22.12.19

📌GC(가비지 컬렉터)

동작 방식

Parrall GC나 CMS, G1 알고리즘의 세부동작은 아래 동작을 따른다.

1. 객체를 생성해서 Eden에 할당한다.
2. Eden이 다 차면 Minor GC가 발생한다. (JVM의 suspend상태, STW)

— GC Root로부터 unreachable 객체와 reachable 객체로 구분한다(mark작업)

— 살아남은 객체는 Suvivor 영역 중 하나로 옮겨진다.(복사)

→ 다른 영역으로 살아있는 객체를 옮김으로써 mark와 copy를 동시에 할 수 있다는 이점이 있다. copy할때, mark할 때 각각 STW가 발생

→ age값이 증가한다.

— 이전 영역은 비워진다 (sweep)

— JVM의 suspend 상태가 해제된다.

3. Age 값의 초과

Minor GC 과정을 많이 반복하면, Age 값이 설정값(MaxTenuringThreshold)을 초과하여 old영역으로 이동한다.

4. Old 영역이 다 차면 Major GC가 발생한다.

GC root

- Class Loader에 의해 로딩된 클래스
- 지역변수, 매개변수
- 현재 활성화된 쓰레드
- 정적 변수
- JNI reference
  - JNI 메소드의 지역변수/ 매개변수

    - 전역 JNI 참조변수
- Monitor로 사용된 객체

GC 종류

1. **Serial GC**

- 하나의 CPU로 young 영역과 old 영역을 연속적으로 처리한다. (싱글쓰레드방식). 따라서 young 영역 및 old 영역에 대한 GC는 모두 STW가 발생한다.
- mark,sweep,compact 알고리즘을 사용한다.
    - mark : old 영역에서 reachable한 객체를 식별
    - sweep : 살아있는 객체만 남기고 unreachable한 객체 제거.
    - compact : **marking 된 객체들을 메모리 영역의 처음부터 몰라넣는다. (메모리 단편화의 문제 제거)**
- young 영역: mark-copy

  old 영역 : mark-sweep-compact

객체를 새로 생성할 때는 그 객체가 점유할 메모리 공간은 반드시 연속적이어야 한다. 따라서 메모리 단편화가 심해질수록, 전체 가용 메모리 공간은 여유로운데도 불구하고 객체 생성에 실패할 수 있다.

2. **Paraller GC → Java 7,8 사용시 default**

- young 영역: mark-copy

  old 영역 : mark-sweep-compact
- 여러 쓰레드가 병렬로 수행되어서 Serial GC보다 걸리는 시간이 짧다.
- Hotspot JVM에서는 여러 쓰레드에 의해 병렬로 GC가 수행될 경우, young 영역에서 old로 객체를 이동시킬때의 동기화문제를 피하기 위해 Paraller allocation buffer(PLAB)를 사용한다. PLAB은 각 GC 쓰레드가 받는 old영역에 객체 할당 공간으로 이를 통해 쓰레드가 경합하지 않고 바로 old영역에 객체를 이동시킬 수 있다.

3. **Concurrent mark and sweep**

- young 영역 : mark - copy

  → STW 발생, 멀티쓰레드를 통해 병렬적으로 수행

- old 영역 : mark- weep

  → compact 수행없이 객체를 할당할 수 있는 공간을 관리하는 구조(free-list)를 따로 관리한다.

동작순서

1) initial mark : GC Root가 참조하는 객체만 마킹 (STW발생)

2) concurrent mark : 참조하는 객체를 따라가며 지속적으로 마킹 (STW발생X)

3) remark : concurrent mark에서 수정된 사항 마킹(STW 발생)

4) concurrent sweep : 접근 X되는 객제 제거 (STW 발생X)

4. G1 → **JDK 7에서 처음 등장한뒤, JDK 9부터 default, 10도 default**

- STW의 시간과 빈도를 예측하게 만들고, 이를 설정할 수 있도록 하는 것
- 힙영역을 일정한 크기의 작은 공간으로 나눈다.
- region마다 우선순위가 존재한다. - region의 liveness(살아있는 객체)가 가장 적은 region을 선택하여 collect한다.

5. ZGC

- G1이 reign을 나누는 공간을 일정하게 나누지 않고 객체 크기의 대중소에 맞게 reign 크기를 조절해서 넣는다.
- G1GC와는 다르게 메모리를 재배치하는 과정에 **STW**
  없이 재배치
- Pointer를 이용해서 객체를 Marking하고 관리하는 것

https://docs.oracle.com/javase/9/gctuning/garbage-first-garbage-collector.htm#JSGCT-GUID-98E80C82-24D8-41D4-BC39-B2583F04F1FF

https://dhsim86.github.io/java/2018/02/05/gc_algorithms-post.html

Garbage-First Garbage Collector (oracle.com)

https://huisam.tistory.com/entry/jvmgc

📌 JVM 구조

다시 공부해야함.

1. class loader
2. runtime data area
3. 실행엔진
4. 네이티브 메소드

📌 String, StringBuilder, StringBuffer

1. String : Thread- safe, string 값 수정시 새로운 메모리 할당함(불변). 이전 값은 GC로 제거

   생성방식

   - new 연산자 이용: heap영역에 존재
   - 리터럴을 이용하는 방식 : string constant pool이라는 영역에 존재(heap)
2. StringBuilder : 가변, Thread-safe하지 않다. 단일쓰레드일때 사용. 속도가 빠르다
3. StringBuffer : 가변, Thread-safe하다.