Multithreaded http server design

In very brief, designed this static file server based of **sun's simple implementation of HttpServer** and customizing server's handleRequest functionality for static file handling.

Using Java's **ExecutorService fixed threadpool** to support multithreading (One could define the pool size using an environment variable depending on where the server is deployed and cpu resources that are shared)

Implemented HTTP functionality for following headers for static file scenarios:

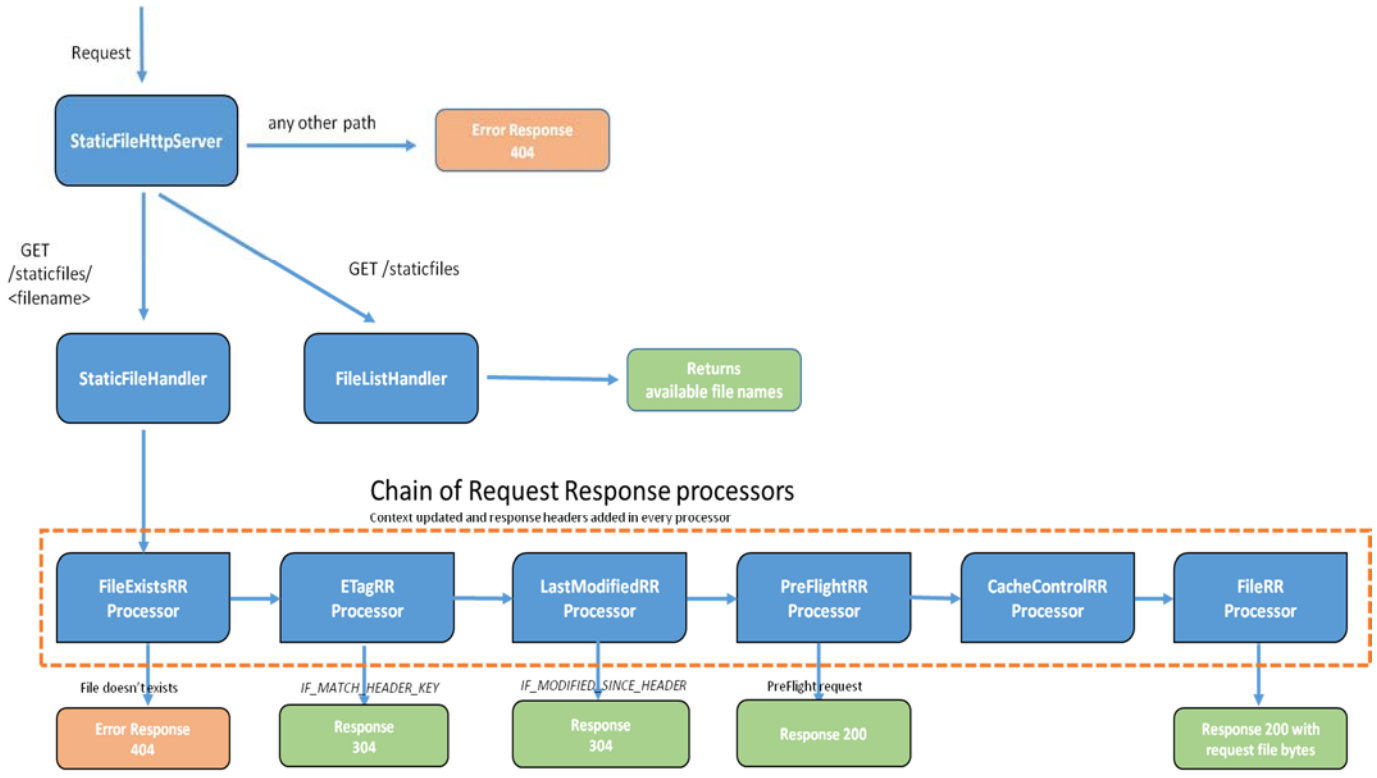
1. Cache-headers
2. E-tags
3. Last-modified
4. Mime-type
5. Http-range

Used chain of responsibility pattern to process a request for each type of header and accordingly updating the processing context. Couldn't put together keep-alive header processor but it should be very easy to add another RequestResponseProcessor to the pipeline.

2 types of Get requests supported:

1] http://localhost:HOST_PORT/staticfiles (very rudimentary needs to be enhanced to return json for filename list

2] http://localhost:HOST_PORT/staticfiles/<filename>



Some logic details on these **request response processors**:

1] FileExists Processor: Checks if file exists, if it does set some basic content-type, content-length, accept-range headers otherwise pipeline is terminated with 404 response code.

2] ETag Processor: Calculates ETag header(hex(modifiedTime)-hex(size)) and also checks if there is an IF_MATCH_HEADER_KEY is provided and matches then we return 304 response code.

3] Last modified Processor: Gets the last modified time and sets it on the response header in the required format. IF_MODIFIED_SINCE_HEADER is part of request header then timestamps are compared to see if file has been modified, if not then return 304 response code.

4] PreFlight processor: Checks if it is a PreFlight request then return with all the response headers we have appended so far in the pipeline with response code 200.

5] CacheControl processor: Appends the Cache control header depending on the max cache age that needs to be configured. Currently using default of 7 days.

6] File processor: This processor is the primary one that supports HTTP Range headers to read the requested file partially and return the results. If Range header is not specified then returns the complete file. This processor should be optimized to check the requested file size and available memory to respond accordingly. Also needs to be enhanced to support multi part request ranges.

6] All the processors implement an interface, are package private and are instantiated using a factory.

How to deploy the server and run:

Docker image for the server (docker hub): pkasliwal/com.namo.dist.server.staticfileserver:1.0

Dockerized the server to make the deployment easy and reliable (Dockerfile available in the codebase)

1] Docker command to run with in-built example static files:

docker run -p HOST_PORT:8500 pkasliwal/com.namo.dist.server.staticfileserver:1.0

2] Docker command for mounting static files to the docker container

docker run -p HOST_PORT:8500 -v ABSOLUTE_FILES_DIR:/usr/src/myserver/files pkasliwal/com.namo.dist.server.staticfileserver:1.0

Test Cases

Have configured some very basic postman test cases which are available at tests/postman directory. They test following happy path, example urls:

http://localhost:{{HOST_PORT}}/staticfiles/Serverless_comparison.pdf
http://localhost:{{HOST_PORT}}/staticfiles/light_green.jpg
http://localhost:{{HOST_PORT}}/staticfiles/solah01.mp3
http://localhost:{{HOST_PORT}}/staticfiles/simple-test.txt

I have a readme populated for details. With postman, one could easily automate testing of request and response headers. Due to time crunch I couldn't add those testing scenarios and negative testcases.

Design patterns used:

1] Chain of responsibility pattern to chain request response processors to process a request.

2] Factory pattern to instantiate all request response processors from the RRProcessorFactory and keep processors package protected.

3] Singleton pattern to instantiate RRProcessorFactory instance.

Enhancements/improvements that I couldn't get to:

1] To make it a secure server using HTTPS.

2] FileRRPRocessor to be enhanced for memory limitation, appropriate exception handling when trying to load a large file on one request and also to support multipart range requests.

3] FileListHandler to return json response of file names list.

4] Addition of unit testing, negative testing, http range testing including scenarios with different header combinations.

5] Minor improvements like using String constants, String buffers.