Documentazione del Codice Python per un Server HTTP

Giacomo Biagioni

July 1, 2024

1 Introduzione

Questo documento descrive il funzionamento di uno script Python che avvia un server HTTP multithreading. Il server è configurato per gestire richieste simultanee utilizzando i thread. Inoltre, supporta la chiusura pulita tramite l'uso del segnale di interruzione (Ctrl+C).

2 Codice

Di seguito è riportato il codice completo con commenti esplicativi:

```
import sys, signal
import http.server
import socketserver

# Se c'e' un argomento nella riga di comando, viene usato come
    porta. Altrimenti, viene usata la porta 8080.
if sys.argv[1:]:
    port = int(sys.argv[1])
else:
    port = 8080

# Crea un server TCP che gestisce piu' richieste simultaneamente
    usando i thread.
server = socketserver.ThreadingTCPServer(('', port), http.server.
    SimpleHTTPRequestHandler)
print(f"Il server HTTP verra' eseguito all'indirizzo http://
    localhost:{port}/")

# Configura il server per usare thread daemon e permettere il
    riutilizzo dell'indirizzo.
server.daemon_threads = True
server.allow_reuse_address = True
```

```
# Definisce una funzione per gestire il segnale di interruzione (Ctrl+C).
def signal_handler(signal, frame):
    print('Uscita dal server HTTP (premuto Ctrl+C)')
    try:
        if server:
            server.server_close()  # Chiude il server se attivo.
    finally:
        sys.exit(0)  # Esce dal programma.

# Associa il segnale SIGINT (interruzione, ad esempio, Ctrl+C) alla funzione signal_handler.
signal.signal(signal.SIGINT, signal_handler)

# Tenta di eseguire il server indefinitamente.
try:
    while True:
        server.serve_forever()  # Mantiene il server in esecuzione.
except KeyboardInterrupt:
    pass  # Cattura l'eccezione di interruzione della tastiera (Ctrl+C).

# Chiude il server all'uscita dal ciclo.
server.server_close()
```

Listing 1: Codice del Server HTTP

3 Descrizione del Codice

3.1 Importazione delle Librerie

Il codice inizia importando i moduli necessari:

```
import sys, signal
import http.server
import socketserver
```

- `sys, signal`: Moduli per gestire segnali e parametri della riga di comando.
- `http.server`: Modulo per creare un server HTTP.
- `socketserver`: Modulo per creare server di rete con capacità di threading.

3.2 Configurazione della Porta

Il codice verifica se è stato passato un argomento nella riga di comando per determinare la porta su cui il server ascolterà. Se non viene passato alcun argomento, usa la porta predefinita 8080.

```
if sys.argv[1:]:
    port = int(sys.argv[1])
else:
    port = 8080
```

3.3 Creazione del Server

Un'istanza di `ThreadingTCPServer` viene creata per gestire richieste HTTP simultanee utilizzando i thread. Il server è configurato per usare thread daemon e per permettere il riutilizzo dell'indirizzo.

```
server = socketserver.ThreadingTCPServer(('', port), http.server.
    SimpleHTTPRequestHandler)
print(f"Il server HTTP verra' eseguito all'indirizzo http://
    localhost:{port}/")

server.daemon_threads = True
server.allow_reuse_address = True
```

3.4 Gestione del Segnale di Interruzione

Viene definita una funzione `signal_handler` per gestire il segnale di interruzione (Ctrl+C). Questa funzione chiude il server e termina il programma in modo pulito.

```
def signal_handler(signal, frame):
    print('Uscita dal server HTTP (premuto Ctrl+C)')
    try:
        if server:
            server.server_close()  # Chiude il server se attivo.
    finally:
        sys.exit(0)  # Esce dal programma.
```

3.5 Associazione del Segnale

Il segnale SIGINT, generato premendo Ctrl+C, è associato alla funzione `signal_handler`.

```
signal.signal(signal.SIGINT, signal_handler)
```

3.6 Esecuzione del Server

Il server è eseguito indefinitamente tramite un ciclo `while` che chiama `serve_forever()`. In caso di interruzione della tastiera (Ctrl+C), l'eccezione `KeyboardInterrupt` viene catturata per uscire dal ciclo e chiudere il server.

```
try:
    while True:
        server.serve_forever()  # Mantiene il server in esecuzione.
except KeyboardInterrupt:
    pass  # Cattura l'eccezione di interruzione della tastiera (
        Ctrl+C).

server.server_close()
```

4 Esecuzione del Programma

Per eseguire il programma, apri un terminale e naviga nella directory in cui si trovano i files webserver.py, image.jpg e index.html. Poi, esegui il seguente comando:

```
python webserver.py
```

Se desideri specificare una porta diversa da quella predefinita (8080), aggiungi il numero di porta come argomento:

```
python webserver.py 9090
```

In questo esempio, il server HTTP verrà eseguito sulla porta 9090.
Dopo aver eseguito il comando, vedrai un messaggio simile a questo nel terminale:

```
Il server HTTP verra' eseguito all'indirizzo http://localhost:9090/
```

A questo punto, puoi aprire il tuo browser e navigare all'indirizzo indicato per vedere il server HTTP in funzione. Per interrompere l'esecuzione del server, premi `Ctrl+C` nel terminale. Questo attiverà la gestione del segnale e chiuderà il server in modo pulito.

5 Considerazioni sui Server Multithread

L'implementazione di server multithread presenta numerosi vantaggi:

5.1 Vantaggi

- **Gestione delle Richieste Simultanee**: I server multithread sono in grado di gestire più richieste simultaneamente, migliorando la reattività e le prestazioni complessive del server.
- **Semplicità di Implementazione**: Utilizzare thread per gestire le richieste può semplificare l'implementazione del server, poiché ogni richiesta viene gestita in modo indipendente.
- **Utilizzo Efficiente delle Risorse**: I server multithread possono utilizzare le risorse del sistema in modo più efficiente, sfruttando i core multipli dei moderni processori.

5.2 Funzionamendo dei WebServer Multithread

I server multithread funzionano creando e gestendo più thread di esecuzione all'interno dello stesso processo per gestire le richieste dei client. Quando un server multithread riceve una nuova richiesta, non blocca l'esecuzione di altre richieste in arrivo; invece, assegna la richiesta a un thread separato. Questo thread si occupa della richiesta in modo indipendente e parallelo agli altri thread. Grazie a questa struttura, il server può continuare ad accettare e gestire altre richieste mentre un thread è occupato con una richiesta specifica.
Questo approccio migliora notevolmente la capacità del server di gestire più richieste contemporaneamente rispetto ai server single-threaded, che elaborano una richiesta alla volta. La creazione e gestione di thread consente al server di sfruttare meglio le risorse del sistema, specialmente su macchine con processori multi-core, dove più thread possono essere eseguiti effettivamente in parallelo. Tuttavia, è importante gestire correttamente la sincronizzazione tra i thread per evitare problemi di concorrenza, come condizioni di competizione e deadlock.

5.3 Alternativa: Server Multiprocess

Un'alternativa ai server multithread è l'uso di server multiprocess, dove ogni richiesta viene gestita da un processo separato invece che da un thread. Questo approccio può risolvere alcuni dei problemi di sincronizzazione e scalabilità, ma può introdurre altri problemi, come un maggiore overhead di creazione dei processi e un uso inefficiente delle risorse di sistema.

In conclusione, la scelta tra un server multithread e un server multiprocess dipende dalle specifiche esigenze dell'applicazione e dalle risorse disponibili. È importante valutare attentamente i pro e i contro di ciascun approccio per garantire le migliori prestazioni e affidabilità del server.