Milestone 3

Java.Lang.NoClassFoundException

By: Caleb Post and Matthew Lash

2. Change History:

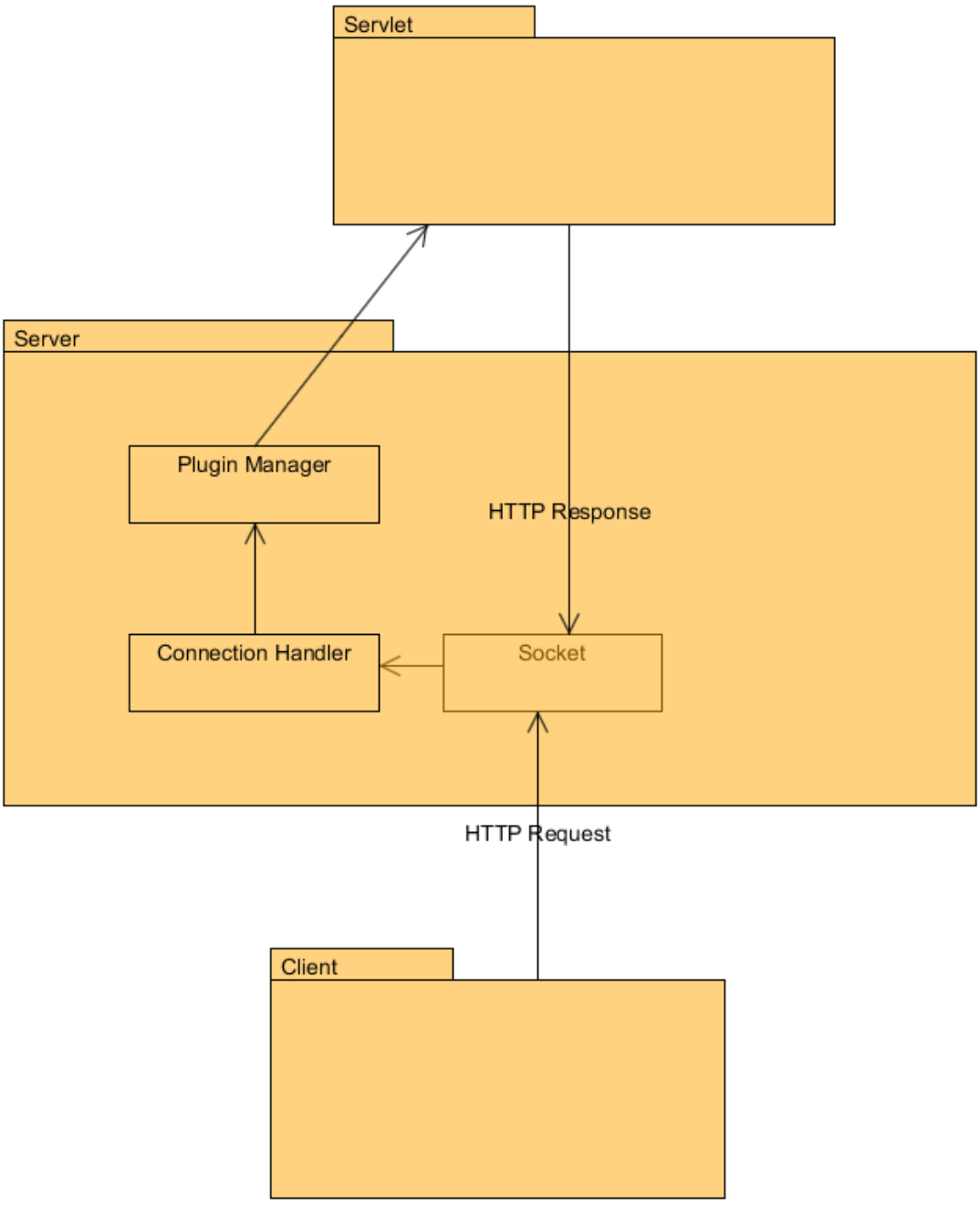
Revision Milestone 3

- Modified UML Diagram to reflect new methods and Logging System
- Added Tactics
- Added Improvements and Scenarios

Revision Milestone 2:

- Modified architecture to follow Servlets and Plugin Architecture
- Updated Architecture and UML Diagrams
- Added testing section for Servlet Features
- Added explanation of extension mechanism
- Feature Listing
- Expanded improvement section

3. Architecture and Design



UML Diagram Follows

Graphics

SpringUtilities
Resp: Provide utilities to help create SpringLayouts with Java Swing

WebServer
Resp: GUI containing visuals for Server.

