Thread-Safety

CS-challenge 1주차

배서은 2023.04.20

멀티 스레드 프로그래밍

- 멀티스레드 프로그래밍은 하나의 프로세스에서 여러 개의 스레드를 만들어 자원의 생성과 관리의 중복을 최소화하는 것이다.
- 장점
    - 멀티 프로세스에 비해 메모리 자원소모가 줄어든다.
    - Heap 영역을 통해서 스레드 간의 통신이 가능하기 때문에 프로세스 간의 통신이 간단해진다.
    - 스레드의 컨텍스트 스위칭은 프로세스의 컨텍스트 스위칭보다 훨씬 빠르다.
- 단점
    - 힙 영역에 있는 자원을 사용할 때 동기화를 해야한다.
    - 동기화를 위해서 락을 과도하게 사용하면 성능 저하가 발생할 수도 있다.
    - 하나의 스레드가 비정상적으로 동작하면 다른 스레드도 영향을 받아 종료하게 될 수도 있다.

스레드 안전(Thread-Safety)

- 스레드 안전(Thread-Satety)란 멀티 스레드 프로그래밍에서 일반적으로 어떤 함수나 변수, 혹은 객체가 여러 스레드로부터 동시에 접근이 이루어져도 프로그램의 실행에 문제가 없는 것을 말한다.
- 하나의 함수가 한 스레드로부터 호출되어 실행 중일 때, 다른 스레드가 그 함수를 호출하여 동시에 함께 실행되더라도 각 스레드에서의 함수의 수행 결과가 옳바르게 나오는 것을 말한다.

> Thread-sate하다라는 의미는 두 개 이상의 스레드가 race condition에 들어가거나 같은 객체에 동시에 접근해도 연산 결과는 정합성이 보장될 수 있게 메모리 가시성이 확보된 상태이다.

스레드 안전(Thread-Safety)의 여부 판단 방법

- 다음과 같이 세 가지를 통해 thread-safe한 상태인지 판단할 수 있다.

> 1. 전역 변수나 힙, 파일과 같이 여러 스레드가 동시에 접근할 수 있는 자원을 사용하는가?
> 2. 핸들과 포인터를 통한 데이터의 간접 접근이 가능한가?
> 3. 부수 효과를 가져오는 코드가 있는가?

S

스레드 안전과 동기화가 다른가요?

스레드 안전성과 동기화는 관련된 개념이지만 같은 것은 아닙니다.

스레드 안전성은 동작의 정확성과 일관성을 보장하는 방식으로 여러 스레드 또는 여러 스레드의 동시 액세스를 처리하는 소프트웨어 구성 요소 또는 시스템의 기능을 나타냅니다. 스레드로부터 안전한 구성 요소 또는 시스템은 경합 상태, 교착 상태 또는 기타 동시성 관련 문제를 일으키지 않고 여러 스레드에서 동시에 사용할 수 있습니다.

반면에 동기화는 소프트웨어 시스템에서 스레드 안전성을 달성하는 데 사용되는 특정 기술 또는 메커니즘입니다. 동기화에는 변수 또는 데이터 구조와 같은 공유 리소스에 대한 액세스를 조정하여 한 번에 하나의 스레드만 액세스할 수 있도록 하고 공유 상태가 일관되고 올바르게 유지되도록 합니다.

즉, 동기화는 스레드 안전성을 달성하는 수단이지만 유일한 방법은 아닙니다. 스레드 안전성을 달성하기 위한 다른 기술로는 불변성, 스레드 로컬 저장소 및 메시지 전달이 있습니다.

요약하면 스레드 안전성은 동기화를 비롯한 다양한 수단을 통해 달성할 수 있는 소프트웨어 시스템의 속성이며, 동기화는 스레드 안전성을 달성하는 데 사용되는 특정 메커니즘입니다.

스레드 안전을 지키기 위한 4가지 방법

- Mutual Exclusion (상호 배제)
- Atomic Operation (원자 연산)
- Thread-Local Storage (쓰레드 지역 저장소)
- Re-Entrancy (재진입성)

1) Mutual Exclusion(상호 배제)

- 공유 자원에 하나의 Thread만 접근할 수 있도록 세마포어 / 뮤텍스로 락을 통제하는 방법이다.
- 일반적으로 많이 사용하는 방법이다.
- 적용 예시
  - Python은 Thread Safe하게 메모리를 관리하지 않아 GIL(Global Interpreter Lock)을 통해 Thread Safe를 보장한다.
  - Python에서 threading.lock이 있는데 threading.lock을 acqurie하면 해당 쓰레드만 공유 데이터에 접근할 수 있고 lock을 release해야만 다른 쓰레드에서 공유 데이터에 접근할 수 있다.

