파이썬

(주)인피닉스 – 강호용 연구원

Infinyx

쓰레드, 메시지큐

Infinyx
THINK LIFE SYNC AI

쓰레드(Thread) 개요

쓰레드는 프로그램 안에서 실행되는 흐름의 단위입니다. 멀티 태스킹과 동시성은 쓰레드와 관련된 중요한 개념입니다.

멀티태스킹

여러 작업이 동시에 실행되는 것처럼 보이도록 하는 개념입니다. 이것은 실제로 CPU의 처리 속도가 빠르기 때문에 여러 작업이 번 갈아가며 실행되는 것처럼 보이게 합니다.

동시성

여러 작업이 동시에 실행되는 것처럼 보이지만, 실제로는 한 순간에 하나의 작업만 실행된다는 개념입니다. 이는 여러 작업이 서로 번 갈아가며 실행되는 것처럼 보이지만, 실제로는 동시에 실행되는 것이 아닙니다.

쓰레드 생성 및 관리

- 쓰레드를 생성하고 관리하는 것은 파이썬의 threading 라이브러리를 사용하여 가능합니다. 이 라이브러리는 쓰레드를 만들고 실행하는 데 필요한 기능을 제공합니다.

쓰레드 생성 간단한 예시)

```
import threading


1 usage
def my_function():
    print("This is my thread!")

my_thread = threading.Thread(target=my_function)

my_thread.start()
```

Thread 객체 생성

쓰레드 시작

쓰레드 생성 및 관리

- 쓰레드를 생성하고 관리하는 것은 파이썬의 threading 라이브러리를 사용하여 가능합니다. 이 라이브러리는 쓰레드를 만들고 실행하는 데 필요한 기능을 제공합니다.

쓰레드 종료 간단한 예시) : 작업을 완료하거나 중단시키는 등의 방법으로 종료될 수 있음

```
import threading
import time

1 usage
def my_function():
    while running:
        print("This is my thread!")
        time.sleep(1)

running = True
my_thread = threading.Thread(target=my_function)

my_thread.start()

running = False
```

running 플래그를 통해 쓰레드 종료 조건 설정

쓰레드가 빠르게 종료되는 것을 방지 하기위한 1초 대기

쓰레드 생성

어떤 조건에서든 running 플래그를 False로 설정하여 쓰레드 종료

 # Python Thread 관리에서 중요하게 사용되는 메서드 join(), start(), stop(), Lock() 소개

➢ start()

- **역할**: 스레드를 시작하는 데 사용됩니다.
- **동작**: 이 메서드를 호출하면 스레드가 생성되고, 해당 스레드에서 run() 메서드가 실행됩니다. 즉, 실제로 스레드가 작업을 수행하기 시작합니다.
- **주의사항**: start() 메서드는 한 번만 호출할 수 있습니다. 스레드가 이미 실행 중인 경우 다시 start()를 호출하면 에러가 발생합니다.

 # Python Thread 관리에서 중요하게 사용되는 메서드 join(), start(), stop(), Lock() 소개

➢ join()

- **역할**: 메인 스레드(main thread)가 다른 스레드의 작업을 기다리도록 합니다.
- **동작**: 메인 스레드에서 join()을 호출하면 메인 스레드는 해당 스레드가 종료될 때까지 기다립니다. 즉, 다른 스레드의 작업이 완료될 때까지 메인 스레드가 차단(blocked)됩니다.
- **활용**: 주로 작업이 완료될 때까지 기다렸다가 그 결과를 이용하고자 할 때 사용됩니다.

 # Python Thread 관리에서 중요하게 사용되는 메서드 join(), start(), stop(), Lock() 소개

- stop()
  - **주의**: Python 3.3 이전에 존재했으나 현재는 deprecated(사용 중단된) 상태입니다. 현재의 권장 방식은 스레드를 종료하기 위해 공식적인 메서드가 제공되지 않고 있습니다. 스레드를 안전하게 종료하는 방법은 해당 스레드가 종료할 수 있는 조건을 설정하고, 해당 조건이 충족되면 스레드가 종료되도록 하는 것입니다.

