5.14 Tech-Seminar : JAVA...에 대하여 1

728x90

짧은 시간 안에 Java의 모든 내용을 다룰 수는 없으므로

다음에 또 설명할 기회가 있길 바라며 간단히 목차를 작성해보았습니당

목차

더보기

... 유동적으로 진행할 예정.

---

1. Java

Java는 객체지향 언어의 한 종류이다.

1-1. 특징

JVM만 설치했다면 운영체제에 독립적이고,

객체 지향 프로그래밍의 특징인 캡슐화, 상속, 다형성, 추상화를 지원한다.

보안성이 뛰어나다.

GC가 자동으로 메모리를 관리해준다.

멀티 쓰레드를 지원한다.

더보기

자바는 한번에 여러개의 스레드가 동시에 수행된다. 즉, 멀티스레드 기반이다.

따라서 하나의 CPU가 여러개의 프로그램을 동시에 수행하도록 한다.

분산 환경에 적합하다.

더보기

분산 환경이란?

넓은 의미로는 웹서버와 데이터베이스 서버를 물리적으로 떨어트려 놓는 것,

또는 같은 역할을 여러 기계에서 수행하도록 하는 것,

좁은 의미로는 프로그래밍 관점에서 어떤 데이터를 조작하거나 함수를 수행하는 일을

다중 프로세서나 컴퓨터에 분산하여 처리할 수 있는 것.

자바는 분산 환경에서 TCP, IP등의 프로토콜을 통해 효율적으로 실행할 수 있도록 설계된 언어이다.

1-2. 장단점

장점

JVM위에서 동작하므로 운영체제 독립적이다.

GC가 메모리를 관리해주기 때문에 편리하다.

단점

JVM위에서 동작하기 때문에 실행 속도가 상대적으로 느리다.

다중 상속을 지원하지 않는다.

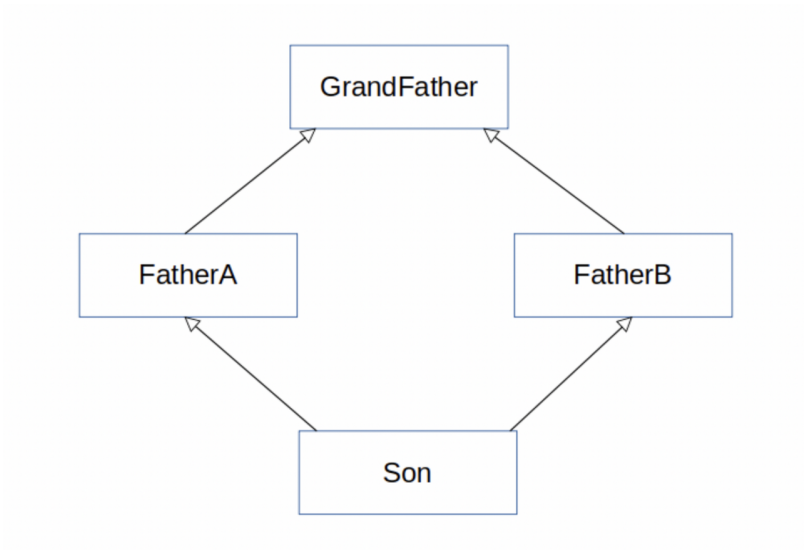
더보기

Java에서 다중 상속을 지원하지 않는 이유

다중 상속을 지원한다면 다이아몬드 문제가 발생할 수 있기 때문이다.

다이아몬드 문제란, 하나의 클래스가 여러개의 상위 클래스를 상속받을 수 있게 되어

일어나는 문제 상황을 의미한다.



<다이아몬드 문제>

두 부모 클래스가 모두 같은 이름의 메소드를 가지고 있다면,

이를 상속받은 Son 클래스는 어떤 부모 클래스의 메소드를 사용해야 할지 모르게 된다.

이러한 충동 상황을 방지해야 하므로 자바에서는 다중 상속을 지원하지 않는다.

