
aio sonic

Release 0.6.0

Johanderson Mogollon

May 08, 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	15
	Index	19

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

Current version is 0.6.0.

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

    url = "https://postman-echo.com/post"
    posted_data = {'foo': 'bar'}

    # post data as multipart form
    response = await aiohttp.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 aiohttp.post(url, json=posted_data)

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

    # Sample get request + timeout
    from aiohttp.timeout import Timeouts
    timeouts = Timeouts(
        sock_read=10,
        sock_connect=3
    )
    response = await aiohttp.get('https://www.google.com/', timeouts=timeouts)
    assert response.status_code == 200
    assert 'Google' in (await response.text())

    print('success')

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

(continues on next page)

(continued from previous page)

```
loop.run_until_complete(run())
```

BENCHMARKS

The numbers speak for themselves

```
$ python ./tests/performance.py
doing tests...
httpx did break with: [Errno 24] Too many open files
{
  "aiosonic": "1000 requests in 146.89 ms",
  "aiosonic cyclic": "1000 requests in 146.17 ms",
  "aiohttp": "1000 requests in 319.42 ms",
  "requests": "1000 requests in 1766.96 ms"
}
aiosonic is 117.46% faster than aiohttp
aiosonic is 1102.93% faster than requests
aiosonic is -0.49% faster than aiosonic cyclic
```


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

7.2 Reference

TODO: get better this page

7.2.1 Methods

async `aiosonic.request` (*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, connector: aiosonic.connectors.TCPConnector = None, multipart: bool = False, verify: bool = True, ssl: ssl.SSLContext = None, timeouts: aiosonic.timeout.Timeouts = None, follow: bool = False, http2: bool = False) → *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.
- **connector:** TCPConnector to be used if provided
- **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.get (url: str, headers: Union[Dict[str, str], List[Tuple[str, str]], aiosonic.HttpHeaders]
    = None, params: Union[Dict[str, str], Sequence[Tuple[str, str]]] = None, connector: aiosonic.connectors.TCPConnector = None, verify: bool = True, ssl:
    ssl.SSLContext = None, timeouts: aiosonic.timeout.Timeouts = None, follow: bool
    = False, http2: bool = False) → aiosonic.HttpResponse
```

Do get http request.

```
async aiosonic.post (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, connector:
    aiosonic.connectors.TCPConnector = 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 post http request.

```
async aiosonic.put (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, connector: aiosonic.connectors.TCPConnector
    = 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.patch (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, connector:
    aiosonic.connectors.TCPConnector = 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.delete (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, connector:
    aiosonic.connectors.TCPConnector = 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.

7.2.2 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.

7.2.3 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*), 17

`delete()` (*in module aiosonic*), 16

G

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

H

`HeadersType` (*in module aiosonic*), 17

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

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

J

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

P

`patch()` (*in module aiosonic*), 16

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

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

R

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

`request()` (*in module aiosonic*), 15

S

`status_code()` (*aiosonic.HttpResponse property*), 17

T

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

`Timeouts` (*class in aiosonic.timeout*), 17