Chapter 11. 동시성

2021-05-03 김정수

Contents

1. Item 78. 공유 중인 가변 데이터는 동기화해 사용하라

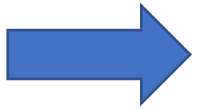
Syncronized 키워드

- 메서드나 블록을 한 번에 한 스레드씩 수행하도록 보장함. (배타적 수행)
- 동기화 없이는 한 스레드가 만든 변화를 다른 스레드에서 확인하지 못하게 한다.

⇒ 배타적 실행 + 스레드 간 안정적인 통신에 필요하다.

데이터가 원자적이라도 컴파일러의 코드 최적화 때문에 의도했던 대로 코드가 동작하지 않을 수도 있음.

```
// 코드 최적화 전 thread pooling
while(!stopRequested)
    i++;
```



```
// 코드 최적화 후
if(!stopRequested)
    while(true)
        i++;
```

1. Item 78. 공유 중인 가변 데이터는 동기화해 사용하라

해결 방법 1: synchronized 키워드 사용

공유 변수인 stropRequested에 접근하는 코드를 묶어 synchronized 키워드로 동기화하게 한다.
⇒ 쓰기와 읽기 모두!!

해결 방법 2: volatile 로 선언

volatile 한정자는 배타적 수행과는 상관없지만 항상 가장 최근에 기록된 값을 읽게 됨을 보장한다.
=> 원자적으로 접근할 때만 안전하다.

해결 방법 3. 가변 데이터는 공유하지 않는다.

2. Item 79. 과도한 동기화는 피하라

- 동기화 메서드나 동기화 블록 안에서는 제어를 절대로 클라이언트에 양도하지 말자.

- 클라이언트에게 양도되는 메서드를 alien method라 한다.
  (재정의할 수 있는 메서드, 클라이언트가 넘겨준 함수 객체)

⇒ alien method 호출을 동기화 블록 바깥으로 옮기자.

- 열린 호출(open call): 동기화 영역 바깥에서 호출되는 alien method

- 동기화는 가상머신의 코드 최적화를 제한한다. (지연 시간에 영향 미침)

2. Item 79. 과도한 동기화는 피하라

Class 내부 동기화 방법

- 락 분할 (lock splitting)
서로 다른 락을 사용해 여러 개의 독립적인 상태 변수를 각자 묶어두게 한다.

- 락 스트라이핑 (lock striping)
독립적인 객체를 여러 가지 크기의 block으로 묶고, block 단위로 락을 나눈다.

- 비차단 동시성 제어 (nonblocking concurrency control)
스레드 경쟁 상황에서 상호 배제에 의한 지연을 방지한다. 대부분의 상황에서 처리량이 높게 나타난다.
Non-blocking 알고리즘을 사용해야 한다.

2. Item 79. 과도한 동기화는 피하라

```java
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicStampedReference;

public class NonblockingTemplate {

    public static class IntendedModification {
        public AtomicBoolean completed =
                new AtomicBoolean(false);
    }

    private AtomicStampedReference<IntendedModification>
        ongoingMod =
            new AtomicStampedReference<IntendedModification>(null, 0);

    // 자료구조의 상태를 선언한다.


    public void modify() {
        while(!attemptModifyASR());
    }
```

```java
public boolean attemptModifyASR(){

    boolean modified = false;

    IntendedModification currentlyOngoingMod =
    ongoingMod.getReference();
    int stamp = ongoingMod.getStamp();

    if(currentlyOngoingMod == null){
        //변경 예정을 만들기 위한 자료구조 상태를 복제한다.

        //변경 예정을 준비한다.
        IntendedModification newMod =
        new IntendedModification();

        boolean modSubmitted =
            ongoingMod.compareAndSet(null, newMod, stamp, stamp + 1);

        if(modSubmitted){

            //컴페어 스왑 연산을 통해 변경을 완료한다.
            //note: 다른 쓰레드가 컴페어 스왑 연산의 완료를 도울 수 있기 때문에,
            //CAS 는 실패할 수 있음.

            modified = true;
        }

    } else {
        //진행중인 변경 작업을 완료하려고 시도하므로, 이 쓰레의 접근을 허용하기 위해
        //자료구조는 해제된다.

        modified = false;
    }

    return modified;
}
```

3. Item 81. wait와 notify보다는 동시성 유틸리티를 애용하라

wait : 스레드가 어떤 조건이 충족될 때까지 기다리는 데 사용 (wait loop 관용구를 사용)
notify : wait 상태인 다른 스레드를 깨운다. notifyAll은 모든 대기 스레드 중 일부만 조건을 충족해도 깨운다.

동시성 유틸리티
- 실행자 프레임워크 : synchronized

- 동시성 컬렉션 : 표준 Collection 인터페이스 + 동시성
  - 동시성 수준을 매개변수로 받는다.
  - Map -> ConcurrentHashMap
  - Queue -> BlockingQueue

- 동기화 장치 : 스레드가 다른 스레드를 기다릴 수 있게 한다.
  - CountDownLatch : 일회성 장벽. 동기화 수준을 매개변수로 받는다.
  - Semaphore
  - Phaser

4. Item 82. 스레드 안정성 수준을 문서화하라

- 불변(Immutable) @Immutable
: 인스턴스가 상수와 같을 때. (ex. String, Long, BigInteger)

- 무조건적 스레드 안전(unconditionally thread-safe) @ThreadSafe
: 수정될 수 있으나 내부에서 충분히 동기화 되었을 때. (ex. ConcurruntHashMap)

- 조건부 스레드 안전(conditionally thread-safe) @ThreadSafe
: 수정될 수 있고, 일부 메서드만 외부에서 동기화 시켜 줘야 한다. (ex. Collections.synchronized 래퍼 매서드가 반환한 인스턴스)

- 스레드 안전하지 않음(not thread-safe) @NotThreadSafe
: 수정될 수 있으며, 각각의 메서드 호출을 외부 동기화해야 한다. (ex. ArrayList, HashMap)

- 스레드 적대적(thread-hostile)
: 모든 메서드 호출을 외부에서 동기화하더라도 스레드 안전하지 않다. 되도록이면 쓰지 말자.

References

- https://stackoverflow.com/questions/2824225/what-is-non-blocking-concurrency-and-how-is-it-different-than-normal-concurren/8918074
- https://aroundck.tistory.com/4529
- https://parkcheolu.tistory.com/33

Q&A