simple-websocket

Miguel Grinberg

Oct 13, 2024

CONTENTS

Simple WebSocket server and client for Python.

CHAPTER

ONE

GETTING STARTED

`simple-websocket` includes a collection of WebSocket servers and clients for Python, including support for both traditional and asynchronous (asyncio) workflows. The servers are designed to be integrated into larger web applications if desired.

1.1 Installation

This package is installed with `pip`:

```
pip install simple-websocket
```

1.2 Server Example #1: Flask

The following example shows how to add a WebSocket route to a Flask application.

```
from flask import Flask, request
from simple_websocket import Server, ConnectionClosed

app = Flask(__name__)

@app.route('/echo', websocket=True)
def echo():
    ws = Server.accept(request.environ)
    try:
        while True:
            data = ws.receive()
            ws.send(data)
    except ConnectionClosed:
        pass
    return ''
```

Integration with web applications using other WSGI frameworks works in a similar way. The only requirement is to pass the `environ` dictionary to the `Server.accept()` method to initiate the WebSocket handshake.

1.3 Server Example #2: Aiohttp

The following example shows how to add a WebSocket route to a web application built with the aiohttp framework.

```
from aiohttp import web
from simple_websocket import AioServer, ConnectionClosed

app = web.Application()

async def echo(request):
    ws = await AioServer.accept(aiohttp=request)
    try:
        while True:
            data = await ws.receive()
            await ws.send(data)
    except ConnectionClosed:
        pass
    return web.Response(text='')

app.add_routes([web.get('/echo', echo)])

if __name__ == '__main__':
    web.run_app(app, port=5000)
```

1.4 Server Example #3: ASGI

The next server example shows an asynchronous application that supports the ASGI protocol.

```
from simple_websocket import AioServer, ConnectionClosed

async def echo(scope, receive, send):
    ws = await AioServer.accept(asgi=(scope, receive, send))
    try:
        while True:
            data = await ws.receive()
            await ws.send(data)
    except ConnectionClosed:
        pass
```

1.5 Client Example #1: Synchronous

The client example that follows can connect to any of the server examples above using a synchronous interface.

```
from simple_websocket import Client, ConnectionClosed

def main():
    ws = Client.connect('ws://localhost:5000/echo')
    try:
        while True:
```

(continues on next page)

(continued from previous page)

```
            data = input('> ')
            ws.send(data)
            data = ws.receive()
            print(f'< {data}')
    except (KeyboardInterrupt, EOFError, ConnectionClosed):
        ws.close()

if __name__ == '__main__':
    main()
```