 # Python Thread 관리에서 중요하게 사용되는 메서드 join(), start(), stop(), Lock() 소개

- stop()
  - **주의**: Python 3.3 이전에 존재했으나 현재는 deprecated(사용 중단된) 상태입니다. 현재의 권장 방식은 스레드를 종료하기 위해 공식적인 메서드가 제공되지 않고 있습니다. 스레드를 안전하게 종료하는 방법은 해당 스레드가 종료할 수 있는 조건을 설정하고, 해당 조건이 충족되면 스레드가 종료되도록 하는 것입니다.

Python Thread 관리에서 중요하게 사용되는 메서드 join(), start(), stop(), Lock() 소개

➢ Lock()

파이썬의 threading 모듈에서 제공하는 동기화 메커니즘 중 하나입니다.
이를 사용하여 여러 스레드가 동시에 공유 자원에 접근하는 것을 제어할 수 있습니다.

Lock의 역할

- **상호배제(Mutual Exclusion)**: 한 번에 하나의 스레드만 Lock을 소유하고, 나머지 스레드들은 해당 자원에 접근하지 못하도록 합니다.
- **Lock 획득 및 해제**: acquire() 메서드를 사용하여 Lock을 획득하고, release() 메서드를 사용하여 Lock을 해제합니다.
- **동기화**: 여러 스레드가 동시에 공유 자원에 접근하는 것을 조정하여, 데이터 일관성을 유지하고 경쟁 조건(race condition)을 방지합니다.

Python 에서 쓰레드 사용방법 'threading 라이브러리 활용'

- threading 라이브러리를 사용하여 쓰레드 만들고 실행 방법을 보여주는 간단한 코드

```
import threading
import time

1 usage
def print_numbers():
    for i in range(5):
        print(i)
        time.sleep(1)

1 usage
def print_letters():
    for char in 'ABCDE':
        print(char)
        time.sleep(1)
```

단순하게 숫자를 0~4까지 출력하는 함수

단순하게 'ABCDE' → 단어 하나씩 출력 예) A B C D E

Python 에서 쓰레드 사용방법 'threading 라이브러리 활용'

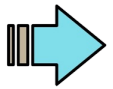
- threading 라이브러리를 사용하여 쓰레드 만들고 실행 방법을 보여주는 간단한 코드

```
# Thread 객체 생성
numbers_thread = threading.Thread(target=print_numbers)
letters_thread = threading.Thread(target=print_letters)

# 쓰레드 시작
numbers_thread.start()
letters_thread.start()

# 모든 쓰레드가 실행을 마치기를 기다림
numbers_thread.join()
letters_thread.join()

print("모든 쓰레드가 종료됨")
```



출력 결과
number_thread → 0
letters_thread → A
letters_thread → B
number_thread → 1
letters_thread → C
number_thread → 2
letters_thread → D
number_thread → 3
number_thread → 4
letters_thread → E

모든 쓰레드가 종료

쓰레드가 동시에 실행되는 것을 확인하고, 각각이 독립적으로 작업하는 것을 보실 수 있을 거예요!

큐(Queue) 소개

Queue Data Structure

큐는 데이터를 저장하고 처리하는 추상적인 자료 구조로, 데이터를 일시적으로 저장하는데 사용됩니다.

큐(Queue)를 이용한 Thread 핸들링

```python
import queue
import threading
import time

# Lock 객체 생성
print_lock = threading.Lock()

1 usage
def producer(q):
    for i in range(5):
        time.sleep(1)
        q.put(i)
        with print_lock:
            print(f"Produced: {i}")

1 usage
def consumer(q):
    while True:
        item = q.get()
        if item is None:
            break
        with print_lock:
            print(f"Consumed: {item}")
        time.sleep(2)
```

q.put(i)를 사용하여 큐 데이터를 추가합니다.
print_lock을 사용하여 스레드 간에 출력이 교차되지 않도록 출력 동기화
값은 0부터 4까지의 큐에 넣습니다.

