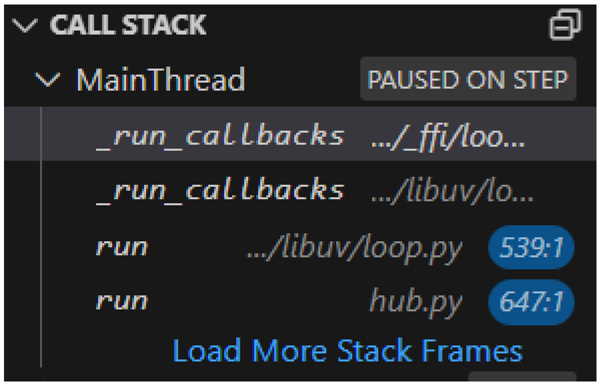
[Flask SocketIO HTTP Header Parser]

General Information & Licensing

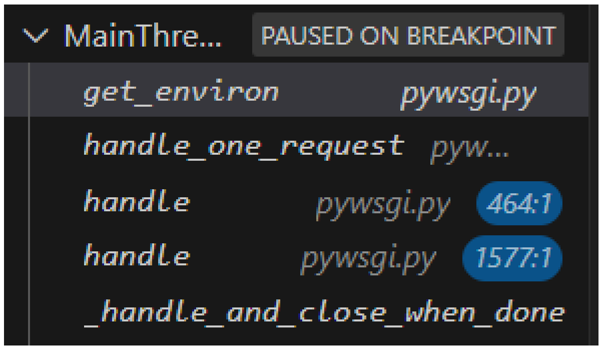
Code Repository	https://github.com/miguelgrinberg/Flask-SocketIO/blob/main/src/flask_socketio/__init__.py
License Type	MIT License
License Description	● Permits redistribution and use with/without modification ● Permits distribution under different sublicense ● Does not require distribution of source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

Magic ⋆★˚｡⋆.˚☽˚｡⋆★⋆



```
CALL STACK
  MainThread                PAUSED ON STEP
    _run_callbacks   .../_ffi/loo...
    _run_callbacks   .../libuv/lo...
    run              .../libuv/loop.py   539:1
    run              hub.py              647:1
             Load More Stack Frames
```

After coroutine change:



```
MainThre...   PAUSED ON BREAKPOINT
  get_environ                pywsgi.py
  handle_one_request   pyw...
  handle               pywsgi.py   464:1
  handle               pywsgi.py   1577:1
  _handle_and_close_when_done
```

The header parsing begins from within the run() loop of the gevent loop.py. When the watcher receives a request, the request will be added to the callback queue for run_callback to process.

https://github.com/gevent/gevent/blob/master/src/gevent/libuv/loop.py#L539
in run(), the greenlet event loop will constantly check and process tasks. In the case of parsing HTTP headers, the function will call _run_callbacks()

https://github.com/gevent/gevent/blob/master/src/gevent/libuv/loop.py#L197
_run_callbacks() will first perform a couple exception catching such as stops or aborts. It will call super._run_callbacks() to actually perform the task.

https://github.com/gevent/gevent/blob/master/src/gevent/_ffi/loop.py#L474
_run_callbacks() will first perform timer update, followed by popping a the left most callback function, and will run it using the arguments passed along with it using callback(*args)

In this case, callback(*args) will allow the coroutine to call gevent baseserver.py's _handle_and_close_when_done() to begin the actual request processing.

https://github.com/gevent/gevent/blob/master/src/gevent/baseserver.py#L32
_handle_and_close_when_done() will call handle(). For our case, this handle method references the gevent.pywsgi.py WSGIServer class' handle() method.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L1569
handle() will create a WSGIHandler instance and pass it constructor values for the socket, address, and the server instance. Then it will call the WSGIHandler's handle() function.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L450
handle() will perform a handle loop until the accept socket is out of requests. For each request, WSGIHandler's handle_one_request() function is used.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L613
handle_one_request() will use read_requestline() to read the request line, as well as read_request() to parse the actual headers.

The function first checks to see if the file reading buffer is open, returning if closed. it will then call read_requestline().

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L599
read_requestline() will rfile.readline() to read the request line via the BufferedReader and return the resulting string back to handle_one_request()

Back in handle_run_request(), it will check to see if the request line is valid, and then run self.read_request().

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L503
read_request() is responsible for parsing the request. It will first take the request line and remove the trailing \r\n\r\n string. It will then split the string and store it into words.

words is then used to set the command, path, and request_version constructors of the handler instance. Afterwards, MessageClass() is called to read the rest of the headers.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L398
MessageClass() simply returns headers_factory()

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L368
headers_factory() will use http's client.parse_headers() to obtain the remaining header and return the HTTPMessage object.

MessageClass() will return the HTTPMessage object from parse_headers()

Back in read_request(), several conditional is used to validate that the request packet is properly formatted. After doing so, it will return True to handle_one_request()

Back in handle_one_request(), get_environ() is called to perform the header parsing.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L1092
get_environ() first instantiates a dict object env, and begin passing it pre-processed key, value pairs, such as self.command that was set during read_request().

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L1142-L1149
Here the function loops through the return values of _headers(), where _headers() will return a key, value pair for the env dict to store.

https://github.com/gevent/gevent/blob/master/src/gevent/pywsgi.py#L1070
_headers() loops through all the headers obtained during read_request(), it eliminates content type and content length as they have been individually added earlier. It will then determine within the loop if the key is valid and return the key, value pair using yield.

After returning the value using Yield, get_environ()'s loop will add that key, value to the dictionary, and _headers() will proceed onto the next key, value pair, yield it, and so on.