httpy

Release 2.0.3

Adam Jenca

Feb 27, 2024

CONTENTS

A Python lightweight socket-based library to create HTTP(s) and WebSocket connections.

CHAPTER ONE

FEATURES

- Cookies support
- Caching support
- Easy debugging
- HTTP Basic and Digest authentication
- Form support
- Keep-Alive and Sessions support
- JSON support
- Sessions support
- Runs in PyPy
- Independent of http.client
- HTTP/2 Support
- Async IO support

CHAPTER

TWO

REQUIREMENTS

- Python>=3.6

CHAPTER

THREE

INSTALLATION

3.1 Any platform

3.1.1 Git

1. `git clone https://github.com/jenca-adam/httpy`
2. `cd httpy`
3. `python3 setup.py install`

The Python version check will be performed automatically

3.1.2 Pip

1. `python3 -m pip install httpy`

3.2 Arch Linux

1. `yay -S httpy`

CHAPTER

FOUR

USAGE

REFERENCE

4.1 HTTP

It's easy.

```
import httpy
resp = httpy.request("https://example.com/") # Do a request
resp.content #Access content
```

4.1.1 Specifying a HTTP version

Set the `http_version` argument, but keep in mind the following

1. You can't make an asynchronous request using HTTP/1.1
2. HTTP/2 requests can't be performed over insecure (http scheme) connections.

If you don't set it, the HTTP version will be automatically detected using ALPN <https://datatracker.ietf.org/doc/html/rfc7301>.

Valid `http_version` values are `"1.1"` and `"2"`.

4.1.2 Non-blocking requests

```
import httpy
pending = httpy.request("https://example.com/", blocking = False)
```

`PendingRequest.response` returns the result of the response. You can check if the request is already done using `PendingRequest.finished`

4.1.3 I want cookies!

The Dir class allows you to store httpy's data (cache and cookies) on the path of your choice. By default, the data is stored in ~/.cache/httpy. If you want to store the data without using the Dir class, use the enable_cookies or enable_cache argument of request. .. code-block:: python

> import httpy directory = httpy.Dir("your/path") directory.request("https://example.com/") # …

4.1.4 Keep-Alive requests

If you want to reuse a connection, it is highly recommended to use a Session class. It offers more control over connection closure than the standard request

```python
import httpy
session = httpy.Session()
session.request("https://example.com/")
```

HTTPy sets Connection: close by default in non-Session requests. If you want to keep the connection alive outside a session, you must specify so in the headers argument.

4.1.5 Asynchronous requests

You can perform async requests using the async_request method.

The simplest use case:

```python
import httpy

async def my_function():
    return await httpy.request("https://example.com/")
```

If you want to perform multiple requests at once on the same connection (i.e. with asyncio.gather), use the initiate_http2_connection method of Session:

```python
import httpy
import asyncio

async def my_function():
    session = httpy.Session()
    await session.initiate_http2_connection(host="example.com")
    return await asyncio.gather(*(session.async_request("https://www.example.com/") for␣
↪_ in range(69)))
```

Session and Dir and everything with a request() method has an async_request() equivalent.

4.1.6 Response class attributes

The Response class returned by `request()` has some useful attributes:

`Response.content`

The response content as `bytes`. Example:

```python
import httpy
resp = httpy.request("https://www.google.com/")
print(resp.content)
#b'!<doctype html>\n<html>...
```

`Response.status`

The response status as a `Status` object. Example:

```python
import httpy
resp = httpy.request("https://www.example.com/this_url_doesnt_exist")
print(resp.status)
# 404
print(resp.status.reason)
# NOT FOUND
print(resp.status.description)
# indicates that the origin server did not find a current representation for the target␣
↪resource or is not willing to disclose that one exists.
print(resp.status>400)
# True
```

`Status` subclasses `int`.

`Response.history`

All the redirects on the way to this response as `list`.

Example:

```python
import httpy
resp = httpy.request("https://httpbin.org/redirect/1")
print(resp.history)
# [<Response GET [302 Found] (https://httpbin.org/redirect/1/)>, <Response GET [200 OK]␣
↪(https://httpbin.org/get/)>]
```

`Response.history` is ordered from oldest to newest

Response.fromcache

Indicates whether the response was loaded from cache (bool).