무한 루프를 통해 큐에서 데이터를 가져와 출력
큐에서 None을 받으면 루프를 탈출합니다.

큐(Queue) 를 이용한 Thread 핸들링

```
q = queue.Queue()

producer_thread = threading.Thread(target=producer, args=(q,))
consumer_thread = threading.Thread(target=consumer, args=(q,))

producer_thread.start()
consumer_thread.start()

producer_thread.join()
q.put(None)
consumer_thread.join()

print("모든 작업 완료")
```

스레드 생성과 시작

producer_thread.join()을 통해 생산자 스레드가 모든 작업을 마치면 None을 큐에 넣어 소비자 스레드에게 종료 신호를 보냄

이 코드는 생산자와 소비자 패턴을 사용하여 스레드 간 데이터를 안전하게 주고받는 예시입니다.
생산자는 데이터를 생성하고 큐에 넣고, 소비자는 큐에서 데이터를 꺼내어 소비합니다.

Python 메시지 큐 역할

파이썬에서 메시지 큐는 다양한 목적으로 활용됩니다. 주로 queue 모듈을 사용하여 구현하며, 다음과 같은 역할을 합니다.

- **스레드 간 통신**: queue.Queue를 사용하여 스레드 간에 안전하게 데이터를 주고받을 수 있습니다. 이를 통해 다수의 스레드가 안전하게 데이터를 교환하고 작업을 조정할 수 있습니다.
- **이벤트 처리**: 이벤트가 발생하는 것을 감지하고, 해당 이벤트에 대한 처리를 메시지 큐를 통해 수행합니다. 이를 통해 이벤트에 따른 작업을 비동기적으로 처리할 수 있습니다.
- **프로듀서-컨슈머 패턴**: 메시지 큐는 프로듀서가 데이터를 생성하고, 컨슈머가 그 데이터를 소비하는 패턴에서 중요한 역할을 합니다. 큐를 통해 데이터를 안전하게 교환하고 처리할 수 있습니다.

Python 메시지 큐 역할

파이썬에서 메시지 큐는 다양한 목적으로 활용됩니다. 주로 queue 모듈을 사용하여 구현하며, 다음과 같은 역할을 합니다.

- **비동기 처리 및 작업 조율**: 비동기 작업을 수행할 때 메시지 큐를 사용하여 작업의 실행 순서를 관리하고, 결과를 수집하는 데 활용됩니다. 이를 통해 여러 작업을 병렬로 처리하고 조율할 수 있습니다.
- **이벤트 기반 아키텍처**: 이벤트가 발생하면 해당 이벤트를 큐에 추가하여 필요한 시간에 처리할 수 있도록 합니다. 이를 통해 시스템이 이벤트에 따라 동적으로 동작할 수 있습니다.
- **분산 시스템 통신**: 여러 노드 또는 서버 간에 데이터를 안전하게 교환하고 처리하는 데 사용됩니다. 이를 통해 분산 시스템 간에 효율적인 통신이 가능해집니다.

메시지 큐 패턴

➢ Python에서 메시지 큐 패턴은 프로듀서(Producer)와 컨슈머(Consumer) 사이의 데이터 교환을 위해 사용됩니다.

메시지 큐 패턴의 주요 구성 요소

- **Queue(큐)**:
  파이썬에서는 queue 모듈을 사용하여 구현됩니다. 큐는 데이터를 안전하게 저장하고 스레드 또는 프로세스 간에 전달합니다. 주로 FIFO(First-In, First-Out) 방식으로 작동합니다.

- **프로듀서(Producer)**:
  데이터를 생성하고 큐에 넣는 역할을 합니다. 데이터를 생산하고 큐에 삽입하여 컨슈머가 사용할 수 있도록 합니다.

- **컨슈머(Consumer)**:
  큐에서 데이터를 가져와 소비하는 역할을 합니다. 큐에서 데이터를 추출하여 필요한 처리를 수행합니다.


감사합니다.