같은 객체 지향 언어인 C++과 같은 경우,

이러한 문제 상황 해결의 책임을 프로그래머에게 주고 있다.

Java에서는 내부적으로 구현 불가하게 막아놓은 것이다.

헷갈리면 안되는 부분!

Java는 "클래스의" 다중상속을 지원하지 않는 것이다. 인터페이스는 다중 상속이 가능하다.
인터페이스는 기능에 대한 선언을 하는 역할이기 때문에, 다이아몬드 상속이 되더라도 상위 인터페이스에서 직접 구현된 것이 없기 때문에 충돌하지 않는다.

타입에 엄격하다.

1-3. 객체 지향과 절차 지향과의 차이

더보기

절차 지향 프로그래밍이란?

'순차적인 처리'가 중요시되며 프로그램 전체가 유기적으로 연결되도록 하는 프로그래밍 기법.

컴퓨터의 작업 처리 구조와 유사하다.

따라서 객체지향 언어를 사용하는 것보다 더 빠른 시간안에 처리된다.

그러나 실행 순서가 정해져 있어 코드의 순서가 바뀌면 동일한 결과를 보장할 수 없다.

또한 디버깅, 유지보수가 어렵다.

객체 지향 프로그래밍이란?

테이터와 절차를 하나의 덩어리로 묶어서 생각하는 것.

기능별로 묶어 모듈화를 하여 하드웨어는 같은 기능을 중복 연산하지 않고,

모듈을 재사용하기 때문에 하드웨어 처리양이 줄어들었다.

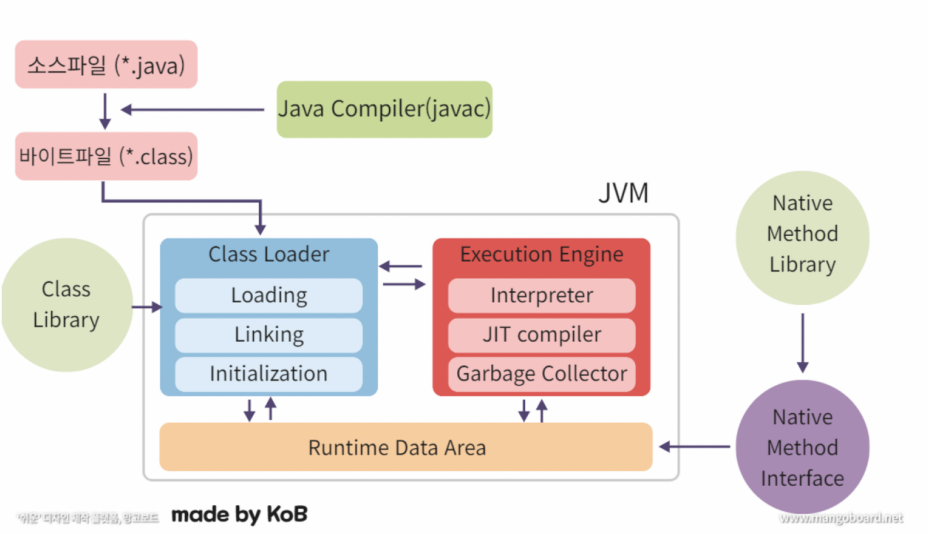
메소드를 통해서만 접근이 가능해 특정 함수에 직접적으로 접근할 수 없고,

식으로만 접근 가능하기 때문에 속도면에서 불이익이 있을 수 있다.

---

2. Java의 동작 과정

더보기



자바 동작 과정

동작 과정

1. 자바 소스파일을 컴파일러(javac)로 컴파일하여 .class 파일(자바 **바이트** 코드)를 생성한다.

C++같은 애들은 컴파일러가 컴파일하면 바이너리 코드의 형태를 갖는다.

이런 애들은 CPU가 이해할 수 있는 언어, JAVA 소스 파일이 컴파일 된 바이트 코드는

가상머신(JVM)이 이해할 수 있는 언어가 된다.

