Open-Source Report

Parsing HTTP Headers

Flask

Link to Documentation: https://flask.palletsprojects.com/en/2.0.x/

General Information & Licensing

Code Repository	https://github.com/pallets/flask
License Type	BSD 3-Clause License
License Description	• A permissive license with unlimited freedom of usage. • Can be copied, modified, or distributed. • Can be used commercially.
License Restrictions	• Must include the full text of the license. • Must include the original copyright notice and the specified disclaimer. • Name of copyright holder or contributors may not be used to endorse or promote products derived from use of software without written permission.

Magic

After the TCP connection is made, a BaseWSGIServer is created (refer to TCP connections report) which allows Flask to process requests under a WSGI environment.

In order to parse HTTP headers, Flask uses another library, *werkzeug*. In the *serving.py* file, the make_environ function on line 160 will create an WSGI object that splits the URL by the path, with the functionality coming from importing urllib.parse. The environ object will take in various information ranging from WSGI to the server and convert such into a key-value dictionary.

When an HTTP request is made, Flask will move into *werkzeug*/local.py to collect the content of the request by initializing a LocalProxy object on line 471.

Then, it will call the ".headers" function which creates a Headers object from *werkzeug/*datastructures/headers.py that will return an object of type EnvironHeaders at the end. In the Headers class, the function "__str__" is called on line 461, which will return formatted headers that will be suitable for HTTP transmission. Here's how it's done:

Within the "__str__" function, the request is converted to a list of key-value pairs. To do so, this function needs to create a WSGI list in which it will iterate over.

To create a WSGI list, the function ".to_wsgi_list" is called on line 448 which converts the headers into a list that is suitable for WSGI. Since "__str__" will be iterating over this WSGI list, it will call the "__len__" function on line 546 from the EnvironHeaders class, which is the same as the Headers class but suitable for the WSGI environment. Here, this uses the "__iter__" function called on line 551 which looks for any key within the EnvironHeader object (created by make_environ from before) that starts with "HTTP_" to return a reformatted tuple (key, value) relevant to the HTTP request. The reformatting is done through replacing the underscore with a dash and the values are capitalized properly.

After returning this new WSGI list to the function "__str__", this function will loop through each key value pair within the WSGI list to create another list of key-value pairs in the form of "key: value".

After reformatting, the Flask object will dispatch and finalize the request. Here, a response is made by creating a Response object in *werkzeug* to initialize the response. This function will first check if the response headers passed into the function are of type Headers. Then, for each header within the response headers, it checks for status, content type, and mime type. This is to reformat the headers appropriately.

As an example, when getting content type, the function "get_content_type" from *utils.py* on line 167 is called to get the string containing the full content type with charset for a mime type. This is done through passing the mime type and determining if it is valid by comparing it to a dictionary of mime types listed in the XDG mime info document. After comparing this, the content type is reformatted as "*mimetype*; charset=*charset*" and returned.

Flask parses headers from HTTP requests using multiple functions built in the *http.py* file.

Within *werkzeug/http.py*, there are a series of data structures that contain common header names and values starting from line 28 to 119.

For example, there is a set of entity headers such as “content-length” and “content-type”, as well as a dictionary containing all of the HTTP status codes like “200” and “404”.

Starting from the link above at line 346, each header value is parsed using any of the functions: parse_list_header, parse_dict_header, parse_options_header, parse_accept_header, parse_cache_control_header, parse_csp_header, parse_set_header, parse_authorization_header, parse_www_authenticate_header, parse_if_range_header, parse_range_header, and parse_content_range_header.

For an example of how the functions work, here’s an overview of how parse_list_header and parse_dict_header work.

At line 346, parse_list_header will parse a header value consisting of a list of comma separated items that correspond to “RFC 9110”. Overall, this function will remove quotes from the given set of header values. For example, this function will convert ‘token, “quoted value”’ to [‘token’, ‘quoted value’]. To do this, for each item within this list of values, the quotation marks existing at the first and last index would be spliced, resulting in an unquoted header value.

At line 373, parse_dict_header will parse a list header using the previous function and then continue to parse each item as a key-value pair in a dictionary. For example, this function should convert ‘a=b, c=”d, e”, f’ to {“a”: “b”, “c”: “d, e”, “f”: None}. This is done by splitting each item by “=” using the partition function. This returns a tuple containing the key, value, and a flag for whether the key has a value based on the existence of the equal sign. Then, the function will check for asterisks and quotation marks, handling each accordingly. Using this information, the parse_dict_header function will append the key and value to the dictionary and return such.

After parsing each header individually using the functions, they are all passed into the EnvironHeaders class as a dictionary to create a new object. This is the returned value type after “request.headers” is called. It is important to note that EnvironHeaders and Headers class are essentially the same, with both carrying a dictionary-like format. However, EnvironHeaders is created from a WSGI environment. Within the EnvironHeaders class, the function “getitem” on line 536 will be used to retrieve a value given a key by searching through an environ object. An environ object is initialized through the “init” function. Both functions are linked here at their respective lines.

Overall, parsing HTTP headers within *werkzeug* is largely done through converting the header to be suitable for a WSGI environment, then splicing each header and reformatting the header values. After, the headers are returned as a dictionary-like data structure where key-value pairs can be accessed easily.