Simple Web Server

ZHOU YUKAI

24/05/2024

Matricola: 0001090431

Email:yukai.zhou@studio.unibo.it

1. Introduzione:

MyServer.py implementa un server HTTP di base, supporta il multi-threading, può gestire più richieste GET e restituisce file statici corrispondenti in base alla richiesta del Cliente.

2. Funzionamento del Server:

Il server gestisce le richieste HTTP attraverso la **classe MyCustomRequestHandler**, che comprende principalmente le seguenti funzioni:

2.1 Gestione delle richieste GET

Dopo che il server ha ricevuto la richiesta GET, inizierà ad analizzare l'indirizzo URL per determinare il contenuto a cui il client desidera accedere.

In caso non specifica alcun nome di file : *http:///localhost:8080/*

il percorso analizzato è la **directory root** (ovvero, non è specificato alcun nome del file), il server fornirà il contenuto del file indice.html.
Se il file index.html non viene trovato, verrà fornita una pagina predefinita e verrà visualizzato un messaggio che informa che il server corrente non ha alcun contenuto disponibile.

In caso ha specificato un nome(con estensione):
http:///localhost:8080/filename.pdf/html/txt…

In questo caso, il nome del file verrà concatenato con la directory di lavoro corrente per formare un percorso assoluto e verrà controllato se il file corrispondente esiste nella directory di lavoro corrente. Se esiste e tipo di file venga riconosciuto, una risposta HTTP 200 ok e il contenuto del file corrispondente sarà mandato al Cliente. In caso contrario viene inviato un codice 404 che indica che il file richiesto non esiste.

Se il file esiste ma il tipo non viene riconosciuto dal gestore della richiesta, il file verrà scaricato localmente. (per esempio file.exe)

2.2 Gestione della cache

Inoltre, vengono forniti tempi di cache diversi per diversi tipi di file statici.
E controllando la data dell'ultima modifica del file, determina se il contenuto nella cache deve essere aggiornato. In caso che il file venga modificato, il server invierà HTTP 200 e il contenuto del file più recente.
Altrimenti, invierà un codice 304 solo per comunicare al cliente che non è necessario alcun aggiornamento.

2.3 Gestione degli errori

Oltre al codice di stato 404, il gestore delle richieste ha altre capacità di gestione degli errori. Nel mio server, il client può accedere solo ai contenuti dei file della directory di lavoro corrente e delle sue sottodirectory. Se il client tenta di accedere a directory diverse, il server invierà il codice 403 e l'accesso sarà rifiutato.

（E il server invia codice 500 in caso di errore non previsto）

3. Esecuzione del server

3.1 Sulla riga di comando

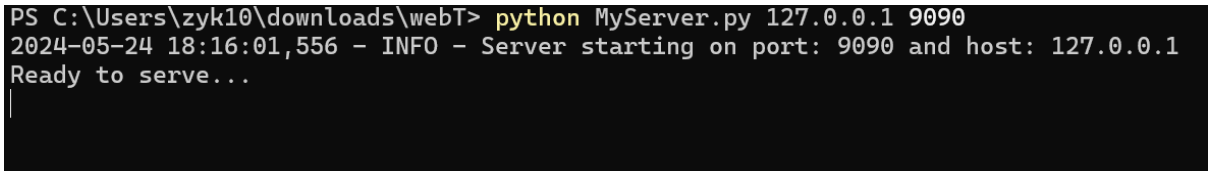
Naviga nella directory dove si trova il file tramite la riga di comando, poi esegue MySercer.py in seguenti modi :

Esecuzione **predefinita**: Utilizza il nome host predefinito (`localhost`) e la porta (`8080`): **python MyServer.py**

Esecuzione con **porta specificata**: Utilizza il nome host predefinito (`localhost`) e una porta specificata, ad esempio: **python MyServer.py 9090**

Esecuzione con nome **host e porta specificati**: Utilizza un nome host e una porta specificati, ad esempio: **python MyServer.py 127.0.0.1 9090**