2. 클래스로더(JVM ClassLoader)가 컴파일된 자바 바이트 코드를 런타임 데이터 영역으로 로드한다.

클래스 로더는 동적 로딩을 통해 .class파일들을 로딩/링크하여 런타임 데이터 영역에 올린다.

3. 실행 엔진(JVM Execution Engine)이 자바 바이트 코드를 실행한다.

실행 엔진은 JVM에 올라온 바이트 코드들(.class 파일들)을 명령어 단위로 하나씩 가져와서 실행한다.

실행 엔진은 **인터프리터**와 **JIT 컴파일러**에 의해 동작한다.

> **인터프리터**란?
> 컴파일러와 같은 역할. 고레벨 언어를 기계어(저레벨 언어)로 해석해주는 프로그램이라고 생각하면 된다.
> 그러나 인터프리터는 소스코드의 각 행을 연속적으로 분석해 실행한다.
>
> 인터프리터는 각 행을 읽기때문에 시간적으로 느리다. 따라서 이를 개선하기 위해 JIT컴파일러가 도입되었다.

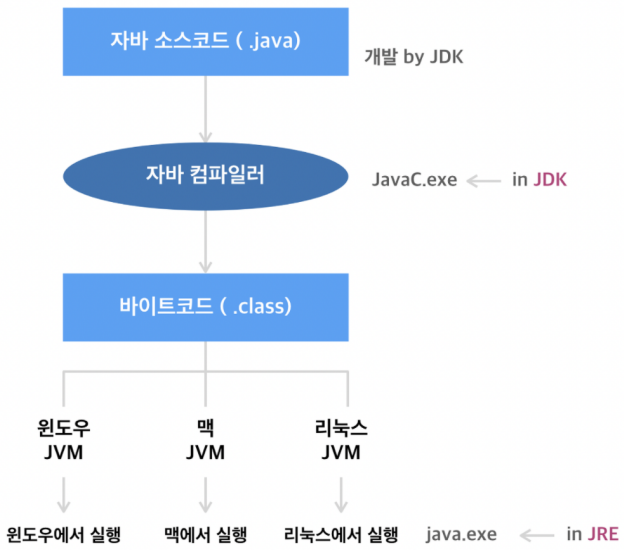
JIT Complier (Just In Time Compiler)
인터프리터와 다르게 각 행이 아닌 바이트 코드 전체를 읽어 한꺼번에 변환을 한다.

보통, 소스코드(.java)는 컴파일이 되어 중간코드인 .class파일이 되고, .class파일은 인터프리터에 의해 실행된다고 한다. 따라서 자바 언어는 컴파일언어+인터프리터 언어라고 한다.

(JS/Python은 인터프리터 언어, C/C++는 컴파일언어, JAVA는 컴파일러/인터프리터 언어)

2-1. JVM의 구조

더보기



JVM의 역할

자바 가상 머신 JVM은, Java와 OS사이에서 중간자 역할을 수행하는 친구이다.

위에서 신나게 설명했던 클래스파일(자바 바이트 코드)들을

각 OS에 맞게 해석해주는 역할을 한다고 생각하면 된다.

