aiosonic

Release 0.4.1

Johanderson Mogollon

Sep 09, 2019

CONTENTS

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

Current version is 0.4.1.

Repo is hosted at Github.

CHAPTER

ONE

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.

CHAPTER

TWO

REQUIREMENTS

- Python>=3.6

CHAPTER

THREE

INSTALL

```
$ pip install aiosonic
```

CHAPTER

FOUR

GETTING STARTED

```
import asyncio
import aiosonic
import json


async def run():
    """Start."""
    # Sample get request
    response = await aiosonic.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 aiosonic.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 aiosonic.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 aiosonic.timeout import Timeouts
    timeouts = Timeouts(
        sock_read=10,
        sock_connect=3
    )
    response = await aiosonic.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())
```

CHAPTER

FIVE

BENCHMARKS

The numbers speak for themselves

```
$ python tests/performance.py
doing tests...
{
 "aiohttp": "1000 requests in 247.47 ms",
 "requests": "1000 requests in 3625.10 ms",
 "aiosonic": "1000 requests in 80.09 ms",
 "aiosonic cyclic": "1000 requests in 128.71 ms",
 "httpx": "1000 requests in 528.73 ms"
}
aiosonic is 209.00% faster than aiohttp
aiosonic is 4426.34% faster than requests
aiosonic is 60.70% faster than aiosonic cyclic
aiosonic is 560.17% faster than httpx
```

CHAPTER

SIX

CONTRIBUTING

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

Thanks :)

CHAPTER

SEVEN

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], 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*) → 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 wether to follow redirects

async aiosonic.**get**(*url: str, headers: Union[Dict[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*) → 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], 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*) → 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], 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*) → 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], 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*) → 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], 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*) → 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 (Sequence[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: int = 5*, *sock_read: int = 60*, *pool_acquire: int = None*, *request_timeout: int = None*)
Timeouts class wrapper.

7.2.3 Types

aiosonic.**DataType = typing.Union[str, bytes, dict, tuple, typing.AsyncIterator[bytes], typ**
Data to be sent in requests, allowed types

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

INDEX