
aio sonic

Release 0.7.1

Johanderson Mogollon

Nov 21, 2020

CONTENTS

1	Features	3
2	Requirements	5
3	Install	7
4	Getting Started	9
5	Benchmarks	11
6	Contributing	13
7	Indices and tables	15
7.1	Examples	15
7.2	Reference	16
	Index	19

Really Fast Python asyncio HTTP 1.1 client, Support for http 2.0 is planned.

Current version is 0.7.1.

Repo is hosted at [Github](#).

FEATURES

- Keepalive and Smart Pool of Connections
- Multipart File Uploads
- Chunked responses handling
- Chunked requests
- Fully type annotated.
- Connection Timeouts
- Automatic Decompression
- Follow Redirects
- 100% test coverage.

REQUIREMENTS

- Python>=3.6

CHAPTER THREE

INSTALL

```
$ pip install aiosonic
```


GETTING STARTED

```
import asyncio
import aiohttp
import json

async def run():
    client = aiohttp.HttpClient()

    # #####
    # Sample get request
    # #####
    response = await client.get('https://www.google.com/')
    assert response.status_code == 200
    assert 'Google' in (await response.text())

    # #####
    # Post data as multipart form
    # #####
    url = "https://postman-echo.com/post"
    posted_data = {'foo': 'bar'}
    response = await client.post(url, data=posted_data)

    assert response.status_code == 200
    data = json.loads(await response.content())
    assert data['form'] == posted_data

    # #####
    # Posted as json
    # #####
    response = await client.post(url, json=posted_data)

    assert response.status_code == 200
    data = json.loads(await response.content())
    assert data['json'] == posted_data

    # #####
    # Sample request + timeout
    # #####
    from aiohttp.timeout import Timeouts
    timeouts = Timeouts(
        sock_read=10,
        sock_connect=3
    )
    response = await client.get('https://www.google.com/', timeouts=timeouts)
```

(continues on next page)

(continued from previous page)

```
assert response.status_code == 200
assert 'Google' in (await response.text())

print('success')

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(run())
```

BENCHMARKS

Some benchmarking

```
$ python ./tests/performance.py
doing tests...
{
  "aiosonic": "1000 requests in 110.56 ms",
  "aiosonic cyclic": "1000 requests in 207.75 ms",
  "aiohttp": "1000 requests in 357.19 ms",
  "requests": "1000 requests in 4274.21 ms",
  "httpx": "1000 requests in 800.98 ms"
}
aiosonic is 223.05% faster than aiohttp
aiosonic is 3765.79% faster than requests
aiosonic is 87.90% faster than aiosonic cyclic
aiosonic is 624.45% faster than httpx
```


CONTRIBUTING

1. Fork
2. create a branch *feature/your_feature*
3. commit - push - pull request

Thanks :)

INDICES AND TABLES

- genindex
- modindex
- search

7.1 Examples

TODO: More examples

7.1.1 Download file

```
import asyncio
import aiohttp
import json

async def run():
    url = 'https://images.dog.ceo/breeds/leonberg/n02111129_2301.jpg'
    client = aiohttp.Client()

    res = await client.get(url)
    assert res.status_code == 200

    if res.chunked:
        # write in chunks
        with open('dog_image.jpg', 'wb') as _file:
            async for chunk in res.read_chunks():
                _file.write(chunk)
    else:
        # or write all bytes, for chunked this also works
        with open('dog_image.jpg', 'wb') as _file:
            _file.write(await res.content())

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(run())
```

7.2 Reference

TODO: get better this page

class `aiosonic.HTTPClient` (*connector: aiosonic.connectors.TCPConnector = None*)
aiosonic.HTTPClient class.

This class holds the client creation that will be used for requests.

async `aiosonic.HTTPClient.request` (*self, url: str, method: str = 'GET', headers: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, params: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, data: Union[str, bytes, dict, tuple, AsyncIterator[bytes], Iterator[bytes]] = None, multipart: bool = False, verify: bool = True, ssl: ssl.SSLContext = None, timeouts: aiosonic.timeout.Timeouts = None, follow: bool = False, http2: bool = False*) \rightarrow *aiosonic.HttpResponse*

Do http request.

Params:

- **url:** url of request
- **method:** Http method of request
- **headers:** headers to add in request
- **params:** query params to add in request if not manually added
- **data:** Data to be sent, this param is ignored for get requests.
- **multipart:** Tell aiosonic if request is multipart
- **verify:** parameter to indicate whether to verify ssl
- **ssl:** this parameter allows to specify a custom ssl context
- **timeouts:** parameter to indicate timeouts for request
- **follow:** parameter to indicate whether to follow redirects
- **http2:** flag to indicate whether to use http2 (experimental)

async `aiosonic.HTTPClient.get` (*self, url: str, headers: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, params: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, verify: bool = True, ssl: ssl.SSLContext = None, timeouts: aiosonic.timeout.Timeouts = None, follow: bool = False, http2: bool = False*) \rightarrow *aiosonic.HttpResponse*

Do get http request.

