[Flask SocketIO TCP Connection]

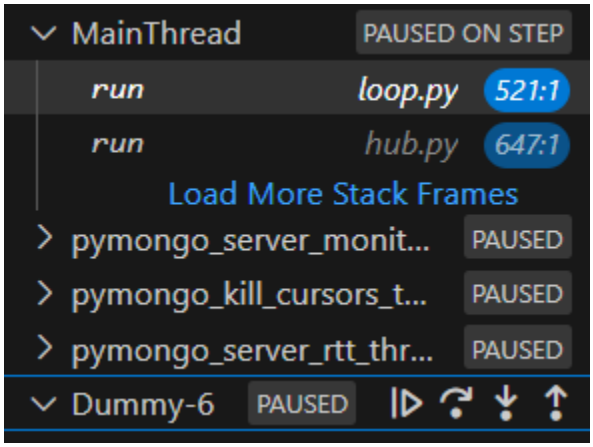
General Information & Licensing

Code Repository	https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py
License Type	MIT License
License Description	• Permits redistribution and use with/without modification • Permits distribution under different sublicense • Does not require distribution of source code
License Restrictions	• Redistribution must preserve copyright and license notice • No forms of liability or warranty responsibility is implied by the author

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≫

Before Asynchronous Watcher Loop:

After Asynchronous Watcher Loop:



The flask_socketio library can choose to use eventlet, gevent, and werkzeug libraries to handle and parse requests. In our server, the flask_socketio library will be using the gevent for the WSGIServer.

https://github.com/keejoonam/CSE312-Team-Project/blob/main/run.py#L323
app_ws.run() refers to the SocketIO class's run function. Its purpose is to start the server, we have passed it the flask instance *app*, a host address for the server, and the port to listen on.

https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py#L553
run() is responsible for determining the type of server to be run and will call the corresponding serve_forever() function to create the listening loop for incoming connections.
run() will first perform various defaulting and defining variables. It will check or default host to '127.0.0.1', check if the listening port is implicitly given as the flask instance's config dictionary or default to port 5000.
After initializing, conditionals determine the type of server is to be used. In our case, it will be gevent and the function will instantiate a pywsgi.WSGIServer object and store it inside the current (self)SocketIO object's wsgi_server constructor.
Finally, the pywsgi.WSGIServer's serve_forever() function is called to proceed onto the listening loop.

https://github.com/bkad/gevent/blob/master/gevent/baseserver.py#L278
serve_forever() checks to see if the server has been started yet, it does this via calling the started function, which will return false here and prompt the start() function to be called.

https://github.com/bkad/gevent/blob/master/gevent/baseserver.py#L228
start() is responsible for beginning the chain of calls to spawn in a listening socket. It starts by calling init_socket, which in this case would be WSGIServer.init_socket()

https://github.com/bkad/gevent/blob/master/gevent/pywsgi.py#L637
WSGIServer.init_socket() will directly call the StreamServer.init_socket and pass down the server instance.

https://github.com/bkad/gevent/blob/master/gevent/server.py#L76

StreamServer.init_socket() first checks to see if the server instance has a socket attribute, since it doesn't, get_listener() is called to create a listening tcp socket.

https://github.com/bkad/gevent/blob/master/gevent/server.py#L86
get_listener() calls the object _tcp_listener(), and returns its result, which will be a listening tcp server

https://github.com/bkad/gevent/blob/master/gevent/server.py#L147
_tcp_listener() will create a socket, bind it to the server address, and set it to listen for connections. It will then return the socket instance.

https://github.com/bkad/gevent/blob/master/gevent/server.py#L89
get_listener() will then return the returned socket instance

https://github.com/bkad/gevent/blob/master/gevent/server.py#L78
Back in init_socket(), the function will take the returned listening socket, and do a couple more configuration, ultimately returning back to WSGIServer.init_socket()

https://github.com/bkad/gevent/blob/master/gevent/pywsgi.py#L639
update_environ() is called to default and configure various WSGI environment values. It will also call getfqdn() to obtain a valid domain name.

https://github.com/gevent/gevent/blob/master/src/gevent/_socketcommon.py#L290
getfqdn() will call gethostbyaddr() to obtain a list of valid aliases for the host name given.

https://github.com/gevent/gevent/blob/master/src/gevent/_socketcommon.py#L267
gethostbyaddr() will simply call and return the name, aliaslist, and address list from gevent._hub_local.py's get_hub().resolver.gethostbyaddr() function.

https://github.com/gevent/gevent/blob/master/src/gevent/resolver/thread.py#L65
the resolver object's gethostbyaddr() function will make a call to threadpool.apply(). This will spawn in a thread that will act as the listening and accept loop for our socket.

https://github.com/gevent/gevent/blob/master/src/gevent/pool.py#L140
apply() will call threadpool.spawn() to start a thread that has the purpose of watching until a request is received. After getting a request, it will asynchronously handle it.

https://github.com/gevent/gevent/blob/master/src/gevent/threadpool.py#L501
spawn() will obtain a semaphore, and create a task_queue. It will allocate and start an asynchronous watcher that serves as the listen loop.

After apply() returns, the greenlet coroutine will begin its work and call the run() object under the hub.py.Hub class.

https://github.com/gevent/gevent/blob/master/src/gevent/hub.py#L631
run() will create an infinite while true loop that will also call the loop.run() function which will be the asynchronous watcher that was created earlier.

https://github.com/gevent/gevent/blob/master/src/gevent/libuv/loop.py#L518
loop.run() will now loop infinitely until request is received. When a task is received,

__run_callbacks() will be used to handle it. Call back will also be used to continue and return the ongoing functions prior to coroutine starting.