httpcache Documentation

Release 0.1.3

Cory Benfield

January 28, 2015

Contents

Love Requests? Wish it had HTTP caching? Well then, this is the project for you.

Built from the ground up for combining with your favourite HTTP library, this library provides totally transparent HTTP caching. Fully RFC 2616-compliant, no muss, no fuss. Just plug in and go.

```
>>> import requests
>>> from httpcache import CachingHTTPAdapter
>>> s = requests.Session()
>>> s.mount('http://', CachingHTTPAdapter())
>>> s.mount('https://', CachingHTTPAdapter())
```

Any request passed through that session will automatically use the HTTP cache. Guess what? Requests just got better.

If this library doesn't float your boat, you can also take a look at the excellent CacheControl library, which directly ports httplib2's caching algorithms.

CHAPTER 1

Features

httpcache fully supports HTTP/1.1 caching, and ties deep into Requests. That means:

- Supports `Expires` headers.
- Supports `Cache-Control` headers.
- Understands `304 Not Modified` responses.
- Can do validation caching, i.e. `If-Modified-Since` headers.
- Correctly navigates the waters of HTTP verbs and urls.
- Fully supports RFC 2616.

Caching with Requests should be as easy as HTTP with Requests. Stop worrying about it. Just benefit from it.

CHAPTER 2

Versions

httpcache supports all the versions of Python that Requests supports. We intend to do this indefinitely. Currently, this means we support 2.6, 2.7 and 3.3. It is possible that httpcache functions on earlier versions of Python, but such functionality is not supported and may be broken in any version change.

CHAPTER 3

Contents

3.1 Quickstart

The beautiful thing about httpcache is that all you need is the quickstart guide. Much like Requests has some very strong opinions about 'the right way' to do HTTP, httpcache has some very strong opinions about 'the right way' to do HTTP caching. This means there is essentially no configuration: httpcache is built from the ground up to be plug-and-play.

3.1.1 Installing httpcache

The recommended way to obtain a copy of httpcache is to get it from PyPI, using `pip`. From your command-line shell, run:

```
$ pip install httpcache
```

If you can't install httpcache using `pip`, try using `easy_install`:

```
$ easy_install httpcache
```

Alternatively, download a gzipped tarball from PyPI. If you extract that, you should find a `setup.py` file. You can install httpcache by running:

```
$ python setup.py install
```

3.1.2 Install The Adapter

The expected way to use httpcache is to install the Requests Transport Adapter that the module provides. This requires a couple of lines of code before you start making web requests:

```
import requests
from httpcache import CachingHTTPAdapter

s = requests.Session()
s.mount('http://', CachingHTTPAdapter())
s.mount('https://', CachingHTTPAdapter())
```

You can now use that Requests session just like normal. Under the covers, httpcache takes care of all the caching decisions for you.

3.1.3 Using The Raw Cache

Although it's not recommended, httpcache grudgingly allows you lower-level access to the cache object itself. You can instantiate it like so:

```
from httpcache import HTTPCache
cache = HTTPCache()
```

You can then interact with the cache directly. Because it isn't recommended, no further detail is given here.

BE WARNED: This API is considered 'semi-public'. The author reserves the right to change this API as part of a minor version increase. If you are interacting with the raw cache, be sure to consult the changelog before making any minor version upgrade to determine whether the update will break your code.

3.1.4 Configuration

httpcache is firm in its belief that it knows what is best for you. This means that there is very little in the way of configuration. In fact, you only have one option: how many entries to cache. If you wanted to cache a maximum of 100 pages, then you would use:

```
CachingHTTPAdapter(capacity=100)
```

3.2 Interfaces

This section of the documentation discusses the httpcache interface.

3.2.1 Caching HTTP Adapter

The Caching HTTP Adapter is the recommended interface to httpcache. A transport adapter that plugs directly into a Requests session, this represents the easiest way to add caching to Requests.

class `httpcache.`**`CachingHTTPAdapter`**(*capacity=50*, ***kwargs*)

A HTTP-caching-aware Transport Adapter for Python Requests. The central portion of the API.

Parameters **capacity** – The maximum capacity of the backing cache.

`add_headers`(*request*, ***kwargs*)

Add any headers needed by the connection. As of v2.0 this does nothing by default, but is left for overriding by users that subclass the `HTTPAdapter`.

This should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **request** – The `PreparedRequest` to add headers to.
- **kwargs** – The keyword arguments from the call to send().

`build_response`(*request*, *response*)

Builds a Response object from a urllib3 response. May involve returning a cached Response.