Mutual Exclusion

1. Semaphore

- 대표적인 동기화 도구
- 한국어뜻 : 수신호

> [세마포어의 두가지 동작]
>
> P (test) : acquire()
>
> - P 동작 전개
>
> 1. 세마포어의 Integer 변수의 값 -1 ;
> 2. Integer 변수의 값이 음수 이면 -> 대기큐에 **현재** 프로세스를 넣고 **블락**한다.
>
> V (increment) : release()
>
> - V 동작 전개
>
> 1. 세마포어의 Integer 변수의 값 +1 ;
> 2. Integer 변수의 값 0이거나 음수인 경우 -> 대기큐에서 **하나의 P 프로세스**를 커내서 **릴리즈**한다.

- **세마포어 스트럭쳐**

```
class Semaphore {
  int value;      // number of permits
  Semaphore(int value) {
    // ...
  }
  void acquire() {
    value--;
    if (value < 0) {
      // add this process/thread to list
      // block
    }
  }
  void release() {
    value++;
    if (value <= 0) {
      // remove a process P from list
      // wakeup P
    }
  }
}
```

JAVA 에서 Thread-safe하게 설계하는 법

- java.util.concurrent 패키지 하위의 클래스를 사용한다.
- 인스턴스 변수를 두지 않는다.
- Singleton 패턴을 사용한다.
- 동기화 (Syncronized) 블럭에서 연산을 수행한다.

- Spring 에서는…
- 이런 싱글톤 패턴을 통해 Bean에 대한 관리를 하는데 너무 복잡해지는 것을 방지하기 위해, Singleton Registry인 Application Context를 가지고 있다.

싱글톤 패턴

- 싱글톤 패턴이란 애플리케이션이 시작되고 하나의 클래스 인스턴스를 보장해서 전역적인 접근점을 제공하는 패턴을 말한다.
- 장점
  - 전역 변수를 사용하지 않기 때문에 디버깅이 쉽고, 네임 스페이스가 망가지는 일이 없다.
  - 유일하게 존재하는 인스턴스로의 접근으로 통제가 가능하며, 데이터 공유가 쉬워진다.
- 단점
  - 싱글톤 인스턴스가 너무 많은 일을 하거나 너무 많은 데이터를 가질 경우 결합도가 높아진다.
  - 멀티스레드 환경에서 동기화 처리를 하지 않으면 여러 개가 생성될 수 있다.

Thread-safe 싱글톤 인스턴스

2. 멀티스레드 환경에서 안전한 싱글톤 인스턴스 만들기

- 게으른 초기화(Syncronized 블록 사용)
  - 속도가 너무 느려서 권장 되지 않음
- Double-Check Locking
  - if문으로 존재 여부 체크하고, Syncronized 블록 사용하는 방법
  - 어느정도 문제 해결은 가능하지만 완벽하지 않음
- Holder에 의한 초기화
  - 클래스 안에서 클래스(Holder)를 두어 JVM Class Loader 매커니즘을 이용한 방법
  - 개발자가 직접 동기화 문제를 해결하기 보다는 JVM의 원자적인 특성을 이용해 초기화의 책임을 JVM으로 이동하는 방법
  - 일반적으로 Singleton을 이용하는 방법

Spring Bean 의 Thread-safe

-> Non Thread-safe Object가 주입된다면 해당객체는 쓰레드에 안전하지않다.

(1) Builder Pattern 의 사용

간단하면서도 Tricky 한 방식으로 Builder Pattern 을 사용하면 좋다.

Builder 패턴을 이용하면 객체의 Setter 를 정의하지 않은 상태에서 생성자만으로 객체의 Mutation 을 관리할 수 있으며, Spring 의 Bean 들은 Container 에 의해 life cycle 관리가 위임된다.

따라서 Builder Pattern 을 통해 Set 을 관리하면 간단하면서도 명확히 Thread Safe 를 구현할 수 있다.

(2) Stateless Bean

Bean 을 상태값과 무관하게 동작할 수 있는 Bean 으로 설계하는 것이다.

Bean 이 특정 상태를 나타내는 변수를 계속 메모리에 들고 상주하는 형태가 아닌, 가장 단순한 형태의 도메인 모델로 사용이 추천된다.

(3) Lock the beans

가장 최후의 수단으로 여겨야 하는 방법으로 Bean 에 대해 Thread Safe 하게 설계를 하는 것이다. Spring 은 Lower level Library 들을 통해 Bean 단위의 Lock 을 지원하고 있으며 이를 통해 병렬처리에 있어서 동시성 문제의 해결이 필요하다.

물론 실제로 Safe 하지 않은 상황을 고려해야할 경우는 많지 않지만, 각기 다른 Request 를 공통으로 분류해야한다거나, 내부에서 Internal Thread 를 구동하는 경우에는 반드시 신경써보자.

감사합니다

https://developer-ellen.tistory.com/205