string with the `URL_TEMPLATE`.

The subclasses could override `URL_TEMPLATE` or use a different implementation.

class `brownant.dinergate.`**DinergateType**
Bases: `type`

The metaclass of `Dinergate` and its subclasses.

This metaclass will give all members are instance of `cached_property` default names. It is because many pipeline properties are subclasses of `cached_property`, but them would not be created by decorating functions. They will has not built-in `__name__`, which may cause them could not cache values as expected.

2.2.2 brownant.pipeline.base

class `brownant.pipeline.base.`**PipelineProperty**(***kwargs*)
Bases: `werkzeug.utils.cached_property`

The base class of pipeline properties.

There are three kinds of initial parameters.

- The required attribute. If a keyword argument's name was defined in `required_attrs`, it will be assigned as an instance attribute.
- The attr_name. It is the member of `attr_names`, whose name always end with *_attr*, such as *text_attr*.
- The option. It will be placed at an instance owned `dict` named `options`. The subclasses could set default option value in the `prepare()`.

A workable subclass of `PipelineProperty` should implement the abstruct method `provide_value()`, which accept an argument, the instance of `Dinergate`.

Overriding `prepare()` is optional in subclasses.

Parameters **kwargs** – the parameters with the three kinds.

provide_value(*obj*)
The abstruct method which should be implemented by subclasses. It provide the value expected by us from the subject instance.

Parameters **obj** (`Dinergate`) – the subject instance.

attr_names = None
the definition of attr_names

get_attr(*obj*, *name*)
Get attribute of the target object with the configured attribute name in the `attr_names` of this instance.

Parameters

- **obj** (`Dinergate`) – the target object.
- **name** – the internal name used in the `attr_names`. (.e.g. *"text_attr"*)

options = None
the definition of options

prepare()
This method will be called after instance ininialized. The subclasses could override the implementation.

In general purpose, the implementation of this method should give default value to options and the members of `attr_names`.

Example:

```
def prepare(self):
    self.attr_names.setdefault("text_attr", "text")
    self.options.setdefault("use_proxy", False)
```

required_attrs = set([])
the names of required attributes.

2.2.3 brownant.pipeline.network

class brownant.pipeline.network.**URLQueryProperty**(***kwargs*)
The query argument property. The usage is simple:

```
class MySite(Dinergate):
    item_id = URLQueryProperty(name="item_id", type=int)
```

It equals to this:

```
class MySite(Dinergate):
    @cached_property
    def item_id(self):
        value = self.request.args.get("item_id", type=int)
        if not value:
            raise NotSupported
        return value
```

A failure convertion with given type (`ValueError` be raised) will lead the value fallback to `None`. It is the same with the behavior of the `MultiDict`.

Parameters

- **name** – the query argument name.
- **request_attr** – optional. default: *"request"*.
- **type** – optionl. default: *None*. this value will be passed to `get()`.
- **required** – optionl. default: *True*. while this value be true, the `NotSupported` will be raised for meeting empty value.

class brownant.pipeline.network.**TextResponseProperty**(***kwargs*)
The text response which returned by fetching network resource.

Getting this property is network I/O operation in the first time. The http request implementations are all provided by `requests`.

The usage example:

```
class MySite(Dinergate):
    foo_http = requests.Session()
    foo_url = "http://example.com"
    foo_text = TextResponseProperty(url_attr="foo_url",
                                    http_client="foo_http",
                                    proxies=PROXIES)
```

Parameters

- **url_attr** – optional. default: *"url"*. it point to the property which could provide the fetched url.
- **http_client_attr** – optional. default: *"http_client"*. it point to an http client property which is instance of `requests.Session`
- **kwargs** – the optional arguments which will be passed to `requests.Session.get()`.

2.2.4 brownant.pipeline.html

class `brownant.pipeline.html.`**`ElementTreeProperty`**(***kwargs*)

The element tree built from a text response property. There is an usage example:

```
class MySite(Dinergate):
    text_response = "<html></html>"
    div_response = "<div></div>"
    xml_response = (u"<?xml version='1.0' encoding='UTF-8'?>"
                    u"<result>\u6d4b\u8bd5</result>")
    etree = ElementTreeProperty()
    div_etree = ElementTreeProperty(text_response_attr="div_response")
    xml_etree = ElementTreeProperty(text_response_attr="xml_response",
                                    encoding="utf-8")

site = MySite(request)
print(site.etree)  # output: <Element html at 0x1f59350>
print(site.div_etree)  # output: <Element div at 0x1f594d0>
print(site.xml_etree)  # output: <Element result at 0x25b14b0>
```

Parameters

- **text_response_attr** – optional. default: *"text_response"*.
- **encoding** – optional. default: *None*. The output text could be encoded to a specific encoding.

New in version 0.1.4: The *encoding* optional parameter.

class `brownant.pipeline.html.`**`XPathTextProperty`**(***kwargs*)

The text extracted from a element tree property by XPath. There is an example for usage:

```
class MySite(Dinergate):
    # omit page_etree
    title = XPathTextProperty(xpath=".//h1[@id='title']/text()",
                              etree_attr="page_etree",
                              strip_spaces=True,
                              pick_mode="first")
    links = XPathTextProperty(xpath=".//*[@id='links']/a/@href",
                              etree_attr="page_etree",
                              strip_spaces=True,
```

```
                pick_mode="join",
                joiner="|")
```

Parameters

- **xpath** – the xpath expression for extracting text.
- **etree_attr** – optional. default: *"etree"*.
- **strip_spaces** – optional. default: *False*. if it be *True*, the spaces in the beginning and the end of texts will be striped.
- **pick_mode** – optional. default: *"join"*, and could be "join", "first" or "keep". while *"join"* be detected, the texts will be joined to one. if the *"first"* be detected, only the first text would be picked. if the *"keep"* be detected, the original value will be picked.
- **joiner** – optional. default is a space string. it is no sense in assigning this parameter while the *pick_mode* is not *"join"*. otherwise, the texts will be joined by this string.

New in version 0.1.4: The new option value *"keep"* of the *pick_mode* parameter.

CHAPTER

THREE

RELEASE CHANGES

3.1 Release 0.2

(in development)

3.2 Release 0.1.4 (Oct 24, 2013)

- Fix the RequestRedirect raised problem.
- Add the new pick mode "keep" of XPathTextProperty. (by VeryCB)
- Add the encoding parameter of the ElementTreeProperty. That could let the property provide bytes instead of unicode string. (by VeryCB)

3.3 Release 0.1.3 (Oct 19, 2013)

- Fix the broken CI (travis-ci).

3.4 Release 0.1.2 (Oct 19, 2013)

- Fix some unicode compatible problems for URL string.
- Prevent the invalid URL string input.
- Change the theme of document into built-in one named "nature".

3.5 Release 0.1.1 (Sep 30, 2013)

- Refine the documents and give an example in the Quick Start section.

3.6 Release 0.1.0 (Sep 29, 2013)

- First public release.

CHAPTER

FOUR

AUTHOR & CONTRIBUTOR

- Jiangge Zhang <tonyseek@gmail.com>
- VeryCB <imcaibin@gmail.com>

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*