Parameters

- **request** – The Requests `PreparedRequest` object sent.
- **response** – The urllib3 response.

cache = None
The HTTP Cache backing the adapter.

cert_verify(*conn*, *url*, *verify*, *cert*)
Verify a SSL certificate. This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **conn** – The urllib3 connection object associated with the cert.
- **url** – The requested URL.
- **verify** – Whether we should actually verify the certificate.
- **cert** – The SSL certificate to verify.

close()
Disposes of any internal state.

Currently, this just closes the PoolManager, which closes pooled connections.

get_connection(*url*, *proxies=None*)
Returns a urllib3 connection for the given URL. This should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **url** – The URL to connect to.
- **proxies** – (optional) A Requests-style dictionary of proxies used on this request.

init_poolmanager(*connections*, *maxsize*, *block=False*, ***pool_kwargs*)
Initializes a urllib3 PoolManager.

This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **connections** – The number of urllib3 connection pools to cache.
- **maxsize** – The maximum number of connections to save in the pool.
- **block** – Block when no free connections are available.
- **pool_kwargs** – Extra keyword arguments used to initialize the Pool Manager.

proxy_headers(*proxy*)
Returns a dictionary of the headers to add to any request sent through a proxy. This works with urllib3 magic to ensure that they are correctly sent to the proxy, rather than in a tunnelled request if CONNECT is being used.

This should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **proxies** – The url of the proxy being used for this request.
- **kwargs** – Optional additional keyword arguments.

proxy_manager_for(*proxy*, ***proxy_kwargs*)
Return urllib3 ProxyManager for the given proxy.

This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **proxy** – The proxy to return a urllib3 ProxyManager for.
- **proxy_kwargs** – Extra keyword arguments used to configure the Proxy Manager.

Returns ProxyManager

request_url(*request*, *proxies*)

Obtain the url to use when making the final request.

If the message is being sent through a HTTP proxy, the full URL has to be used. Otherwise, we should only use the path portion of the URL.

This should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **request** – The `PreparedRequest` being sent.
- **proxies** – A dictionary of schemes to proxy URLs.

send(*request*, ***kwargs*)

Sends a PreparedRequest object, respecting RFC 2616's rules about HTTP caching. Returns a Response object that may have been cached.

Parameters **request** – The Requests `PreparedRequest` object to send.

3.2.2 HTTP Cache

The HTTP Cache is the object that backs the Caching HTTP Adapter. This provides the primary caching functionality. The object is documented here for reference and for developers who believe they need better control over the caching behaviour than the Caching HTTP Adapter provides.

BE WARNED: This API is considered 'semi-public'. The author reserves the right to change this API in a minor version increase, rather than having to wait for a major version increase as per Semantic Versioning.

class `httpcache.`**HTTPCache**(*capacity=50*)

The HTTP Cache object. Manages caching of responses according to RFC 2616, adding necessary headers to HTTP request objects, and returning cached responses based on server responses.

This object is not expected to be used by most users. It is exposed as part of the public API for users who feel the need for more control. This API may change in a minor version increase. Be warned.

Parameters **capacity** – (Optional) The maximum capacity of the HTTP cache.

capacity = None

The maximum capacity of the HTTP cache. When this many cache entries end up in the cache, the oldest entries are removed.

handle_304(*response*)

Given a 304 response, retrieves the cached entry. This unconditionally returns the cached entry, so it can be used when the 'intelligent' behaviour of retrieve() is not desired.

Returns None if there is no entry in the cache.

Parameters **response** – The 304 response to find the cached entry for. Should be a Requests `Response`.

retrieve(*request*)

Retrieves a cached response if possible.

If there is a response that can be unconditionally returned (e.g. one that had a Cache-Control header set), that response is returned. If there is one that can be conditionally returned (if a 304 is returned), applies an If-Modified-Since header to the request and returns None.

Parameters **request** – The Requests `PreparedRequest` object.

store(*response*)

Takes an HTTP response object and stores it in the cache according to RFC 2616. Returns a boolean value indicating whether the response was cached or not.

Parameters **response** – Requests `Response` object to cache.

3.3 Contributing

Contributions are always welcome! Please abide by the following rules when contributing:

1. Check that no-one has opened an issue already covering your bug. If you open a duplicate issue, the maintainer will give you a stern look.
2. Fork the Github repository and start writing your tests. If you're fixing a bug, I recommend writing a failing test first and working until it passes. If you're adding a feature, you're free to add tests after you write the functionality, but please test the functionality thoroughly.
3. Send a Pull Request. If I don't respond within a couple of days, please shout at me on Twitter or via email until I do something about it.

h

A

B

C

G

H

I

P

R

S