Documentazione web server http multithread

1. Introduzione

Il codice presentato è un semplice web server HTTP scritto in Python, progettato per servire file statici, HTML e immagini PNG per esempio, inviati tramite protocollo TCP in risposta alle richieste HTTP GET. Utilizza il modulo `http.server` per gestire le richieste HTTP e `socketserver` per implementare la gestione del server.

2. Descrizione del Funzionamento

2.1. ThreadingTCPServer

Il cuore di questo server è l'oggetto `ThreadingTCPServer`, che estende la classe `TCPServer` di Python. Questa classe permette al server di gestire più richieste contemporaneamente. Ogni volta che riceve una richiesta, viene creato un nuovo thread figlio per gestirla, mentre il server rimane inalterato sul thread padre. In questo modo possono essere gestiti molti client simultaneamente, fintanto che le risorse del sistema e della rete lo permettono. Il parametro `allow_reuse_address` è impostato a `True` per permettere al server di riutilizzare immediatamente l'indirizzo e la porta dopo la chiusura del server. Questo è utile durante lo sviluppo, poiché evita di dover attendere il rilascio della porta dopo ogni esecuzione del server.

2.2. myRequestHandler

La classe `myRequestHandler` estende `BaseHTTPRequestHandler` e definisce il comportamento del server in risposta alle richieste HTTP GET delle quali il metodo `do_GET` è responsabile: quando un client invia una richiesta GET, il server esamina il percorso del file richiesto, cerca il file nel filesystem locale e lo restituisce al client se viene trovato e supportato.

Questo server supporta solo file statici (html e png sono stati presi in esempio), poiché utilizza il protocollo FTP per il trasferimento.

Gestione degli errori con codici standard HTTP:

- se il file richiesto non è presente nella directory del server, esso risponde con il codice `404 Not Found`
- se il file richiesto non è presente nell'elenco delle estensioni supportate, il server risponde con il codice `415 Unsupported Media Type`.
- Se si verifica un errore durante la lettura del file, il server risponde con il codice `500 Internal Server Error`, indicando un problema lato server.

2.3. Gestione delle Eccezioni e Chiusura del Server

Il server può essere interrotto solo tramite eccezioni, assicurandosi che tali, sia previsti (come il `KeyboardInterrupt` “Ctrl + C”) sia imprevisti, siano gestiti correttamente. Alla fine di ogni esecuzione, il server viene chiuso utilizzando il metodo `server_close()` all'interno di un blocco `finally`, liberando le risorse e le porte utilizzate.

3. Utilizzo del Codice

Per eseguire questo server, è necessario fornire l'indirizzo IP e, facoltativamente, una porta attraverso la riga di comando. A seguire le istruzioni su come utilizzare il codice:

3.1 Esecuzione del server

Aprire il terminale nella directory in cui è salvato il file del server e eseguire il comando:

```
python nome_file.py <IP> <port>
```

`<IP>` è l'indirizzo IP del server (es. `127.0.0.1` per localhost), il proprio indirizzo IPv4 può essere ricavato tramite il comando `ipconfig.`

`<port>` è il numero di porta (opzionale; se non specificato, sarà 8080), alcune porte sono riservate al sistema o non adatte al protocollo TCP.

3.2 Accesso ai contenuti

Una volta che il server è in esecuzione, aprire un browser web e navigare all'indirizzo:

```
http://<IP>:<port>/
```

Se un file `index.html` è presente nella directory da cui è stato avviato il server, verrà automaticamente servito come homepage.

È anche possibile richiedere specifici file aggiungendo il loro nome al percorso, ad esempio:

```
http://<hostIP>:<port>/title.html
```

3.3 Chiusura del server

Per interrompere il server, premi `Ctrl+C` nel terminale. Il server risponderà chiudendo tutte le connessioni aperte e liberando le risorse.