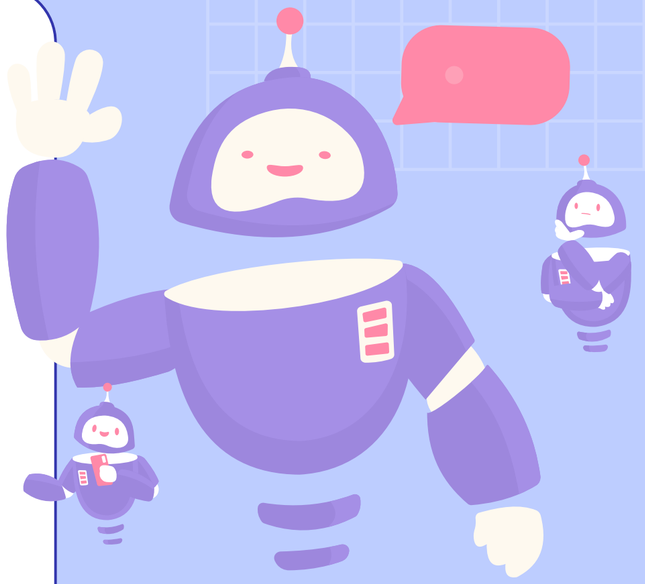
자바 가비지 컬렉션

동작 원리와 GC 종류



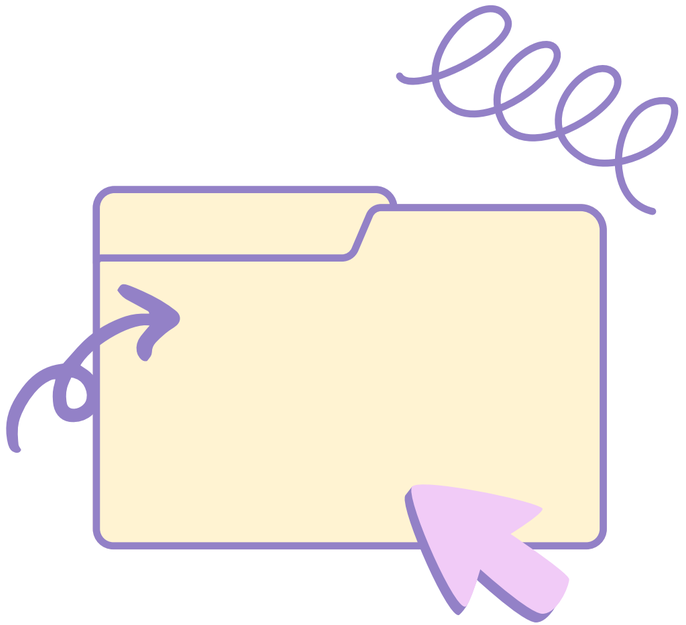
목차

01 개요

가비지 컬렉션의 정의와 개념

가비지 컬렉션이란?

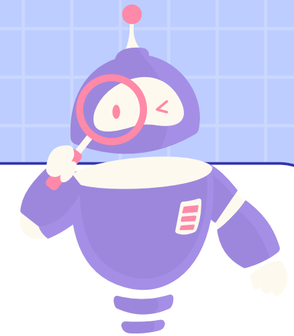
- 프로그램이 동적으로 할당했던 메모리 영역 중에서 더 이상 사용하지 않는 영역을 자동으로 탐지하고 해제하는 메모리 관리 기법
- 개발자가 명시적으로 메모리를 해제하지 않아도 시스템이 자동으로 처리



