Python: effettuare richieste HTTP concorrenti

Le richieste HTTP concorrenti sono un aspetto fondamentale dello sviluppo di applicazioni moderne che richiedono il recupero di dati da servizi web o l'interazione con API remote. Effettuare richieste HTTP in modo sequenziale può essere inefficiente e rallentare l'esecuzione del programma. Fortunatamente, Python offre diverse librerie e strumenti che consentono di effettuare richieste HTTP in modo concorrente, migliorando l'efficienza e le prestazioni dell'applicazione.

In questo articolo, esploreremo alcune delle librerie più comuni per effettuare richieste HTTP concorrenti in Python: asyncio e requests. Vedremo come utilizzare queste librerie per eseguire richieste HTTP in modo parallelo, migliorando così le prestazioni del nostro codice.

Prima di iniziare, assicuriamoci di avere installate le seguenti librerie:

- `aiohttp`: Una libreria per effettuare richieste HTTP asincrone in modo efficiente. È possibile installarla utilizzando `pip`.
- `requests`: Una libreria ampiamente utilizzata per effettuare richieste HTTP sincrone. Se non l'hai già installata, puoi farlo con pip.

Iniziamo con un esempio di come effettuare richieste HTTP concorrenti utilizzando `asyncio` e `aiohttp`. Supponiamo di voler effettuare richieste a tre URL diversi e recuperare il contenuto di ciascuno.

```
import asyncio
import aiohttp

async def fetch_url(url):
```

```
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()

async def main():
    urls = ['https://example.com',
'https://google.com', 'https://github.com']

    tasks = [fetch_url(url) for url in urls]
    results = await asyncio.gather(*tasks)

    for url, result in zip(urls, results):
        print(f'Response from {url}:
{result[:100]}...')

if __name__ == '__main__':
    asyncio.run(main())
```

In questo esempio, definiamo una funzione `fetch_url` che effettua una richiesta HTTP asincrona a un URL specifico utilizzando `aiohttp`. La funzione `main` crea una lista di URL da richiedere, quindi avvia in modo concorrente le richieste utilizzando `asyncio.gather`. Alla fine, viene stampato il contenuto dei primi 100 caratteri di ciascuna risposta.

Se preferisci utilizzare `requests`, puoi utilizzare il modulo `concurrent.futures` per eseguire richieste HTTP in modo concorrente. Ecco un esempio:

```
import concurrent.futures
import requests

def fetch_url(url):
```

```
    response = requests.get(url)
    return response.text

if __name__ == '__main__':
    urls = ['https://example.com',
'https://google.com', 'https://github.com']

    with concurrent.futures.ThreadPoolExecutor() as
executor:
        results = list(executor.map(fetch_url, urls))

    for url, result in zip(urls, results):
        print(f'Response from {url}:
{result[:100]}...')
```

In questo caso, utilizziamo un `ThreadPoolExecutor` per eseguire le richieste HTTP in thread separati. Questo permette di eseguire richieste HTTP in modo concorrente, anche se `requests` è sincrono per natura.

Conclusione

Le richieste HTTP concorrenti possono notevolmente migliorare le prestazioni delle tue applicazioni Python, consentendo di recuperare dati da servizi web in modo più efficiente. Scegli la libreria che meglio si adatta alle tue esigenze e ai requisiti del tuo progetto. Le librerie come `asyncio` e `aiohttp` sono particolarmente utili per applicazioni asincrone, mentre `requests` con `concurrent.futures` può essere utile quando si lavora con codice legacy sincrono.