Documentazione del Codice Python

Federico Morsucci

1 Introduzione

Il codice presentato è uno script Python che implementa un semplice server HTTP multithread. Il server legge il numero della porta dalla riga di comando o utilizza la porta di default 8080. Inoltre, il server gestisce in modo pulito l'interruzione del processo tramite la combinazione di tasti Ctrl-C.

2 Codice

```
#!/bin/env python
import sys, signal
import http.server
import socketserver

# Legge il numero della porta dalla riga di comando
if sys.argv[1:]:
  port = int(sys.argv[1])
else:
  port = 8080

# ThreadingTCPServer per gestire piu richieste
server = socketserver.ThreadingTCPServer(('',port), http.server.
    SimpleHTTPRequestHandler )

server.daemon_threads = True
server.allow_reuse_address = True

#definiamo una funzione per permetterci di uscire alla pressione di Ctrl-
    C
def signal_handler(signal, frame):
    print( 'Exiting http server (Ctrl+C pressed)')
    try:
      if( server ):
        server.server_close()
    finally:
      sys.exit(0)

#interrompe l'esecuzione se da tastiera arriva la sequenza (CTRL + C)
```

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

#loop
try:
  while True:
    server.serve_forever()
except KeyboardInterrupt:
  pass

server.server_close()
```

3 Descrizione del Codice

3.1 Importazione delle Librerie

Le librerie `sys`, `signal`, `http.server` e `socketserver` vengono importate all'inizio dello script per gestire la porta, i segnali, il server HTTP e la gestione multi-thread.

3.2 Lettura della Porta

Il codice legge il numero della porta dalla riga di comando. Se non viene specificata alcuna porta, utilizza la porta di default 8080.

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

3.3 Creazione del Server

Il server viene creato utilizzando `ThreadingTCPServer` per gestire più richieste simultaneamente. Vengono impostate le proprietà `daemon_threads` per terminare i thread in modo pulito e `allow_reuse_address` per riutilizzare il socket.

```
server = socketserver.ThreadingTCPServer(('',port), http.server.
    SimpleHTTPRequestHandler )
server.daemon_threads = True
server.allow_reuse_address = True
```

3.4 Gestione del Segnale di Interruzione

Viene definita una funzione `signal_handler` per gestire l'interruzione del processo tramite Ctrl-C, chiudendo il server e terminando il processo.

```
def signal_handler(signal, frame):
    print( 'Exiting http server (Ctrl+C pressed)')
    try:
      if( server ):
        server.server_close()
    finally:
      sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)
```

3.5 Esecuzione del Server

Il server entra in un loop infinito per gestire le richieste in arrivo. Il loop viene interrotto se si riceve un'interruzione da tastiera (Ctrl-C).

```
try:
  while True:
    server.serve_forever()
except KeyboardInterrupt:
  pass

server.server_close()
```

4 Esecuzione del Programma

Per eseguire il programma, seguire questi passaggi:

1. Aprire un terminale e navigare alla directory in cui è salvato `server.py` `index.html` e `image.jpg`.
2. Eseguire il comando:

   ```
   python server.py <numero_porta>
   ```

   dove `<numero_porta>` è opzionale. Se non specificato, il server utilizzerà la porta 8080.

5 Funzionamento di un HTTP Server Multithread

Un server HTTP multithread è un server che può gestire più richieste contemporaneamente. Utilizza thread separati per ogni richiesta, migliorando l'efficienza e la capacità di risposta rispetto a un server a thread singolo.

5.1 Vantaggi

- **Prestazioni Migliorate**: Può gestire più richieste simultaneamente senza bloccare il server.
- **Scalabilità**: È in grado di scalare meglio sotto carichi elevati, rispondendo a più client contemporaneamente.
- **Reattività**: Migliora l'esperienza utente rispondendo rapidamente alle richieste.

5.2 Come Funziona

Quando una richiesta arriva al server, un nuovo thread viene creato per gestire quella richiesta specifica. Questo permette al server principale di continuare a ricevere nuove richieste mentre le richieste esistenti vengono elaborate nei thread separati. In questo script, `ThreadingTCPServer` viene utilizzato per abilitare il supporto multithread.