핵심 개념

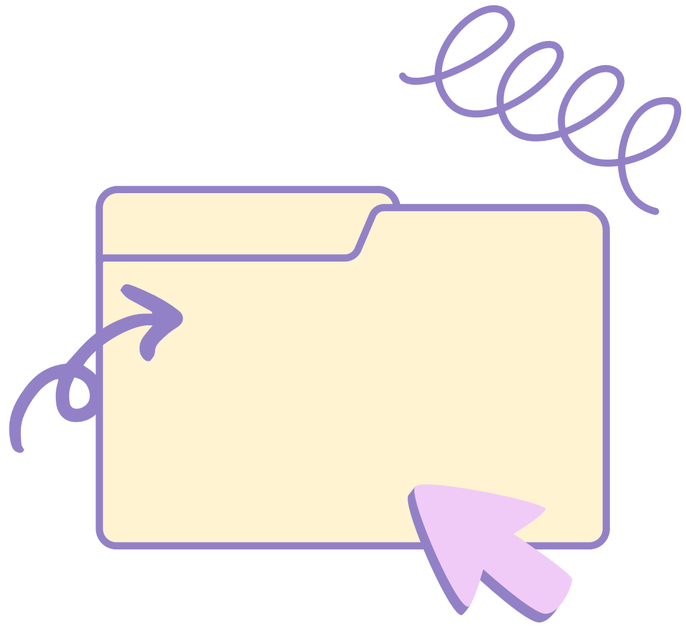
- 가비지(Garbage): 더 이상 참조되지 않는 객체 또는 메모리 영역
- 자동 메모리 관리(Automatic Memory Management): 개발자 개입 없이 시스템이 메모리 관리
- 런타임 프로세스: 프로그램 실행 중에 동작

01 개요



가비지 컬렉션의 목적과 중요성

- 메모리 누수(Memory Leak) 방지
  - 사용하지 않는 메모리가 계속 점유되는 문제 해결
  - 시스템 자원의 효율적 사용 보장
- 개발자 생산성 향상
  - 메모리 관리에 대한 부담 감소
  - 로직 개발에 집중 가능
- 메모리 관련 버그 감소
  - Dangling Pointer(허상 포인터) 문제 방지
  - Double Free(이중 해제) 문제 방지
  - 메모리 파편화 자동 관리
- 안정성 제공
  - 예측 가능한 메모리 사용 패턴
  - 시스템 크래시 가능성 감소

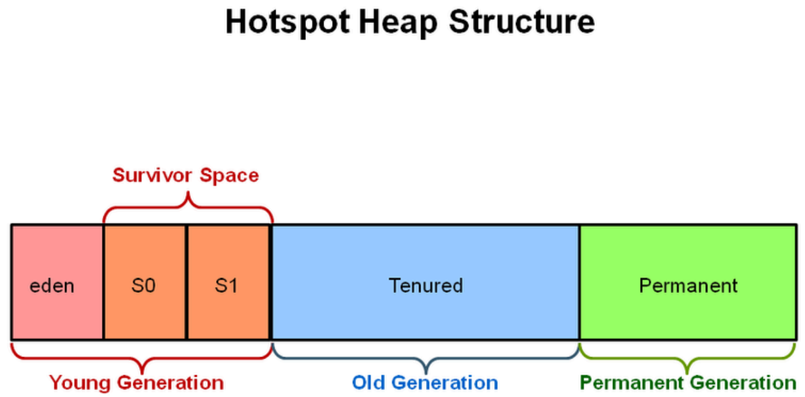


02 JVM 메모리 구조

JVM 메모리의 주요 구성 요소

힙 메모리 구조 (Heap Memory Structure)

- Young Generation (새로운 객체)
  - Eden Space: 새로 생성된 모든 객체가 처음 할당되는 공간
  - Survivor Spaces (S0, S1): Minor GC 후 살아남은 객체가 이동하는 두 개의 영역, 한 번에 하나만 사용됨
  - Young Generation은 전체 힙의 1/3 정도 차지 (조정 가능)
- Old Generation (오래된 객체)
  - 여러 GC 주기를 살아남은 객체들이 승격되어 저장되는 공간
  - 주로 장기 수명 객체 저장
  - Young Generation보다 크게 설정됨
  - Major GC의 대상



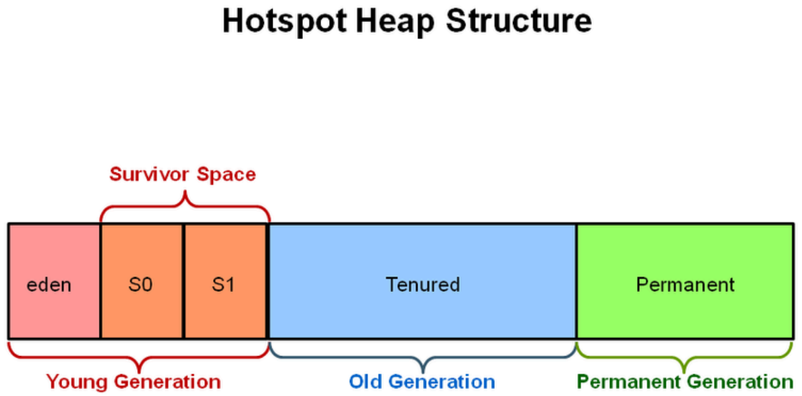
02 JVM 메모리 구조

객체가 Old Generation으로 넘어가는 조건

1. 나이(Age) 임계값 도달
- Minor GC에서 살아남을 때마다 객체의 나이가 증가
- 기본적으로 객체가 특정 나이(보통 15 or 31)에 도달하면 Old Generation으로 승격