Server Only

«interface»
Runnable

LogRequest
- SimpleDateFormat FORMATTER
- String LOG_FORMAT
String caller
String resource
String result
String timestamp
String toString()

Server
- LogManager manager
- HashMap<String, String> blacklistMap
- HashMap<String, Integer> connectionsMap
- WebServer window
- long connections
- long serviceTime
- ServerSocket welcomeSocket
- boolean stop
- int port
- String rootDirectory
- ServletManager manager
String getRootDirectory()
int getPort()
double getServiceRate()
incrementConnections(long value)
incrementServiceTime(long value)
run()
stop()
boolean isStoped()
removeThread(String)

ConnectionHandler
Server server
Socket socket
PluginManager manag
getSocket()
run()
Resp: Handle a single connection to the server

LogManager
Queue<LogRequest> requests
BufferedWriter writer
run()
Resp: writes log requests to disk while being thread safe.

GetRequestHandler
HTTPResponse handle(Server server, HTTPRequest request, HttpResponse toFill) throws ProtocolException

PostRequestHandler
HTTPResponse handle(Server server, HTTPRequest request, HttpResponse toFill) throws ProtocolException

PutRequestHandler
HTTPResponse handle(Server server, HTTPRequest request, HttpResponse toFill) throws ProtocolException

DeleteRequestHandler
HTTPResponse handle(Server server, HTTPRequest request, HttpResponse toFill) throws ProtocolException

HTTPResponseFactory
- HashMap<Integer,String> codesMapping
- fillGeneralHeader(HttpResponse response, String connection)
- HTTPResponse create200OK(File file, String connection, OutputStream stream)
- HTTPResponse create201Created(File newFile, String connection, OutputStream stream)
- HTTPResponse create202Accepted(String connection, OutputStream stream)
- HTTPResponse create204NoContent(String connection, OutputStream stream)
- HTTPResponse create304NotModified(String connection, OutputStream stream)
- HTTPResponse create400BadRequest(String connection, OutputStream stream)
- HTTPResponse create404NotFound(String connection, OutputStream stream)
- HTTPResponse create405MethodNotAllowed(String connection, OutputStream stream)
- HTTPResponse create410Gone(String connection, OutputStream stream)
- HTTPResponse create505NotSupported(String connection, OutputStream stream)
- HTTPResponse create500InternalServerError(String connection, OutputStream stream)
- HTTPResponse getResponse(int code, String text, OutputStream stream)
+ HTTPResponse createResponse(int code, String connection, File file, OutputStream strea
+ HTTPResponse createBlankResponse(String connection, OutputStream stream)

Runnable

PluginManager
Map<String, Servlet> locationMapping
Servlet getServletAtLocation(String location)
run()
- load(Servlet servlet)
- loadPlugin(Plugin plugin, String fileName)
- registerWatcher()
- loadPlugins()
- loadAndScanJar(File file,Action action)
Resp: Read Servlets from specified application directory. Load servlets dynamical runtime.
Serves the proper servlet for location

DefaultHTTPServlet
load()
shutdown()
Resp: Default HTTPServlet which handles the requests in the base fashion. Not registered. Default catch all.

«interface»
RequestHandler
HTTPResponse handle(Server server, HTTPRequest request, HttpResponse toFill) throws ProtocolException

RequestHandlerFactory
+ RequestHandler getRequestHandler(HTTPRequest request) throws ProtocolException

Exposed for Extension

Exception

ProtocolException
- int status

Servlet
Server running
Plugin container
abstract String getPath()
String getRootDirectory()
setServer(Server running)
setPlugin(Plugin container)
handle(HttpRequest request, HttpResponse response) throws ProtocolExcept
+ List<String> getAcceptedMethods()
Resp: Extensible class for implementing new servlets to add to server.

