Relazione sul Progetto dell'Esame di Reti di Calcolatori Anno Accademico 2017/18 (Sessione Gennaio/Febbraio)

D'Autilia Mattia - 5765968 – mattia.dautilia@stud.unifi.it

Descrizione Progetto

L'obiettivo del progetto è quello di sviluppare un server HTTP. Il progetto prevede:

- Lo sviluppo di una serie di classi in Java che implementano le funzionalità e che estendono le interfacce messe a disposizione sul sito del corso;
- La realizzazione di una serie di test in Java che verifichino il corretto funzionamento delle funzionalità base;
- La realizzazione di una serie di script Python che mostrano il comportamento del server.

Consegna

Il file .zip 5765968 contiene:

- Una directory chiamata Java con all'interno le Classi implementate (nel del pacchetto "it.unifi.rc.httpserver.m5765968"), Classi di Test (nel pacchetto "it.unifi.rc.httpserver.m5765968.Test") e Javadoc (nella cartella "doc").
- Una directory chiamata Python con all'interno gli Script in Python.

Livelli Java implementati: 0, 1, 2, 3, 4, 5, 6, 7.

Classi implementate :

- *MyHTTPRequest e MyHTTPReply*
- *MyHTTPInputStream e MyHTTPOutputStream*
- *MyHTTPHandler1_0,MyHTTPHandlerHost1_0, MyHTTPHandler1_1 e MyHTTPHandlerHost1_1*
- *MyHTTPServer, MyHTTPThreadServer e MyHTTPThreadClient*
- *MyHTTPFactory*
- *MyHTTPServerMain*

Classi Test :

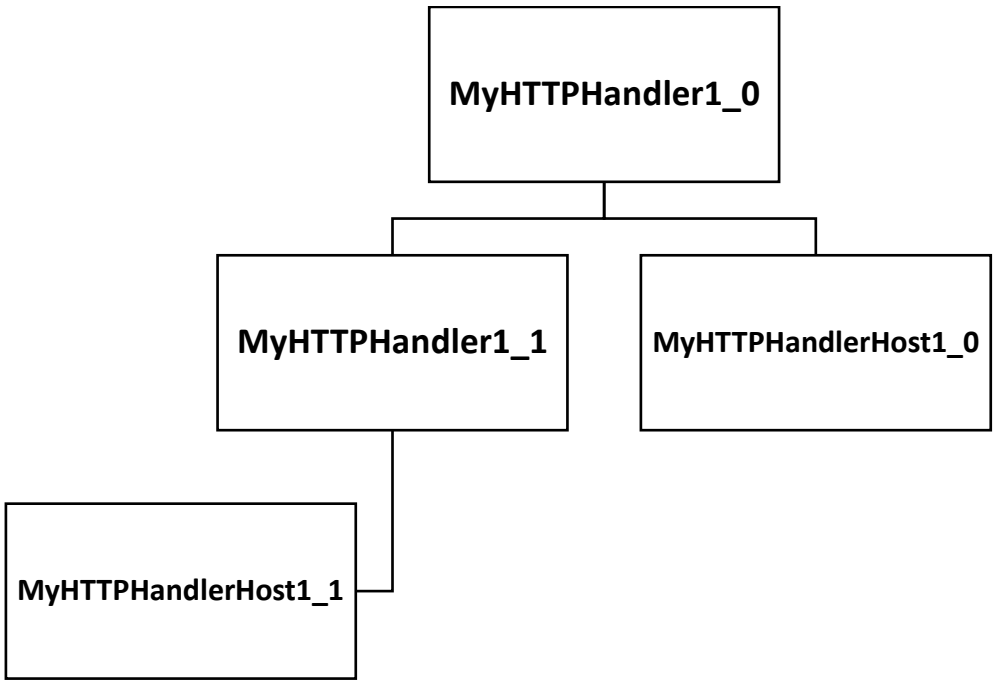
- *MyHTTPRequestTest e MyHTTPReplyTest*
- *MyHTTPInputStreamTest e MyHTTPOutputStreamTest*
- *MyHTTPHandler1_0Test, MyHTTPHandlerHost1_0Test, MyHTTPHandler1_1Test e MyHTTPHandlerHost1_1Test*
- *MyHTTPSingleClientTest*

Script Python:

- *RequestHTTP1_0 :* verifica le funzionalità implementate del protocollo HTTP/1.0 tramite l'invio da parte dello stesso client di tre richieste corrette (GET, HEAD, POST).
- *RequestHTTP1_1 :* verifica le funzionalità implementate del protocollo HTTP/1.1 tramite l'invio da parte dello stesso client di 6 richieste corrette (GET, HEAD, POST, DELETE, PUT(File esistente), PUT(File nuovo)).
- *RequestErrorHost :* verifica le funzionalità implementate del protocollo HTTP/1.0 e HTTP/1.1 tramite l'invio da parte dello stesso client di 2 richeste le quali hanno un valore "Host" non corretto (inesistente).
- *RequestErrorMethod :* verifica le funzionalità implementate del protocollo HTTP/1.0 e HTTP/1.1 tramite l'invio da parte dello stesso client di 2 richeste le quali hanno un valore "Metodo" non corretto.
- *RequestNoFile :* verifica le funzionalità implementate del protocollo HTTP/1.0 e HTTP/1.1 tramite l'invio da parte dello stesso client di 2 richeste le quali hanno un valore "Path" che riferisce a una directory.
- *RequestNoPath :* verifica le funzionalità implementate del protocollo HTTP/1.0 e HTTP/1.1 tramite l'invio da parte dello stesso client di 2 richeste le quali hanno un valore "Path" che riferisce a una file non esistente.