2. 동적 나이 임계값
- JVM은 나이별 객체 분포를 모니터링
- 특정 나이의 모든 객체 크기 합이 Survivor 공간의 50% 이상을 차지 하면, 해당 나이 외 그보다 오래된 객체들을 Old Generation으로 즉시 승격

3. Survivor 공간 부족
- Survivor 영역이 가득 차면, 살아남은 객체들이 나이와 상관없이 Old Generation으로 즉시 승격(조기 승격)



03 가비지 컬렉션 알고리즘

Mark-Sweep 알고리즘

가장 기본적인 GC 알고리즘이며 두 단계로 구성: Mark(표시) + Sweep(제거)

- Mark 단계: GC Roots에서 시작해 모든 접근 가능한 객체 식별
- Sweep 단계: 표시되지 않은 객체 해제

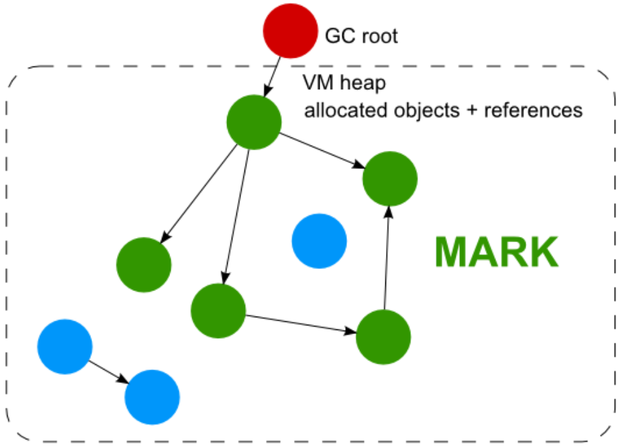
간단하게 말하면 회수할 객체들을 Mark(표시)한 다음 표시된 객체를 쓸어 담는 (Sweep) 식

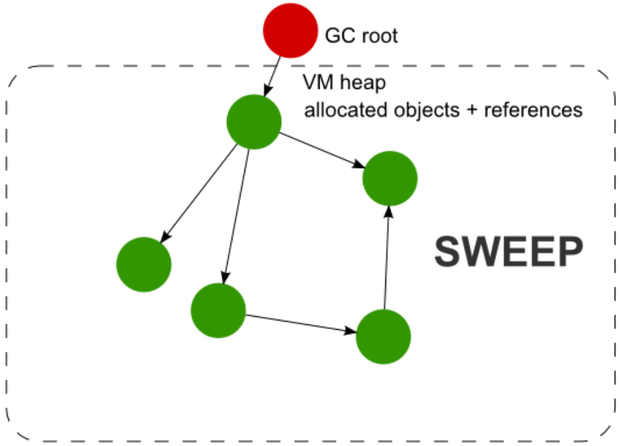
장점

- 구조가 단순하고 이해하기 쉬움, 객체 이동이 없음

단점

- 해제된 공간이 흩어져 있어 새 객체 할당이 어려울 수 있음(단편화 문제 발생)
- 단편화로 인해 전체 공간이 충분해도 할당 실패 가능





03 가비지 컬렉션 알고리즘

Mark-Sweep-Compact 알고리즘

Mark-Sweep 알고리즘에 Compaction(압축) 단계를 추가하여 메모리 단편화를 해결

- Mark 단계: 도달 가능한 객체를 표시
- Sweep 단계: 표시되지 않은 객체를 해제
- Compaction 단계: 남은 객체들을 메모리의 한쪽 끝으로 이동시켜 연속적인 공간 확보