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.fromcache)
# False
resp = httpy.request("https://example.com/")
print(resp.fromcache)
# True
```

Response.request

Some of the attributes of the request that produced this response, as a Request object.

Request's attributes

- Request.url - the URL requested (str)
- Request.headers - the requests' headers (Headers)
- Request.socket - the underlying connection (either socket.socket or httpy.http2.connection.HTTP2Connection)
- Request.cache - the same as Response.fromcache (bool)
- Request.http_version - the HTTP version used (str)
- Request.method - the HTTP method used (str)

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.request.url)
# https://example.com/
print(resp.request.headers)
# {'Accept-Encoding': 'gzip, deflate, identity', 'Host': 'example.com', 'User-Agent': 'httpy/2.
↪0.0', 'Connection': 'close', 'Accept': '*/*'}
print(resp.request.method)
# GET
```

Response.original_content

Raw content received from the server, not decoded with Content-Encoding (`bytes`).

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.original_content)
# b'\x1f\x8b\x08\x00\xc2 ...
```

Response.time_elapsed

Time the request took, in seconds. Only the loading time of this particular request, doesn't account for redirects. (`float`).

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.time_elapsed)
# 0.2497
```

Response.speed

The download speed for the response, in bytes per second. (`float`). Might be different for HTTP/2 request. Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.speed)
# 2594.79
```

Response.content_type

The response's `Content-Type` header contents, with the charset information stripped. If the headers lack `Content-Type`, it's `text/html` by default.

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.content_type)
# text/html
```

Response.charset (property)

Gets the charset of the response (str or None):

1. If a charset was specified in the response headers, return it
2. If a charset was not specified, but chardet is available, try to detect the charset (Note that this still returns None if chardet fails)
3. If a charset was not specified, and chardet is not available, return None

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.charset)
# UTF-8
```

Response.string (property)

Response.content, decoded using Response.charset (str)

Warning: Do not try to access Response.string, if Response.charset is None, unless you are absolutely sure the response data is decodable by the default locale encoding.

For ASCII responses this is probably harmless, but you have been warned!

Example:

```python
import httpy
resp = httpy.request("https://example.com/")
print(resp.string)
#<!doctype html>
...
```

Response.json (property)

If Response.content_type is application/json, try to parse Response.string using JSON. Throw an error otherwise.

Warning: The same as above applies.

Example:

```python
import httpy
resp = httpy.request("https://httpbin.org/get")
print(resp.json["url"])
# https://httpbin.org/get
```

`Response.method`

The same as `Response.request.method`

4.2 WebSockets

Easy again…

```
>>> import httpy
>>> sock = httpy.WebSocket("wss://echo.websocket.events/")# create a websocket␣
↪client(echo server example)
>>> sock.send("Hello, world!")# you can send also bytes
>>> sock.recv()
"Hello, world!"
```

CHAPTER

FIVE

EXAMPLES

5.1 POST method

5.1.1 Simple Form

```
import httpy
resp = httpy.request("https://example.com/", method="POST", body = {"foo":"bar"})
# ...
```

5.1.2 Sending files

```
import httpy
resp = httpy.request("https://example.com/", method = "POST", body = { "foo" : "bar",
↪"file" : httpy.File.open( "example.txt" ) })
# ...
```

5.1.3 Sending binary data

```
import httpy
resp = httpy.request("https://example.com/", method = "POST", body= b" Hello, World ! ")
# ...
```

5.1.4 Sending plain text

```
resp = httpy.request("https://example.com/", method = "POST", body = "I support Ünicode !
↪")
# ...
```

5.1.5 Sending JSON

```
resp = httpy.request("https://example.com/", method = "POST", body = "{\"foo\" : \"bar\"␣
↪}", content_type = "application/json")
# ...
```

CHAPTER

SIX

DEBUGGING

Just set debug to True :

```
>>> import httpy
>>> httpy.request("https://example.com/",debug=True)
[INFO][request](1266): request() called.
[INFO][_raw_request](1112): _raw_request() called.
[INFO][_raw_request](1113): Accessing cache.
[INFO][_raw_request](1120): No data in cache.
[INFO][_raw_request](1151): Establishing connection
[INFO]Connection[__init__](778): Created new Connection upon <socket.socket fd=3,␣
↪family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=6, laddr=('192.168.
↪100.88', 58998), raddr=('93.184.216.34', 443)>

send:
GET / HTTP/1.1
Accept-Encoding: gzip, deflate, identity
Host: www.example.com
User-Agent: httpy/1.1.0
Connection: keep-alive

response:
HTTP/1.1 200 OK

Content-Encoding: gzip
Age: 438765
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 13 Apr 2022 12:59:07 GMT
Etag: "3147526947+gzip"
Expires: Wed, 20 Apr 2022 12:59:07 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7F37)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648
<Response [200 OK] (https://www.example.com/)>
```