21/09/2023

ASINCRONIA IN PYTHON

Campari Mirko

master.D

SPIEGAZIONE DELL'ASINCRONIA

L'ASINCRONIA È UN MODELLO DI PROGRAMMAZIONE CHE CONSENTE L'ESECUZIONE DI OPERAZIONI SENZA BLOCCARE L'ESECUZIONE PRINCIPALE DI UN'APPLICAZIONE.

IN ALTRE PAROLE, UN'OPERAZIONE ASINCRONA PUÒ AVVENIRE "IN BACKGROUND" MENTRE ALTRE OPERAZIONI CONTINUANO A FUNZIONARE, PERMETTENDO COSÌ UN UTILIZZO PIÙ EFFICIENTE DELLE RISORSE E UNA MIGLIORE RESPONSIVITÀ DELL'APPLICAZIONE.

master.D

QUESTO È PARTICOLARMENTE UTILE IN SCENARI COME L'ELABORAZIONE DI RICHIESTE WEB, LA LETTURA/SCRITTURA DA/A FILE, L'ACCESSO A DATABASE, ECC.

DOVE L'ATTESA PER UN'OPERAZIONE POTREBBE RALLENTARE INUTILMENTE TUTTO IL SISTEMA.

master.D

ASINCRONIA IN PYTHON

IN PYTHON, IL SUPPORTO ALL'ASINCRONIA È STATO INTRODOTTO CON IL MODULO ASYNCIO. QUESTO MODULO FORNISCE STRUMENTI E PRIMITIVE PER SCRIVERE CODICE ASINCRONO UTILIZZANDO LE PAROLE CHIAVE ASYNC E AWAIT.

- ASYNC: USATA PER DICHIARARE UNA FUNZIONE COME ASINCRONA.
- AWAIT: USATA ALL'INTERNO DI UNA FUNZIONE ASINCRONA PER CHIAMARE UN'ALTRA FUNZIONE ASINCRONA E ASPETTARE CHE TERMINI.

master.D

ASINCRONIA IN PYTHON

IN PYTHON, IL SUPPORTO ALL'ASINCRONIA È STATO INTRODOTTO CON IL MODULO ASYNCIO. QUESTO MODULO FORNISCE STRUMENTI E PRIMITIVE PER SCRIVERE CODICE ASINCRONO UTILIZZANDO LE PAROLE CHIAVE ASYNC E AWAIT.

- ASYNC: USATA PER DICHIARARE UNA FUNZIONE COME ASINCRONA.
- AWAIT: USATA ALL'INTERNO DI UNA FUNZIONE ASINCRONA PER CHIAMARE UN'ALTRA FUNZIONE ASINCRONA E ASPETTARE CHE TERMINI.

master.D

ESEMPIO DI CODICE PYTHON ASINCRONO:

SUPPONIAMO DI VOLER SIMULARE DUE FUNZIONI, UNA CHE RAPPRESENTA UN'ATTESA DI 2 SECONDI E UN'ALTRA DI 3 SECONDI, E VOGLIAMO ESEGUIRLE IN MANIERA ASINCRONA:

```
1. import asyncio
2. async def attendi_2_secondi():
3.     print("Inizio attesa di 2 secondi...")
4.     await asyncio.sleep(2)  # Simula un'attesa di 2 secondi
5.     print("Fine attesa di 2 secondi!")
6.
7. async def attendi_3_secondi():
8.     print("Inizio attesa di 3 secondi...")
9.     await asyncio.sleep(3)  # Simula un'attesa di 3 secondi
10.    print("Fine attesa di 3 secondi!")
11.
12. async def main():
13.    # Esegue entrambe le funzioni asincrone contemporaneamente
14.    await asyncio.gather(attendi_2_secondi(), attendi_3_secondi())
15. asyncio.run(main())
```

master.D

NELL'ESEMPIO SOPRA, LE FUNZIONI **ATTENDI_2_SECONDI** E **ATTENDI_3_SECONDI** SONO DICHIARATE COME FUNZIONI ASINCRONE CON **ASYNC**.