장점 : 메모리 단편화 해결, 메모리 할당 효율 향상

단점 : 압축 과정으로 인한 추가 오버헤드, 더 긴 일시 중지 시간

03 가비지 컬렉션 알고리즘

Copying 알고리즘

단편화 문제를 해결하기 위해 제시된 방법이며 힙을 두 개의 영역으로 나누어 사용

두 개의 메모리를 A와 B라고 가정

1. 모든 메모리를 A에 할당
2. A가 가득 차거나 다른 이유로 GC가 발생할 경우 프로그램은 잠시 일시중지(Suspense)되고 A에서 살아남은 메모리를 모두 B로 복사
3. A는 Garbage 객체들만 존재하므로 A 메모리를 비움
4. B에 메모리를 할당하다가 다시 GC가 발생하면 A로 복사

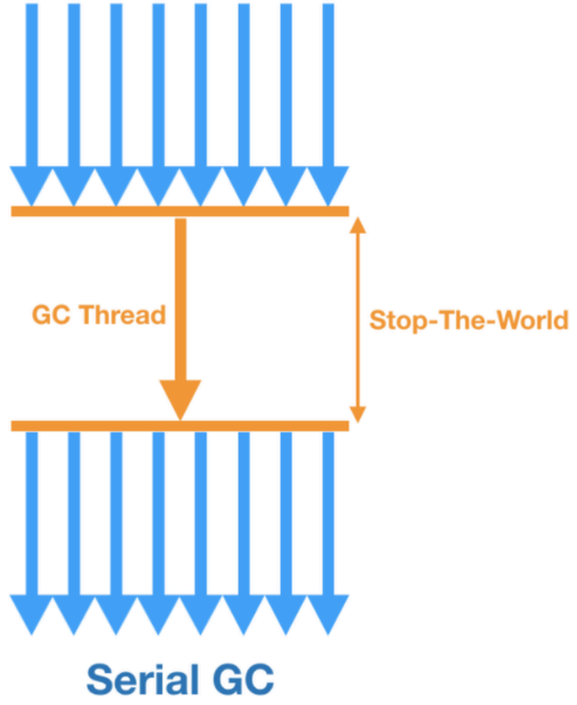
장점 : 단편화 없음, 빠른 할당, 효율적인 객체 배치

단점 : 메모리 사용량 두 배, 살아남는 객체가 많을 때 비효율적

04 JVM GC 종류

Serial GC

- 단일 스레드로 GC 수행 (Young, Old 모두 단일 스레드)
- 구현이 단순하지만 싱글 스레드로 동작하여 느리고, 그만큼 Stop The World 시간이 다른 GC에 비해 길다.
    - Stop The World : GC를 실행하기 위해 JVM이 모든 애플리케이션 실행을 멈추는 것
- Mark & Sweep & Compact 알고리즘을 사용

