

Elaborato per il corso Programmazione di Reti

A.A 2023/2024

Creare un web server semplice in Python, in grado di gestire più richieste simultaneamente e restituire risposte appropriate ai client.

Golesano Giulia
giulia.golesano@studio.unibo.it
0001069061

Introduzione:

Un web server è un software che, in esecuzione su Server, è in grado di gestire le richieste di trasferimento di file che riceve dai Client. Si tratta di un'applicazione che utilizza il modulo `http.server` ed è in grado di gestire più richieste simultaneamente e restituire risposte appropriate ai Client.

Funzionamento del Codice:

1. Il codice definisce una classe `SimpleHTTPRequestHandler` che eredita da `http.server.SimpleHTTPRequestHandler`. Questa classe viene utilizzata per gestire le richieste HTTP GET.
2. La classe sovrascrive il metodo `do_GET()` per personalizzare il comportamento del server quando riceve una richiesta GET.
3. All'interno del metodo `do_GET()`, il server controlla se il percorso richiesto corrisponde a un file esistente utilizzando la funzione `os.path.exists()`.
4. Se il file richiesto esiste, il server lo serve utilizzando il metodo `http.server.SimpleHTTPRequestHandler.do_GET()`, che restituisce il contenuto del file richiesto al client.
5. Se il file richiesto non esiste, il server restituisce un messaggio di errore "File not found" con una risposta HTTP 404.

Concetto di Base:

Il concetto dietro il web server è quello di utilizzare il modulo `http.server` di Python per gestire le richieste HTTP in arrivo e restituire file statici al client. Il server utilizza un approccio basato su thread per gestire più richieste contemporaneamente. Quando il server riceve una richiesta, crea un nuovo thread per gestire quella richiesta, consentendo al server di rimanere attivo e di rispondere a ulteriori richieste in arrivo.

I moduli importati sono molto importanti e specifici per le funzioni richieste: `http.server` fornisce classi base per implementare server HTTP in Python, `socketserver` offre un'infrastruttura di rete generica per gestire diverse operazioni di socket, `os` fornisce funzionalità per l'interazione con il sistema operativo, come la gestione dei percorsi dei file.

Il codice definisce una variabile `port` che specifica la porta su cui il server HTTP ascolterà le richieste. In questo caso, è impostata su 8000.

Successivamente, viene definita una classe `SimpleHTTPRequestHandler` che eredita dalla classe `http.server.SimpleHTTPRequestHandler`. Questa classe personalizzata definisce il comportamento del server per gestire le richieste HTTP GET.

Il metodo `do_GET` sovrascritto controlla se il percorso richiesto dal client esiste nel sistema file. Se il file esiste, il server utilizza il comportamento predefinito della classe base `http.server.SimpleHTTPRequestHandler` per servire il file richiesto. Se il file non esiste, il server restituisce una risposta 404 (File not found).

Infine, il server viene avviato utilizzando la classe `socketserver.ThreadingTCPServer`, che crea un server TCP in un thread separato per ogni richiesta.

In sintesi, questo codice crea un server HTTP che può gestire richieste GET di base e servire file statici. Se il file richiesto non esiste, il server restituirà una risposta 404.

Utilizzo del Codice:

1. Salvare il codice in un file.
2. Inserire nella medesima directory i file statici di cui desideriamo testare l'invio.
3. Avvia il server eseguendo il file Python.
4. A questo punto troveremo i file statici inviati all'indirizzo <http://localhost:8000> del browser web, compreso il file Python, che può essere altrimenti inserito in una sottodirectory.

Conclusioni:

Uno script del genere è uno strumento utile per lo sviluppo e il testing di pagine web e applicazioni web semplici. Il codice può essere personalizzato ulteriormente per aggiungere funzionalità avanzate come il supporto per altri metodi HTTP, l'autenticazione degli utenti, la gestione delle sessioni, e altro ancora, come si può vedere nello script [advancedServer.py](#)

Chiarimenti sul codice più avanzato proposto:

In questa versione del codice con alcune funzionalità aggiuntive vediamo la definizione dei metodi `do_POST`, `do_AUTH` e `do_SESSION`, per gestire quindi post, autenticazioni e sessioni.

do_POST: Il metodo inizia leggendo la lunghezza dei dati POST dal campo `Content-Length` dell'intestazione della richiesta. Questa lunghezza indica quanti byte di dati sono presenti nel corpo della richiesta. Una volta letti i dati POST, il server può elaborarli in base alle esigenze dell'applicazione. Ad esempio, i dati potrebbero essere memorizzati su un database, utilizzati per aggiornare lo stato dell'applicazione o per eseguire altre operazioni. Dopo aver elaborato i dati POST, il server invia una risposta al client per confermare che la richiesta è stata ricevuta e elaborata correttamente. In breve, il metodo `do_POST` gestisce i dati inviati tramite richieste POST, li elabora e invia una risposta al client.

do_AUTH: Il metodo inizia analizzando l'intestazione `Authorization` della richiesta HTTP per cercare eventuali credenziali inviate dal client. Se le credenziali sono presenti nell'intestazione `Authorization`, il server le verifica per determinare se l'utente è autorizzato ad accedere alla risorsa richiesta. Se le credenziali sono valide, il server restituisce una risposta di successo (ad esempio, codice di stato HTTP 200). Altrimenti, il server restituisce una risposta di errore (ad esempio, codice di stato HTTP 401) e richiede all'utente di fornire nuove credenziali. In sintesi, il metodo `do_AUTH` gestisce l'autenticazione degli utenti

verificando le credenziali fornite dal Client e restituendo una risposta appropriata in base al risultato della verifica.

do_SESSION: Il metodo inizia analizzando l'intestazione Cookie della richiesta HTTP per cercare eventuali cookie inviati dal client, tramite `http.cookies`. Se non viene trovato alcun cookie di sessione nella richiesta, il server genera un nuovo ID sessione utilizzando `os.urandom(16).hex()` (questo metodo genera una stringa esadecimale casuale di 16 byte, che viene utilizzata come ID sessione univoco per l'utente). Una volta generato l'ID sessione, il server lo invia al client impostandolo come valore del cookie di sessione nell'intestazione `Set-Cookie`. In questo modo, il client riceve e memorizza l'ID sessione per le future richieste. Se l'ID sessione è già presente nei cookie inviati dal client, il server utilizza quell'ID sessione per identificare l'utente. Questo consente al server di mantenere la coerenza delle sessioni tra le richieste successive dello stesso utente. Infine, il server invia una risposta al client contenente l'ID sessione o altri dati relativi alla sessione. In un'applicazione reale si potrebbe utilizzare l'ID sessione per recuperare dati della sessione dal server e inviarli al client. In breve, il metodo `do_SESSION` si occupa di gestire l'identificazione degli utenti attraverso ID sessione univoci e di mantenere la coerenza delle sessioni tra le richieste successive dello stesso utente utilizzando i cookie.