1 Project Abstract

The project is to build a web browser that can handle HTTP 1.1 (and more) connections between the client and the server while running basic web pages that is composed of HTML, CSS and JavaScript files.

Note:

- The browser mainly supports http or https scheme. The file scheme is there as a test for scalability and hence, will not be properly explained in this design doc yet.
- The project has a priority of correctness followed by the error handling since the project is still in its early stage.

2 Downloading Web Pages

2.1 url.url.py

This python module handles parsing the url input into various components such as scheme, hostname, port, etc. The design pattern here is that there is a one-to-one correponse between some classes and the schemes the browser supports. For example, HttpURL will assume the url to be an HTTP URL and hence parse the input accordingly. **These classes will be refered to as Components classes from now on.**

The clients of this module should only interact with the url.get_url_components method when creating objects of Components classes. The method reads the scheme from the input string and will call constructors of Components classes. If no scheme is found in the string, the method will assume an http scheme.

Note:

- While Python's urllib.parse could work, the pattern mentioned is more versatile when more support for each scheme is added to the browser.
- The http (or https) url is now limited to components of scheme, host, port and path. If http or https scheme is used, the default port will be 80 or 443 respectively.

2.2 request.request.py

This python module conatains classes like HTTPRequest that can create HTTP messages that are ready to be sent to the servers. The constructors need objects of Components classes. For HTTPRequest class, a subset of headers are supported such as "Connection": "close" and "Accept-Encoding": "gzip" (see more in trasfer.transferutil.py). The client of this module should call get_http_request_text or get_http_request_bytes in order to get fully structured message (or bytes) that are ready to be sent to the server.

2.3 transfer.socketutil.py

Socket Programming is done in this module. The socket connection only supports IPv4 address family. Send will sends a message from request.request.HttpRequest object to the server. The request will be completely sent or the failure of the network will be detected by raising runtime error. The sockte only supports HTTP 1.1. The client who creates an object of Socket is responsible for closing it.

2.4 transfer.transferutil.py

The byte stream from the server socket is used to build Response objects. The response can be "transfer-encoded" with "chunked" and "content-encoded" with "gzip". The reponse object will remember the status line, the headers and the body.

Caching is also implemented here with the Least-Recently-Used policy. Server responses will only be cached if and only if the status is 200 code, the http method used for the request was GET method, and the "cache-control" headers value is not "no-store". Moreover, "max-age" value for "cache-control" is supported.