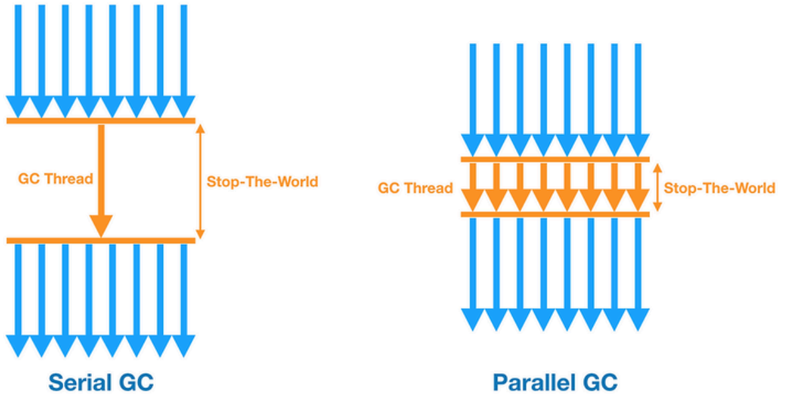


04 JVM GC 종류

Parallel GC

- Young 영역의 GC를 멀티 스레드 방식을 사용하기 때문에, Serial GC에 비해 상대적으로 Stop The World 가 짧다

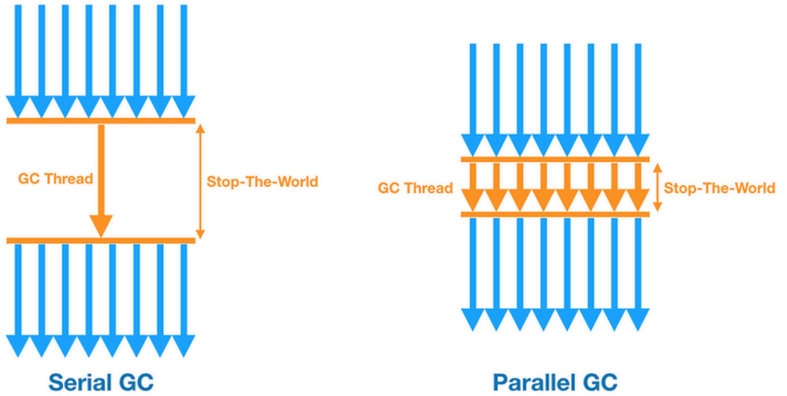
- 장점
  - CPU 활용 극대화, 빠른 전체 처리량
- 단점
  - Stop-The-World 발생 시 일시적 정지 시간 존재



04 JVM GC 종류

Parallel Old GC

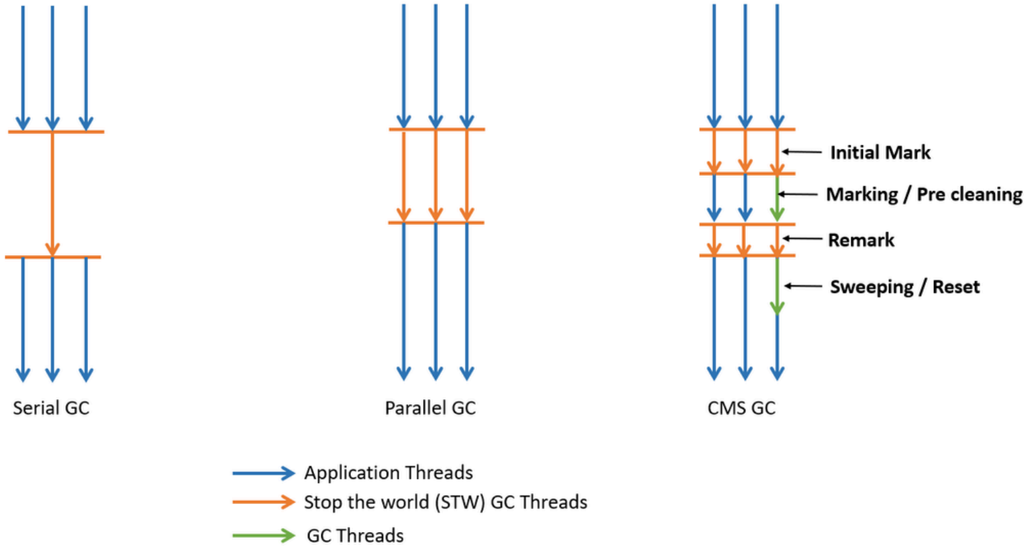
- Parallel GC는 Young 영역에 대해서만 멀티 스레드 방식을 사용했지만
- Parallel Old GC는 Old 영역까지 멀티스레드 방식을 사용
- 새로운 가비지 컬렉션 청소 방식인 Mark-Summary-Compact 방식을 이용
- 장점
  - CPU 활용 극대화, 빠른 전체 처리량
- 단점
  - Stop-The-World 발생 시 일시적 정지 시간 존재



04 JVM GC 종류

CMS GC

- 어플리케이션의 쓰레드와 GC 쓰레드가 동시에 실행되어 stop-the-world 시간을 최대한 줄이기 위해 고안된 GC지만 과정이 매우 복잡해짐.
- GC 대상을 파악하는 과정이 복잡한 여러단계로 수행되기 때문에 다른 GC 대비 CPU 사용량이 높음
- 메모리 파편화 문제 발생
- CMS GC는 Java9 버전부터 deprecated -> Java14에서는 사용 중지



04 JVM GC 종류

G1 GC

- CMS GC를 대체하기 위해 jdk 7 버전에서 최초로 release된 GC
- Java 9+ 버전의 디폴트 GC
- 현재 GC 중 stop-the-world의 시간이 제일 짧음

- 기존에는 Heap 영역을 물리적으로 고정된 Young / Old 영역으로 나누어 사용
- G1 GC는 개념을 뒤엎는 Region이라는 개념을 새로 도입하여 사용
- 전체 Heap 영역을 Region이라는 영역으로 분할하여 상황에 따라 Eden, Survivor, Old 등 역할을 고정이 아닌 동적으로 부여