aiothrift Documentation

Release 0.2.1

moonshadow

Dec 14, 2019

Contents

asyncio (**PEP 3156**) Thrift client and server library.

CHAPTER 1

Installation

The easiest way to install aiothrift is by using the packae on Pypi:

```
pip install aiothrift
```

CHAPTER 2

Requirements

- Python 3.4 +

CHAPTER 3

Contribute

- Issue Tracker: https://github.com/moonshadow/aiothrift/issues
- Source Code: https://github.com/moonshadow/aiothrift

Feel free to file an issue or make pull request if you find any bugs or have some suggestions for library improvement.

CHAPTER 4

User's Guide

4.1 Quickstart

This page gives you a good introduction to aiothrift. It assumes you already have aiothrift installed.

4.1.1 A Minimal Application

At first you should have a thrift file which defines at least one service. Go to create a thrift file named `pingpong.thrift`:

```
service PingPong {
    string ping(),
    i32 add(1:i32 a, 2:i32 b),
}
```

Now you can fire an asyncio thrift server easily:

```
import asyncio
import aiothrift

pingpong_thrift = aiothrift.load('pingpong.thrift', module_name='pingpong_thrift')

class Dispatcher:
    def ping(self):
        return "pong"

    async def add(self, a, b):
        await asyncio.sleep(1)
        return a + b

async def main():
  server = await aiothrift.create_server(pingpong_thrift.PingPong, Dispatcher())
  async with server:
```

(continues on next page)

(continued from previous page)

```
        await server.serve_forever()

asyncio.run(main())
```

let's have a look at what the code above does.

1. First we parse a thrift file to a valid python module, thanks for the great job done by *thriftpy2*, we don't have to generate thrift python sdk files manually.

2. We create a *Dispatcher* class as the namespace for all thrift rpc functions. Here we define a *ping* method which corresponds to the *ping* function defined in `pingpong.thrift`. You may notice that the *add* method is actually a coroutine but a normal one. if you define the rpc function as a coroutine, it would scheduled by our thrift server and send the result back to client after the coroutine task is completed.

3. We then create the server by using `create_server()` function, and it returns a coroutine instance which can be scheduled by the event loop later.

   4. Lastly we call `asyncio.run(main())` to run the event loop to schedule the server task.

Just save it as `server.py` and then you can start the thrift server:

```
$ python3 server.py
```

It will listening at *localhost:6000* by default.

Now you'd like to visit the thrift server through a thrift client:

```
   import asyncio
   import aiothrift

   pingpong_thrift = aiothrift.load('pingpong.thrift', module_name='pingpong_thrift')

   async def go():
       client = await aiothrift.create_pool(pingpong_thrift.PingPong)
       print(await client.ping())
       print(await client.add(5, 6))
       client.close()
       await client.wait_closed()

   asyncio.run(go())


Save it as :file:`client.py`, and run the client by::

   $ python client.py
    * pong
```

That's all you need to make a minimal thrift application on both the server and client side, I hope you will enjoy it.

4.2 Examples of aiothrift usage

4.2.1 sample thrift file

`get source code`

```
service PingPong {
    string ping(),

    i64 add(1:i32 a, 2:i64 b),
}
```

4.2.2 aio thrift server

get source code

```python
import asyncio
import aiothrift

pingpong_thrift = aiothrift.load("pingpong.thrift", module_name="pingpong_thrift")


class Dispatcher:
    def ping(self):
        return "pong"

    async def add(self, a, b):
        await asyncio.sleep(2)
        return a + b


async def main():
    server = await aiothrift.create_server(pingpong_thrift.PingPong, Dispatcher())
    async with server:
        print("server is listening on host {} and port {}".format("127.0.0.1", 6000))
        await server.serve_forever()


if __name__ == "__main__":
    asyncio.run(main())
```

4.2.3 aio thrift client

get source code

```python
import asyncio
import aiothrift

pingpong_thrift = aiothrift.load("pingpong.thrift", module_name="pingpong_thrift")


async def create_connection():
    conn = await aiothrift.create_connection(
        pingpong_thrift.PingPong, ("127.0.0.1", 6000), timeout=10
    )
    print(await conn.ping())
    print(await conn.add(5, 6))
    conn.close()
```

(continues on next page)

(continued from previous page)

```
asyncio.run(create_connection())
```

4.2.4 connection pool sample

get source code

```
import aiothrift
import asyncio

pingpong_thrift = aiothrift.load("pingpong.thrift", module_name="pingpong_thrift")


async def create_pool():
    return await aiothrift.create_pool(
        pingpong_thrift.PingPong, ("127.0.0.1", 6000), timeout=3
    )


async def main(pool):
    print(await pool.ping())
    print(await pool.add(5, 6))
    print(await pool.ping())


if __name__ == "__main__":

    async def f():
        pool = await create_pool()
        await asyncio.gather(main(pool), main(pool))
        pool.close()
        await pool.wait_closed()

    asyncio.run(f())
```

4.3 API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

4.3.1 API

This part of the documentation covers all the interfaces of aiothrift. For parts where aiothrift depends on external libraries, we document the most important right here and provide links to the canonical documentation.

ThriftConnection Object

ThriftConnection Pool

protocol

processor

server

exceptions

Useful functions

4.4 Additional Notes

Information and changelog are here for the interested.

4.4.1 LICENSE

4.4.2 Changelog

Here you can see the full list of changes between each `aiothrift` release.

Version 0.2

Now it works with Python 3.7 and higher.

Version 0.1

First public release.

Python Module Index

Index