Funzionalità Handler

Ho implementato gli Handler per i 2 protocolli HTTP/1.0 e HTTP/1.1 con l'utilizzo della tecnica dell'ereditarietà, nel seguente modo:



La classe **MyHTTPHandler1_0** gestisce le richieste HTTP/1.0 controllando prima se la versione corrisponde a quella della classe (HTTP/1.0), dopo di che controlla se il Path corrisponde a un file e se quel file esiste. In tal caso tale classe gestisce solamente i metodi GET (legge un file e ne restituisce il contenuto), HEAD (legge un file e ne restituisce solo le intestazioni) e POST (scrive su file in modalità APPEND) rispondendo con "200 OK". Altrimenti, in base al controllo errato, risponde negativamente (NoFile: 403 Directory listing not supported; NoPath: 404 Not Found; NoMethod: 405 Method Not Allowed). Solo in caso di versione errata risponde con "null".

La classe **MyHTTPHandler1_1** gestisce le richieste HTTP/1.1 controllando prima sia se la versione corrisponde a quella della classe (HTTP/1.1), sia se la richiesta contiene la riga di intestazione con campo "Host" (in caso contrario ritorna "null"). Dopo di che controlla il metodo, il quale, se corrisponde a (GET,HEAD,POST) richiama il metodo "handle()" della sopra classe MyHTTPHandle1_0 in quanto la gestione di quei metodo è la medesima. Invece se il metodo corrisponde a PUT (scrive su file in modalità WRITE, lo crea eventualmente) chiama il metodo

“handlePut()”, se corrisponde a DELETE (rimuove un file) chiama il metodo “handleDelete()”. Altrimenti, in base al controllo errato, risponde negativamente (NoFile: 403 Directory listing not supported; NoPath: 404 Not Found; NoMethod: 405 Method Not Allowed). Solo in caso di versione errata risponde con “null”.

La classe **MyHTTPHandlerHost1_0** gestisce le richieste HTTP/1.0 il cui valore Host corrisponde con quello indicato nell’handler (nel mio web-server è impostato con il valore Mio.PC). Se tale uguaglianza ci fosse, allora la richiesta sarebbe gestita dall’handler della sopra classe MyHTTPHandler1_0 con tutti i dovuti sotto controlli. In caso contrario la risposta avrebbe valore “null”.

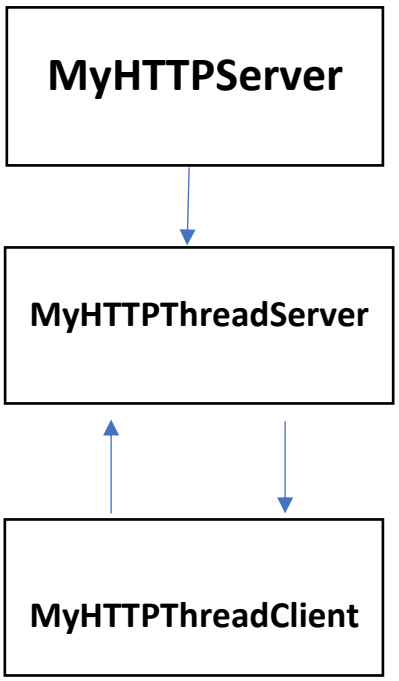
La classe **MyHTTPHandlerHost1_1** gestisce le richieste HTTP/1.1 il cui valore Host corrisponde con quello indicato nell’handler (nel mio web-server è impostato con il valore Mio.PC). Se tale uguaglianza ci fosse, allora la richiesta sarebbe gestita dall’handler della sopra classe MyHTTPHandler1_1 con tutti i dovuti sotto controlli. In caso contrario la risposta avrebbe valore “null”.

Se nessun handler gestisce la richiesta alla sarà inviata al client un risposta la cui riga di stato è di questo tipo : “version 400 Bad Request”.

Funzionalità Server

La class MyHTTPServer implementa HTTPServer e utilizzando il metodo start() è possibile avviare un “Server Thread” (il quale resterà attivo per un tempo limite fissato a 2 min) che avrà il solo compito di accettare i client che si vogliono collegare(ogni Client accettato viene inserito un lista di controllo), ai quali sarà assegnato un “Client Thread” separato, per la gestione delle sue richieste. Attraverso la classe MyHTTPThreadServer, il server può comunicare in parallelo con i vari Client identificati dalla classe MyHTTPThreadClient Thread, utilizzando il multithreading. Le connessioni possono essere persistenti in quanto negli handler vien gestito il tipo di connessione (Close: chiudi connessione; Keep-alive : mantieni

connessione permettendo al client di inviare più di un messaggio al server sulla stessa connessione).Quando il tempo di esecuzione del “Thread Server” termina, viene richiamato il metodo stop() della classe MyHTTPServer, il quale a sua volta chiamerà il metodo stop() del “Thread Server”, scollegando oltre al Server, tutti i client rimasti attivi.



Istruzioni per l’esecuzione

Per eseguire gli Script Python deve essere lanciato inizialmente un Server Java, tramite l’utilizzo della classe MyHTTPServerMain, utilizzando “Host: Mio.PC” e “Porta: 12000”, il quale crea (se non sono già esistenti) i 6 file (TestGetRequest.html; TestHeadRequest.html; TestPostRequest.html”; TestDeleteRequest.html; TestPutNewFileRequest.html; TestPutExistsFileRequest.html) su qui vengono effettuate le richieste degli script. All’interno degli script, per inviare la prima richiesta e per passare da una richiesta all’altra, l’utente dovrà premere invio.