1.6 Client Example #2: Asynchronous

The next client uses Python's `asyncio` framework.

```
import asyncio
from simple_websocket import AioClient, ConnectionClosed

async def main():
    ws = await AioClient.connect('ws://localhost:5000/echo')
    try:
        while True:
            data = input('> ')
            await ws.send(data)
            data = await ws.receive()
            print(f'< {data}')
    except (KeyboardInterrupt, EOFError, ConnectionClosed):
        await ws.close()

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

CHAPTER

TWO

API REFERENCE

2.1 The Server class

class simple_websocket.**Server**(*environ*, *subprotocols=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *thread_class=None*, *event_class=None*, *selector_class=None*)

This class implements a WebSocket server.

Instead of creating an instance of this class directly, use the `accept()` class method to create individual instances of the server, each bound to a client request.

classmethod **accept**(*environ*, *subprotocols=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *thread_class=None*, *event_class=None*, *selector_class=None*)

Accept a WebSocket connection from a client.

Parameters

- **environ** – A WSGI `environ` dictionary with the request details. Among other things, this class expects to find the low-level network socket for the connection somewhere in this dictionary. Since the WSGI specification does not cover where or how to store this socket, each web server does this in its own different way. Werkzeug, Gunicorn, Eventlet and Gevent are the only web servers that are currently supported.
- **subprotocols** – A list of supported subprotocols, or `None` (the default) to disable subprotocol negotiation.
- **receive_bytes** – The size of the receive buffer, in bytes. The default is 4096.
- **ping_interval** – Send ping packets to clients at the requested interval in seconds. Set to `None` (the default) to disable ping/pong logic. Enable to prevent disconnections when the line is idle for a certain amount of time, or to detect unresponsive clients and disconnect them. A recommended interval is 25 seconds.
- **max_message_size** – The maximum size allowed for a message, in bytes, or `None` for no limit. The default is `None`.
- **thread_class** – The `Thread` class to use when creating background threads. The default is the `threading.Thread` class from the Python standard library.
- **event_class** – The `Event` class to use when creating event objects. The default is the *threading.Event* ` class from the Python standard library.
- **selector_class** – The `Selector` class to use when creating selectors. The default is the `selectors.DefaultSelector` class from the Python standard library.

`choose_subprotocol`(*request*)

Choose a subprotocol to use for the WebSocket connection.

The default implementation selects the first protocol requested by the client that is accepted by the server. Subclasses can override this method to implement a different subprotocol negotiation algorithm.

Parameters
request – A `Request` object.

The method should return the subprotocol to use, or `None` if no subprotocol is chosen.

`close`(*reason=None*, *message=None*)

Close the WebSocket connection.

Parameters

- **reason** – A numeric status code indicating the reason of the closure, as defined by the WebSocket specification. The default is 1000 (normal closure).
- **message** – A text message to be sent to the other side.

`receive`(*timeout=None*)

Receive data over the WebSocket connection.

Parameters
timeout – Amount of time to wait for the data, in seconds. Set to `None` (the default) to wait indefinitely. Set to 0 to read without blocking.

The data received is returned, as `bytes` or `str`, depending on the type of the incoming message.

`send`(*data*)

Send data over the WebSocket connection.

Parameters
data – The data to send. If `data` is of type `bytes`, then a binary message is sent. Else, the message is sent in text format.

`subprotocol`

The name of the subprotocol chosen for the WebSocket connection.

2.2 The `AioServer` class

`class simple_websocket.AioServer`(*request*, *subprotocols=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*)

This class implements a WebSocket server.

Instead of creating an instance of this class directly, use the `accept()` class method to create individual instances of the server, each bound to a client request.

`async classmethod accept`(*aiohttp=None*, *asgi=None*, *sock=None*, *headers=None*, *subprotocols=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*)

Accept a WebSocket connection from a client.

Parameters

- **aiohttp** – The request object from aiohttp. If this argument is provided, `asgi`, `sock` and `headers` must not be set.

- **asgi** – A (scope, receive, send) tuple from an ASGI request. If this argument is provided, `aiohttp`, `sock` and `headers` must not be set.
- **sock** – A connected socket to use. If this argument is provided, `aiohttp` and `asgi` must not be set. The `headers` argument must be set with the incoming request headers.
- **headers** – A dictionary with the incoming request headers, when `sock` is used.
- **subprotocols** – A list of supported subprotocols, or `None` (the default) to disable subprotocol negotiation.
- **receive_bytes** – The size of the receive buffer, in bytes. The default is 4096.
- **ping_interval** – Send ping packets to clients at the requested interval in seconds. Set to `None` (the default) to disable ping/pong logic. Enable to prevent disconnections when the line is idle for a certain amount of time, or to detect unresponsive clients and disconnect them. A recommended interval is 25 seconds.
- **max_message_size** – The maximum size allowed for a message, in bytes, or `None` for no limit. The default is `None`.

choose_subprotocol(*request*)

Choose a subprotocol to use for the WebSocket connection.

The default implementation selects the first protocol requested by the client that is accepted by the server. Subclasses can override this method to implement a different subprotocol negotiation algorithm.

Parameters
request – A `Request` object.

The method should return the subprotocol to use, or `None` if no subprotocol is chosen.

async close(*reason=None*, *message=None*)

Close the WebSocket connection.

Parameters

- **reason** – A numeric status code indicating the reason of the closure, as defined by the WebSocket specification. The default is 1000 (normal closure).
- **message** – A text message to be sent to the other side.

async receive(*timeout=None*)

Receive data over the WebSocket connection.

Parameters