async `aiosonic.HTTPClient.post` (*self, url: str, data: Union[str, bytes, dict, tuple, AsyncIterator[bytes], Iterator[bytes]] = None, headers: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, json: dict = None, params: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, json_serializer=<function dumps>, multipart: bool = False, verify: bool = True, ssl: ssl.SSLContext = None, timeouts: aiosonic.timeout.Timeouts = None, follow: bool = False, http2: bool = False*) \rightarrow *aiosonic.HttpResponse*

Do post http request.

async `aiosonic.HTTPClient.put` (*self*, *url*: str, *data*: Union[str, bytes, dict, tuple, AsyncIterator[bytes], Iterator[bytes]] = None, *headers*: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, *json*: dict = None, *params*: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, *json_serializer*=<function dumps>, *multipart*: bool = False, *verify*: bool = True, *ssl*: ssl.SSLContext = None, *timeouts*: aiosonic.timeout.Timeouts = None, *follow*: bool = False, *http2*: bool = False) → *aiosonic.HttpResponse*

Do put http request.

async `aiosonic.HTTPClient.patch` (*self*, *url*: str, *data*: Union[str, bytes, dict, tuple, AsyncIterator[bytes], Iterator[bytes]] = None, *headers*: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, *json*: dict = None, *params*: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, *json_serializer*=<function dumps>, *multipart*: bool = False, *verify*: bool = True, *ssl*: ssl.SSLContext = None, *timeouts*: aiosonic.timeout.Timeouts = None, *follow*: bool = False, *http2*: bool = False) → *aiosonic.HttpResponse*

Do patch http request.

async `aiosonic.HTTPClient.delete` (*self*, *url*: str, *data*: Union[str, bytes, dict, tuple, AsyncIterator[bytes], Iterator[bytes]] = b'', *headers*: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders] = None, *json*: dict = None, *params*: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, *json_serializer*=<function dumps>, *multipart*: bool = False, *verify*: bool = True, *ssl*: ssl.SSLContext = None, *timeouts*: aiosonic.timeout.Timeouts = None, *follow*: bool = False, *http2*: bool = False) → *aiosonic.HttpResponse*

Do delete http request.

async `aiosonic.HTTPClient.wait_requests` (*self*, *timeout*: int = 30)

Wait until all pending requests are done.

If timeout, returns false.

This is useful when doing safe shutdown of a process.

7.2.1 Classes

class `aiosonic.HttpHeaders` (*data*=None, ***kwargs*)

Http headers dict.

class `aiosonic.HttpResponse`

Custom HttpResponse class for handling responses.

Properties:

- `status_code` (int): response status code
- `headers` (HttpHeaders): headers in case insensitive dict
- `raw_headers` (List[Tuple[bytes, bytes]]): headers as raw format

async `content` () → bytes

Read response body.

async `json` (*json_decoder*=<function loads>) → dict

Read response body.

read_chunks () → AsyncIterator[bytes]
Read chunks from chunked response.

property status_code
Get status code.

async text () → str
Read response body.

class aiosonic.timeout.**Timeouts** (*sock_connect: Optional[float] = 5, sock_read: Optional[float] = 30, pool_acquire: Optional[float] = None, request_timeout: Optional[float] = 60*)
Timeouts class wrapper.

7.2.2 Types

aiosonic.DataType = **typing.Union**[**str**, **bytes**, **dict**, **tuple**, **typing.AsyncIterator**[**bytes**], **typing.List**[**typing.AsyncIterator**[**bytes**]]
The central part of internal API.

This represents a generic version of type ‘origin’ with type arguments ‘params’. There are two kind of these aliases: user defined and special. The special ones are wrappers around builtin collections and ABCs in `collections.abc`. These must have ‘name’ always set. If ‘inst’ is False, then the alias can’t be instantiated, this is used by e.g. `typing.List` and `typing.Dict`.

aiosonic.HeadersType = **typing.Union**[**typing.Dict**[**str**, **str**], **typing.List**[**typing.Tuple**[**str**, **str**]]
Headers

INDEX

C

`content()` (*aiosonic.HttpResponse method*), 17

D

`DataType` (*in module aiosonic*), 18

`delete()` (*in module aiosonic.HTTPClient*), 17

G

`get()` (*in module aiosonic.HTTPClient*), 16

H

`HeadersType` (*in module aiosonic*), 18

`HTTPClient` (*class in aiosonic*), 16

`HttpHeaders` (*class in aiosonic*), 17

`HttpResponse` (*class in aiosonic*), 17

J

`json()` (*aiosonic.HttpResponse method*), 17

P

`patch()` (*in module aiosonic.HTTPClient*), 17

`post()` (*in module aiosonic.HTTPClient*), 16

`put()` (*in module aiosonic.HTTPClient*), 16

R

`read_chunks()` (*aiosonic.HttpResponse method*), 17

`request()` (*in module aiosonic.HTTPClient*), 16

S

`status_code()` (*aiosonic.HttpResponse property*), 18

T

`text()` (*aiosonic.HttpResponse method*), 18

`Timeouts` (*class in aiosonic.timeout*), 18

W

`wait_requests()` (*in module aiosonic.HTTPClient*),
17