WebOb Toolkit Documentation

Release 0.3

Tom Willis

August 16, 2015

Contents

WebOb Toolkit is a tool kit for building HTTP clients using WebOb request and response objects and wsgi middleware.

You may already be familiar with Webob request and response objects if you have experience with a web framework that uses them such as pyramid or WebTest .

CHAPTER 1

Usage

Here are some examples of how WebOb Toolkit can be used.

1.1 A Very Simple HTTP Client

If you didn't know, WebOb's Request object grew a "get_response" method for sending an http request to either a wsgi application, or a url and returning the response.

```
>>> from webob import Request
>>> str(Request.blank("https://google.com").get_response())
'301 Moved Permanently\nLocation: https://www.google.com/\nContent-Type: text/html; charset=UTF-8\nDa
>>>
```

That is a pretty neat trick, but as http clients go, typically you need other functionality like handling http compression for example or handling cookies.

WebOb Toolkit provides this additional functionality as wsgi middleware which allows you to compose your own solutions in much the same way as you compose wsgi applications to be used as HTTP Servers.

1.2 Handling Compression

According to the http compression article on wikipedia, an http client can request a compressed response by including an "Accept-Encoding" header with the request. In WebOb you would do...

```
>>> from webob import Request
>>> Request.blank("https://github.com", headers={"Accept-Encoding": "gzip, deflate"}).get_response()
'gzip'
>>>
```

As you can see, we requested gzipped content from github, and it responded nicely. However, if we were to do anything with the body of the response we would have to uncompress it first. So, it seems that the rules for compression is to a header with each request and uncompress the body of each response if the response includes a "Content-Encoding" header.

The next example, uses webobtoolkit's decode_filter to handle compressed responses.

```
>>> from webob import Request
>>> from webob.client import send_request_app as app
>>> from webobtoolkit.filters import decode_filter
>>> app_gzip = decode_filter(app)
```

```
>>> Request.blank("https://github.com", headers={"Accept-Encoding": "gzip"}).get_response(app).body[
'\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x03\xdd[\xdbr\xe4\xc6\x91}\xf7W\x94\x9b\x1b\xa3\xdd0\xd1\xf7\x
>>> Request.blank("https://github.com", headers={"Accept-Encoding": "gzip"}).get_response(app_gzip).b
'<!DOCTYPE html>\n<html>\n  <head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# object: http
>>>
```

From the above example you will notice a couple of differences from previous examples. Firstly we are importing WebOb's send_request_app and some wsgi middleware from webobtoolkit for handling compressed responses. We wrap webob's app with the decode_filter to create an app that will decompress any response we may encounter.

The first call to github is through webobs app. And as you can see, the response is compressed just like we asked. The second call is through the new app we created by wrapping webob's app with webobtoolkits decode_filter, and as you can see the response has been decompressed.

The name "filter" is being used here to differentiate between wsgi middleware which is usually used in the context of servers and wsgi middleware that is intended for use with clients. Filters and middleware are identical in regards to how they are implemented.

Here is how the decode_filter is implemented. As you can see, it doesn't take much to write a filter.

1.3 A More Robust HTTP Client

requests is a good example of a more useful http client. According to the docs it includes a lot of useful things that one would want for in an http client. Some of the things it includes are.

- handling compression
- handling cookies
- handling redirects
- guessing, handling charset decoding
- connection pooling
- ssl verification
- form posts
- file uploads

And probably much more.

A lot of this functionality can be written on top of webob and a series of filters. We've already seen how one might handle compression. WebOb toolkit provides filters for handling cookies, redirects and handling unspecified charsets.

```
>>> from webob import Request
>>> from webob.client import send_request_app
>>> from webobtoolkit import filters
>>> requests_app = filters.auto_redirect_filter(filters.cookie_filter(filters.decode_filter(filters.c
>>> Request.blank("https://google.com").get_response(requests_app)
>>> str(Request.blank("https://google.com").get_response(requests_app))[:500]
'200 OK\nDate: Sat, 08 Mar 2014 17:35:40 GMT\nExpires: -1\nCache-Control: private, max-age=0\nContent
```

We construct requests_app out of a number of filters that are for handling requests and responses.

- if a charset is not specified on the response(which sometimes happens), a safe default is chosen in order to reduce the chances for decoding errors
- the client advertises support for gzip encoding and decompress the response if necessary
- cookies will be persisted for each response and sent for each request

- if a redirect is encountered follow it automatically.
- form posts are handled by webob already, though some might prefer a better syntax.

1.4 Todo

- for connection pooling, urllib3 provides an implementation that could be easily used to construct an alternate send_request_app (see: webob.client.SendRequest)
- ssl verification is also provided by urllib3

1.4.1 Change History

0.2.4

- merge pull request to address issue #14 https://github.com/twillis/webobtoolkit/issues/14

2.1 0.2.3

- python 3 compatible
- rewrite docs
- make it easier to run tests with py.test via python setup.py test

CHAPTER 3

0.2.2

- merge pull request to address issue #8 https://github.com/twillis/webobtoolkit/issues/8

0.2.1

- check for content_type before attempting to set charset because some webservers behave badly

0.2

- documentation updates
- file uploads syntax sugar
- changed client to not rely on a global pipeline app

0.1.3.3

- breakout stringify dict method as it may be useful

0.1.3.2

- client handles multidict to querystring better

0.1.3.1

- option for enabling auto redirect in test client

0.1.3

- added sugar for the HTTP OPTIONS method

0.1.2

- added preconfigured test client that is similar to webtest

0.1.1

- added file upload support
- changed handling of query_string to look at the url if it is not passed in as a separate param.

0.1

initial release because I need this in my day job

0.0

first pass at a client, and first pass at docs.