Open-Source Technology Use Report: HTTP Parsing

[Flask]

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD-3-Clause
License Description	• The BSD-3-Clause allows for the redistribution or modification of the flask source code as long as a copyright notice, condition list, and disclaimer accompany the distribution.
License Restrictions	• With this license, users cannot create products with this software then promote/endorse those products using the copyright holder's name or the names of any of its contributors without prior written permission.
Who worked with this?	To be decided

Purpose

Flask allows us to parse the HTTP headers of incoming requests using the @app.route decorator. @app.route(rule="path",methods=["method"]) allows us to execute a particular function whenever we get a request of the "method" type at the specified "path". This removes the need for us to parse the raw HTTP request data coming from the TCP socket as we did in the homeworks. This method is used several times in the app.py file in order to route each of the requests we expect to get. For example, if the server receives a GET request to the homepage at the path "/", @app.route("/", *methods* = ["GET"])handles this request.

Magic

The run method from Flask class does the following to establish a server:

1. It imports the run_simple method form the werkzeug library and calls it here
2. The run_simple() method calls make_server() here.

When the server is set up using the make_server() function, request_handler is none. make_server() returns an instance of BaseWSGI server here which assigns the handler to an instance of the WSGIRequestHandler class since the handler is none here. This handler variable will then tell the server how to handle incoming TCP requests since we call the constructor of the TCPServer class here using handler (this is because WSGIRequestHandler inherits from HTPPServer which inherits from sockerserver.TCPSever here)

The WSGIRequestHandler class inherits from http.server.BaseHTTPRequestHandler here. Since BaseHTTPRequestHandler inherits socketserver.StreamRequestHandler here which inherits from socketserver.BaseRequestHandler here, whenever there is an incoming HTTP request on a TCP socket socketserver.BaseRequestHandler calls self.handle() here. Once this call is made, the following takes place so that the HTTP header is parsed:

1. The socketserver.BaseRequestHandler.handle() method calls the WSGIRequestHandler handle() method which calls the http.server.BaseHTTPRequestHander.handle() method here.
2. http.server.BaseHTTPRequestHander.handle() calls self.handle_one_request() here. handle_one_request() then calls self.parse_request() here.
3. BaseHTTPRequestHandler.parse_request() is where the path and request type are parsed by splitting the raw request line on spaces here then storing the first two elements of the slit into self.path and self.command respectively here.
4. HTTPMessage.parse_headers() is then called here.
5. HTTPMessage.parse_headers() calls HTTPMessage._read_headers() here which reads each header line and appends it to the headers variable here which is then returned
6. All of the header lines that are returned by HTTPMessage._read_headers() are combined into one string here.
7. The parsestr() method of the Parser class is then called here. This method calls self.parse here.
8. Parser.parse() reads the input header string here then passes it into the feed method of the FeedParser class here. feed() then calls self._call_parse() here which calls self._parse() here. self._parse() references the self.parsegen() method so this is called.
9. FeedParser._parsegen() appends each header line to the header list here then calls self._parse_headers on that list here.
10. FeedParser._parse_headers then iterates through each header line here and calls self.policy.header_source_parse() on each header line here.
11. header_source_parse() splits each header line on the first colon here to separate the keys from the values then returns them here.
12. FeedParser._cur.set_raw() is then called on the header key value pair generated by header_source_parse() here. Which assigns the separated headers tuple to self._headers here of a HTTPMessage class.
13. Parser.parse() then calls FeedParser.close() here.
14. FeedParser.close() calls self._pop_message() here. Which returns the HTTPMessage object with the parsed headers by calling self._msgstack.pop() here.
15. FeedParser.close() then return the HTTPMessage object generated by

self._pop_message() here. This allows Parser.parse(), Parser.parsestr(), and HTTPMessage.parse_headers() to return this object here, here, and here. BaseHTTPRequestHandler.headers will therefore hold the parsed headers.