ALL'INTERNO DI ESSE, UTILIZZIAMO **AWAIT** PER ATTENDERE L'ESECUZIONE DI **ASYNCIO.SLEEP**, CHE È UNA SIMULAZIONE ASINCRONA DELL'ATTESA.

LA FUNZIONE **MAIN** ESEGUE ENTRAMBE LE FUNZIONI ASINCRONE CONTEMPORANEAMENTE UTILIZZANDO **ASYNCIO.GATHER**.

master.D

ESEGUENDO QUESTO CODICE, VEDRAI CHE ENTRAMBE LE FUNZIONI INIZIANO QUASI CONTEMPORANEAMENTE E TERMINANO DOPO I LORO RISPETTIVI PERIODI DI ATTESA, DIMOSTRANDO L'EFFICACIA DELL'ESECUZIONE ASINCRONA.

master.D

CARATTERISTICHE DELL'ASINCRONIA IN PYTHON:

- NON-BLOCCANTE: IL CODICE ASINCRONO PERMETTE L'ESECUZIONE DI OPERAZIONI SENZA BLOCCARE L'INTERA APPLICAZIONE, PERMETTENDO L'ESECUZIONE DI ALTRE OPERAZIONI CONTEMPORANEAMENTE.

- COROUTINES: UTILIZZANDO ASYNC DEF, PYTHON INTRODUCE LE COROUTINES, CHE SONO FUNZIONI CHE POSSONO "METTERSI IN PAUSA" E PERMETTERE L'ESECUZIONE DI ALTRE COROUTINES, PER POI RIPRENDERE IN SEGUITO DA DOVE SI ERANO INTERROTTE.

- EVENT LOOP: IL CUORE DELL'ASINCRONIA IN PYTHON È L'EVENT LOOP, FORNITO DA ASYNCIO. GESTISCE L'ESECUZIONE DI DIVERSE OPERAZIONI ASINCRONE, PIANIFICANDO LA LORO ESECUZIONE.

master.D

- FUTURES E TASKS: SONO OGGETTI SPECIALI CHE RAPPRESENTANO IL RISULTATO DI UN'OPERAZIONE ASINCRONA. UN TASK È UNA SUBCLASS DI FUTURE CHE RAPPRESENTA UN'OPERAZIONE ASINCRONA PROGRAMMATA PER ESECUZIONE.

- ASYNC/AWAIT: QUESTE PAROLE CHIAVE SONO CENTRALI NELLA PROGRAMMAZIONE ASINCRONA IN PYTHON. ASYNC DICHIARA UNA FUNZIONE COME ASINCRONA, MENTRE AWAIT VIENE UTILIZZATO PER "ATTENDERE" IL COMPLETAMENTO DI UN'OPERAZIONE ASINCRONA.

- LIBRERIE DI TERZE PARTI: ESISTONO MOLTE LIBRERIE DI TERZE PARTI CHE SFRUTTANO E ESTENDONO LE CAPACITÀ ASINCRONE DI PYTHON, COME AIOHTTP PER LE RICHIESTE HTTP ASINCRONE E AIOMYSQL/AIOPG PER L'INTERAZIONE ASINCRONA CON I DATABASE.

master.D

SCRIPT CHE ESEGUE RICHIESTE HTTP ASINCRONE: PER FARE QUESTO, UTILIZZEREMO LA LIBRERIA AIOHTTP.

```
1. import aiohttp
2. import asyncio
3.
4. async def fetch_url(session, url):
5.     async with session.get(url) as response:
6.         return await response.text()
7.
8. async def main():
9.     async with aiohttp.ClientSession() as session:
10.         html = await fetch_url(session,
    'https://www.example.com')
11.         print(html[:100])  # stampa i primi 100 caratteri
12.
13. asyncio.run(main())
```

master.d

Script che simula operazioni asincrone:

```
1. import asyncio
2.
3. async def operazione_asincrona(num):
4.     print(f"Inizio operazione {num}")
5.     await asyncio.sleep(2)  # Simula un'operazione che impiega 2
   secondi
6. print(f"Fine operazione {num}")
7. async def main():
8.     tasks = [operazione_asincrona(i) for i in range(3)]
9.     await asyncio.gather(*tasks)
10.
11. asyncio.run(main())
```