HTTPRequest
String method
String uri
String version
Map<String, String> header
BufferedReader input
boolean EOS
int remainingData
+ HTTPRequest read(InputStream input) throws Exception
boolean hasBodyData()
int readBody(char[] buffer) throws IOException

Protocol
+ String SPACE
+ String SEPERATOR
+ String SLASH
+ String CR
+ String LF
+ String CRLF
+ String VERSION
+ String GET
+ String POST
+ String PUT
+ String DELETE
+ int MISSING_METHOD_CODE
+ String MISSING_METHOD_TEXT
+ int OK_CODE
+ String OK_TEXT
+ int CREATED_CODE
+ String CREATED_TEXT
+ int ACCEPTED_CODE
+ String ACCEPTED_TEXT
+ int NO_CONTENT_CODE
+ String NO_CONTENT_TEXT
+ int MOVED_PERMANENTLY_CODE
+ String MOVED_PERMANENTLY_TEXT
+ int NOT_SUPPORTED_CODE
+ String NOT_SUPPORTED_TEXT
+ int BAD_REQUEST_CODE
+ String BAD_REQUST_TEXT
+ int NOT_FOUND_CODE
+ String NOT_FOUND_TEXT
+ int GONE_CODE
+ String GONE_TEXT
+ int METHOD_NOT_ALLOWED_CODE
+ String METHOD_NOT_ALLOWED_TEXT
+ int INTERNAL_SERVER_ERROR_CODE
+ String INTERNAL_SERVER_ERROR_TEXT
+ String HOST
+ String CONNECTION
+ String USER_AGENT
+ String DATE
+ String LOCATION
+ String Server
+ String LAST_MODIFIED
+ String CONTENT_LENGTH
+ String CONTENT_TYPE
+ String CHUNK_LENGTH
+ String SERVER_INFO
+ String PROVIDER
+ String AUTHOR
+ String CLOSE
+ String OPEN

HTTPResponse
String version
int status
String phrase
Map<String, String> header
File file
OutputStream outStream
write()
setStatus(int status,String statusTe
setStatus(int status)
setConnection(String connection)

«interface»
Plugin
load()
shutdown()
List<Servlet> getServlets()
String getLocation()
Resp: Dynamic program extension that can be added allows more than one servlet to send content

4. Tactics/Feature Listing

Milestone 3

Logging and Audits – Caleb
Dos Attack Prevention – Connection Throttling – Matt
Caching of requested Files – Caleb
Memory Limiting and Refactoring – Matt

Milestone 2

GET Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

POST Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

PUT Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

DELETE Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

Dynamic Loading: Design is was done by Caleb and Matt. Implementation was completed by Matt.

Root Context and Configurable Route: Design is was done by Caleb and Matt. Implementation was completed by Matt.

Milestone 1

GET Requests: Refactoring done by Caleb
POST Requests: Implementation done by Caleb
PUT Requests: Implementation done by Matt
DELETE Requests: Implementation by Matt

5. Architectural Evaluation and Improvements

Availability

Scenario 1

Source: Malicious User

Stimulus: Dos Attack

Artifact: Sever

Environment: Normal run condition

Response: It should cancel the connections that the attacker has

Response measure: How long server can survive before crashing

Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

After Improvement

Server still seems to run indefinitely without crashing. It ran for 3:30 minutes until we stopped the test. It did not drop a significant number of connections as we saw in the initial testing. We stopped the test because we were seeing periodic connection drops which could be expected of a server, with no indication that the server would enter the state at which it would start dropping most connections.

Scenario 2

Source: Users wanting to make requests

Stimulus: A user makes a request

Artifact: Server

Environment: Normal run conditions

Response: Server handles requests in parallel

Response measure: Total memory consumed by application

Test Plan

Tactic is to increase the capacity number of threads that can be running simultaneously on the server. This means limiting the memory impact of each individual thread. Implement the server such that it never needs to have entire files in memory or read entire contents of incoming requests before starting to service the connection. Increasing the total memory of the server could also be used for this tactic.

Before Improvement

During a DOS style attack, 18 threads run with an allocated memory of ~160MB on the heap of which it peaks to 40MB used.

