Documentazione del Server HTTP in Python

Cristian Morbidelli

June 6, 2024

Contents

1 Introduzione

Questo documento descrive in dettaglio il funzionamento di uno script Python che implementa un server HTTP semplice utilizzando il modulo `http.server` e il modulo `socketserver`. Lo script permette di avviare un server web multithreading che può servire file locali sulla rete.

2 Dettagli del Codice

2.1 Specificazione dell'Interprete

La riga seguente specifica quale interprete Python utilizzare:

```
#!/bin/env python
```

Questo *shebang* assicura che lo script venga eseguito con l'interprete Python corretto.

2.2 Importazione dei Moduli

Lo script inizia importando i moduli necessari:

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

- `sys`: Permette di interagire con il sistema.
- `signal`: Gestisce i segnali di sistema, come SIGINT.
- `http.server`: Fornisce la classe `SimpleHTTPRequestHandler` per gestire le richieste HTTP.
- `socketserver`: Fornisce la classe `ThreadingTCPServer` per creare un server multithreading.

2.3 Determinazione della Porta

Lo script verifica se un argomento è stato passato per specificare la porta del server:

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

- Se viene passato un argomento, questo viene usato come porta.
- Se nessun argomento è passato, viene usata la porta predefinita 8080.

2.4 Creazione del Server

Il server TCP multithreading è creato come segue:

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

- `('', port)`: Specifica che il server deve essere raggiungibile da qualsiasi indirizzo IP locale sulla porta specificata.
- `http.server.SimpleHTTPRequestHandler`: Utilizza questo gestore per servire richieste HTTP.

2.5 Configurazione del Server

Vengono impostate alcune opzioni del server:

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

- `daemon_threads`: Assicura che i thread siano demoni, quindi terminano con il processo principale.
- `allow_reuse_address`: Permette al server di riutilizzare l'indirizzo IP e la porta.

2.6 Gestione dei Segnali

Viene definita una funzione per gestire il segnale SIGINT:

```
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)
```

- Quando viene ricevuto un SIGINT (Ctrl+C), il server viene chiuso e lo script termina.

2.7 Esecuzione del Server

Il server viene avviato in un ciclo continuo:

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

- `server.serve_forever()`: Mantiene il server in esecuzione per gestire le richieste.
- L'eccezione `KeyboardInterrupt` è ignorata per permettere la gestione pulita dell'arresto tramite il gestore di segnale.
- `server.server_close()`: Chiude il server alla fine dell'esecuzione.

3 Avvio e Utilizzo dello Script

3.1 Avvio del Server

Per avviare il server, eseguire il seguente comando dal terminale:

```
$ python server.py [porta]
```

- `server.py`: Nome del file dello script.
- `porta` (opzionale): La porta su cui far girare il server. Se non specificata, viene usata la porta 8080.

3.2 Accesso al Server

Una volta avviato, il server può essere accesso tramite un browser web all'indirizzo:

`http://localhost:[porta]`

dove `[porta]` è la porta specificata al momento dell'avvio.

4 Considerazioni Aggiuntive

4.1 Sicurezza

Questo server è progettato per scopi di test e sviluppo. Non è sicuro per l'uso in produzione senza ulteriori misure di sicurezza, come l'autenticazione e la configurazione HTTPS.

4.2 Prestazioni

Il server utilizza il threading per gestire le richieste in modo parallelo, il che può migliorare le prestazioni rispetto a un server single-thread.

4.3 Espandibilità

Per aggiungere funzionalità personalizzate, è possibile estendere `http.server.SimpleHTTPRequestHandler` e sovrascrivere i metodi pertinenti per gestire le richieste HTTP come `GET` o `POST`.