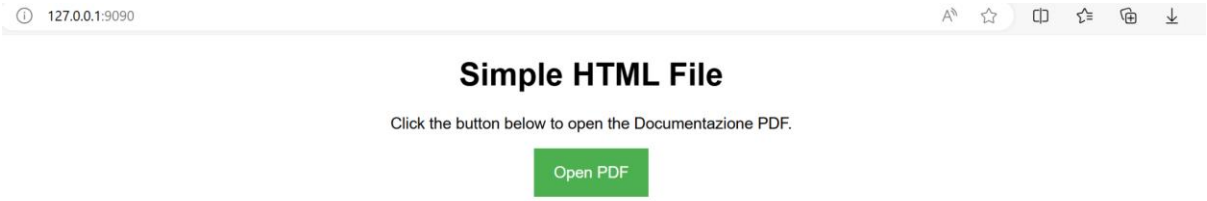
Esempio di esecuzione con successo:



Ora, apri un qualsiasi browser e nella barra degli indirizzi inserisci l'URL, ovvero:
http:// «hostname»:portNumber/index.html.

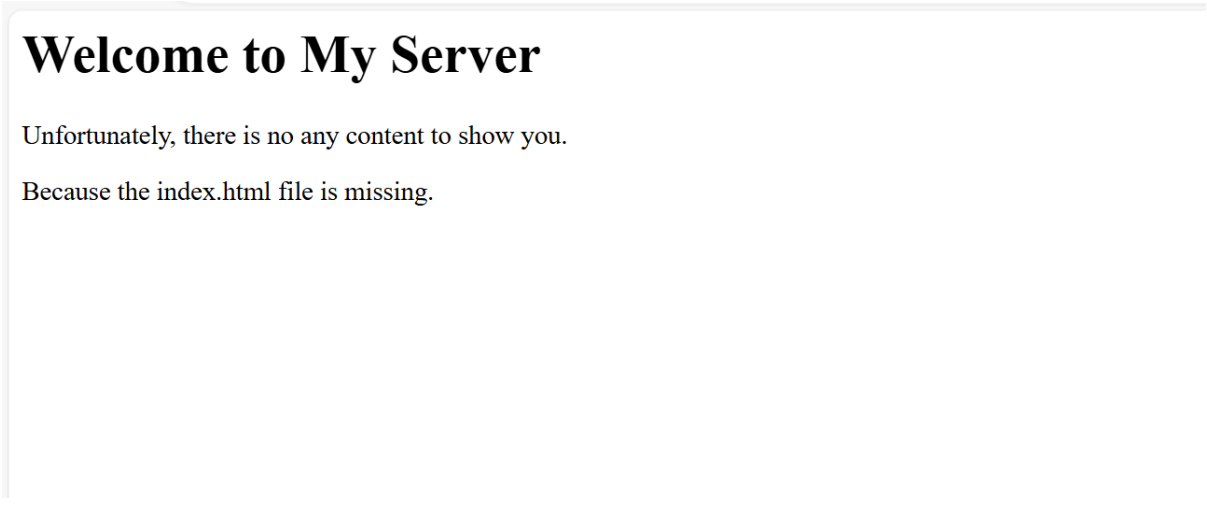
Esempio:
http://127.0.0.1:9090/index.html
oppure
http://127.0.0.1:9090/

Vedremo:



Ora cliccare il pulsante verde nell'immagine, può visualizzare il mio Documentazione del progetto.

Se index.html non viene trovato nella directory, il server invia il contenuto predefinito come sotto:



Chiusura
Per chiudere il server, basta premere la combinazione di tasti **Ctrl + C** nel terminale, Il programma terminerà senza errori.

4.Considerazione aggiuntive

4.1 Utilizzo di logging

Nello sviluppo di implementazione del server, ho utilizzato il modulo di logging per registrare le attività del server, inclusi i registri di accesso e di errore, per facilitare il debug e la manutenzione.

4.2 Test del server

Per verificare che il server risponda correttamente ai vari codici di stati HTTP(come 200, 403, 404), ho anche implementato un codice clinet.py per simulare le richieste del browser e osservare le risposte del server.

4.3 Possibili miglioramenti futuri

Supporto per contenuti dinamici: Implementare funzionalità che consentano al server di fornire contenuti dinamici come video e musica. Per esempio, si potrebbe utilizzare Node.js per gestire lo streaming di contenuti multimediali in modo efficiente.