Without a DOS attack and requesting a single large file uses a single thread which used a maximum of 3 MB change from the server when not requesting anything. (Difference from 33MB to 36MB) when requesting a video file of 118MB.

Improvement tactics

Tactic is to increase the capacity number of threads that can be running simultaneously on the server. This means limiting the memory impact of each individual thread. Implement the server such that it never needs to have entire files in memory or read entire contents of incoming requests before starting to service the connection. Increasing the total memory of the server could also be used for this tactic.

After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

Performance

Scenario 1

Source: A user who wants a file

Stimulus: A user requests a file

Artifact: Sever

Environment: Normal run condition

Response: The server serves the file

Response measure: The time between request and serving the content

Test Plan

We generate back to back requests of similar files and measure the time the server takes to responds. Then we implement our cashing technique and send the requests again and check to see if we get a better overall average response time.

Before Improvement

Initial Connection between 5ms and 7ms

Average connection latency after 100 requests of index page: 0.48623853211009177ms

Average connection latency after 30 request while DOS is hit: 11.072162087149598ms

Improvement tactics

This tactic is to implement a caching service for the server. For the most commonly requested resources, it should be in the cache, in memory, instead of needing to be read from disk with each request.

After Improvement

Without DOS: Initial Connection After: 7ms. Average Connection after improvement: 0.25ms

With DOS: 0.43902439024390244ms with latency dropping the longer we serviced. (Average latency, highest latency is loading the cache and is only done every cache clear.)

Scenario 2

Source: A malicious user

Stimulus: A dos attack on our server

Artifact: Sever

Environment: stressed run condition

Response: The server stops the dos attack to allow other users to not get slower performance

Response measure: Latency for legitimate client under dos attack

Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

Security

Scenario 1

Source: A malicious user

Stimulus: A dos attack on our server

Artifact: Server

Environment: Normal run condition

Response: The server stops the dos attack to allow other users to not get slower performance

Response measure: How long the survive before crashing

Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

Scenario 2

Source: A malicious user

Stimulus: User uploads virus

Artifact: Sever

Environment: Normal run condition

Response: The server should serve the request

Response measure: The server maintains an audit trail so the managers can find the malicious user

Test Plan

Tester 1 does some actions and tester 2 has to tell what different actions the first tester did. If he can successful do this for a variety of request then our audit system works. (he can send requests from different ip addresses)

Before Improvement

There is no way to backtrack malicious users

Improvement tactics

In order to track and trace the actions of potentially malicious individuals, maintain a log of all connections, requests, and origins such that they can be looked up at a later date to determine the actions of any user on the server. This will be implemented as a log file.

After Improvement

There is a way to backtrack malicious users

Future Improvements

Milestone 3

Security could be improved further by implementing HTTPS by securing a certificate for the server and adding in the encryption layer.

Security could be improved further by authentication of users and requiring credentials for use.

Security could be improved further by implementing access controls based on served content.

Caching could be further improved to remove specific elements from cache as they are out of date instead of clearing the entire cache periodically. Logic could be added to cache only those files which are commonly requested.

Milestone 2

Throttling mechanisms could be implemented to allow for the server to monitor connections that are consuming too many resources and limit them. This could be done proactively by blocking requests that are too consuming, or it could be done actively by killing threads that are consuming too many resources and blocking other clients.

Recovery mechanisms could be put into place such as request queueing and blocking if the server becomes stressed under too many requests being made at once.

If resources become a problem we may need to implement servlet cloning. Currently there is a single servlet instance hosted which handles all requests forwarded to that servlet. This works in the small scale scenario that we have been testing. We may need to multithread these servlets so that the same servlet can handle more than one request at a time, or we many need to clone the servlet in order to make each thread have a copy of it.

Preventative measures could be implemented such as monitoring connections and requests, blocking any users who appear to be malicious and consuming more connections. This would help with preventing DDOS attacks against the server.

Milestone 1

As suggested by Chandan, we should do some work with a command pattern perhaps with the generation of the responses. The way that the code currently is, all the work is done within the factory and the system is not very extensible with the responses.

Also, we could encapsulate the handling the HTTP messages into a more plugin like architecture. This would make it so that we could also handle different types of requests beyond HTTP requests without extensive modification of the system, only through adding a new plugin which respects the design we have put in place.