timeout – Amount of time to wait for the data, in seconds. Set to `None` (the default) to wait indefinitely. Set to 0 to read without blocking.

The data received is returned, as `bytes` or `str`, depending on the type of the incoming message.

async send(*data*)

Send data over the WebSocket connection.

Parameters
data – The data to send. If `data` is of type `bytes`, then a binary message is sent. Else, the message is sent in text format.

subprotocol

The name of the subprotocol chosen for the WebSocket connection.

2.3 The `Client` class

`class simple_websocket.Client(`*url*, *subprotocols=None*, *headers=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *ssl_context=None*, *thread_class=None*, *event_class=None*`)`

This class implements a WebSocket client.

Instead of creating an instance of this class directly, use the `connect()` class method to create an instance that is connected to a server.

`receive(`*timeout=None*`)`

Receive data over the WebSocket connection.

Parameters

timeout – Amount of time to wait for the data, in seconds. Set to `None` (the default) to wait indefinitely. Set to 0 to read without blocking.

The data received is returned, as `bytes` or `str`, depending on the type of the incoming message.

`send(`*data*`)`

Send data over the WebSocket connection.

Parameters

data – The data to send. If `data` is of type `bytes`, then a binary message is sent. Else, the message is sent in text format.

`subprotocol`

The name of the subprotocol chosen for the WebSocket connection.

`classmethod connect(`*url*, *subprotocols=None*, *headers=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *ssl_context=None*, *thread_class=None*, *event_class=None*`)`

Returns a WebSocket client connection.

Parameters

- **url** – The connection URL. Both `ws://` and `wss://` URLs are accepted.
- **subprotocols** – The name of the subprotocol to use, or a list of subprotocol names in order of preference. Set to `None` (the default) to not use a subprotocol.
- **headers** – A dictionary or list of tuples with additional HTTP headers to send with the connection request. Note that custom headers are not supported by the WebSocket protocol, so the use of this parameter is not recommended.
- **receive_bytes** – The size of the receive buffer, in bytes. The default is 4096.
- **ping_interval** – Send ping packets to the server at the requested interval in seconds. Set to `None` (the default) to disable ping/pong logic. Enable to prevent disconnections when the line is idle for a certain amount of time, or to detect an unresponsive server and disconnect. A recommended interval is 25 seconds. In general it is preferred to enable ping/pong on the server, and let the client respond with pong (which it does regardless of this setting).
- **max_message_size** – The maximum size allowed for a message, in bytes, or `None` for no limit. The default is `None`.
- **ssl_context** – An `SSLContext` instance, if a default SSL context isn't sufficient.
- **thread_class** – The `Thread` class to use when creating background threads. The default is the `threading.Thread` class from the Python standard library.

- **event_class** – The `Event` class to use when creating event objects. The default is the *threading.Event`* class from the Python standard library.

`close`(*reason=None*, *message=None*)

Close the WebSocket connection.

Parameters

- **reason** – A numeric status code indicating the reason of the closure, as defined by the WebSocket specification. The default is 1000 (normal closure).
- **message** – A text message to be sent to the other side.

2.4 The `AioClient` class

`class simple_websocket.AioClient`(*url*, *subprotocols=None*, *headers=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *ssl_context=None*)

This class implements a WebSocket client.

Instead of creating an instance of this class directly, use the `connect()` class method to create an instance that is connected to a server.

`async classmethod connect`(*url*, *subprotocols=None*, *headers=None*, *receive_bytes=4096*, *ping_interval=None*, *max_message_size=None*, *ssl_context=None*, *thread_class=None*, *event_class=None*)

Returns a WebSocket client connection.

Parameters

- **url** – The connection URL. Both `ws://` and `wss://` URLs are accepted.
- **subprotocols** – The name of the subprotocol to use, or a list of subprotocol names in order of preference. Set to `None` (the default) to not use a subprotocol.
- **headers** – A dictionary or list of tuples with additional HTTP headers to send with the connection request. Note that custom headers are not supported by the WebSocket protocol, so the use of this parameter is not recommended.
- **receive_bytes** – The size of the receive buffer, in bytes. The default is 4096.
- **ping_interval** – Send ping packets to the server at the requested interval in seconds. Set to `None` (the default) to disable ping/pong logic. Enable to prevent disconnections when the line is idle for a certain amount of time, or to detect an unresponsive server and disconnect. A recommended interval is 25 seconds. In general it is preferred to enable ping/pong on the server, and let the client respond with pong (which it does regardless of this setting).
- **max_message_size** – The maximum size allowed for a message, in bytes, or `None` for no limit. The default is `None`.
- **ssl_context** – An `SSLContext` instance, if a default SSL context isn't sufficient.

`async receive`(*timeout=None*)

Receive data over the WebSocket connection.

Parameters

timeout – Amount of time to wait for the data, in seconds. Set to `None` (the default) to wait indefinitely. Set to 0 to read without blocking.

The data received is returned, as `bytes` or `str`, depending on the type of the incoming message.

async send(*data*)

Send data over the WebSocket connection.

Parameters

data – The data to send. If `data` is of type `bytes`, then a binary message is sent. Else, the message is sent in text format.

subprotocol

The name of the subprotocol chosen for the WebSocket connection.

async close(*reason=None*, *message=None*)

Close the WebSocket connection.

Parameters

- **reason** – A numeric status code indicating the reason of the closure, as defined by the WebSocket specification. The default is 1000 (normal closure).
- **message** – A text message to be sent to the other side.

2.5 Exceptions

class simple_websocket.**ConnectionError**(*status_code=None*)

Connection error exception class.

class simple_websocket.**ConnectionClosed**(*reason=CloseReason.NO_STATUS_RCVD*, *message=None*)

Connection closed exception class.

- search

INDEX