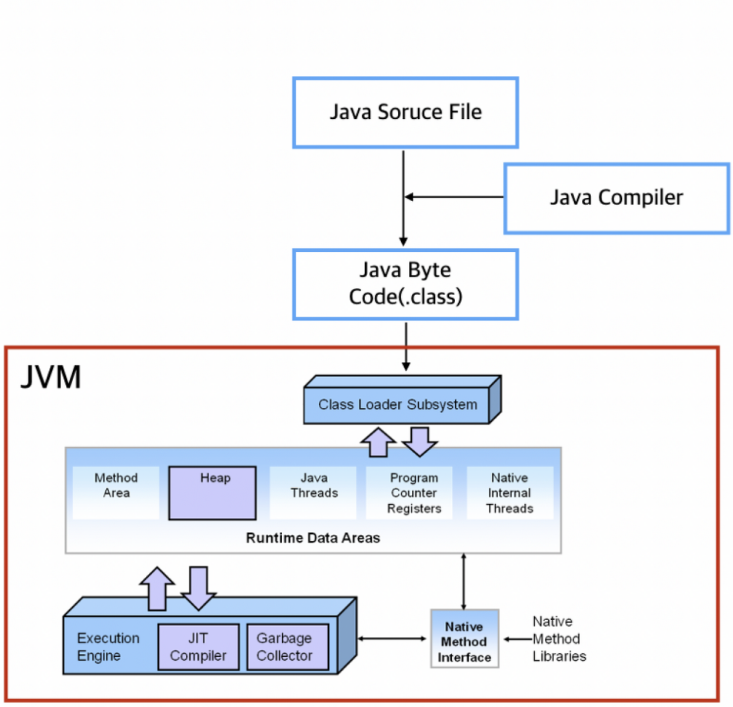
위에도 말했지만, 자바 컴파일러에 의해 생성되는 클래스 파일(자바 바이트 코드)는 기계어가 아니다.

따라서 C/C+ 처름 소스를 컴파일했다고 바로 기계어로 컴파일되어 OS에서 해석될 수 있는게 아니다.

C와 같은 애들은 OS가 바뀔 때마다 그에 맞는 실행 파일을 만들어줘야 한다.

그러나 Java는 JVM이 해석할 수 있는 바이트 코드로의 변환 과정을 추가했기 때문에

OS에 종속되지 않고 자바 파일 하나만 만들면 어느 디바이스든 JVM위에서 실행할 수 있는 것이다.



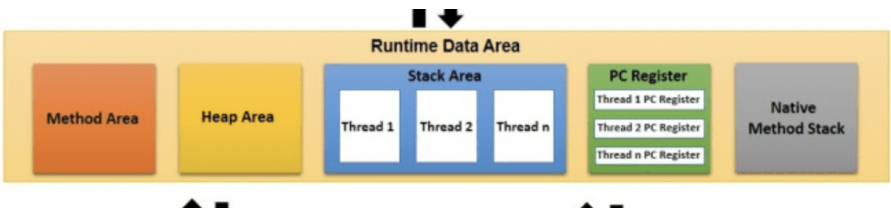
JVM구조

JVM의 구성 요소

1. Class Loader

동적 로딩을 통해 .class파일들을 로딩/링크하여 런타임 데이터 영역에 올린다.

2. Runtime Data Area



JVM의 메모리 영역이다. Method영역과 Heap 영역은 모든 스레드에서 공유된다.

나머지 영역은 개별 스레드가 할당된다.

Method 영역

클래스 변수명, 타입, 접근 제어자 같은 클래스 정보 및 인터페이스, static변수 등이 저장.

Heap 영역

new를 통해 생성된 객체와 배열 인스턴스 저장. GC의 대상이 되는 영역이다.

Stack 영역

메소드가 실행되면 스택 영역에 메소드에 대한 영역이 생긴다. 지역변수, 매개변수, 리턴값 등이 저장된다.

PC register 영역

현재 스레드가 실행되는 부분의 주소, 명령을 저장한다.

Native Method Stack 영역

자바 언어 외 언어로 작성된 코드를 위한 메모리 영역

3. Execution Engine

클래스 로더에 의해 메모리에 올라온 바이트 코드를 실행한다.

4. Garbage Collectors

더이상 참조하지 않는 힙 영역 메모리를 정리한다.

2-2. 가비지 컬렉터의 처리 과정

(N사 면접에서 다루었다고 함)

더보기

가비지 컬렉터 !

더이상 참조되지 않는 메모리를 청소해준다.

JVM이 메모리를 부여받고 프로그램을 실행하다가, 메모리가 부족해지는 순간이 오면

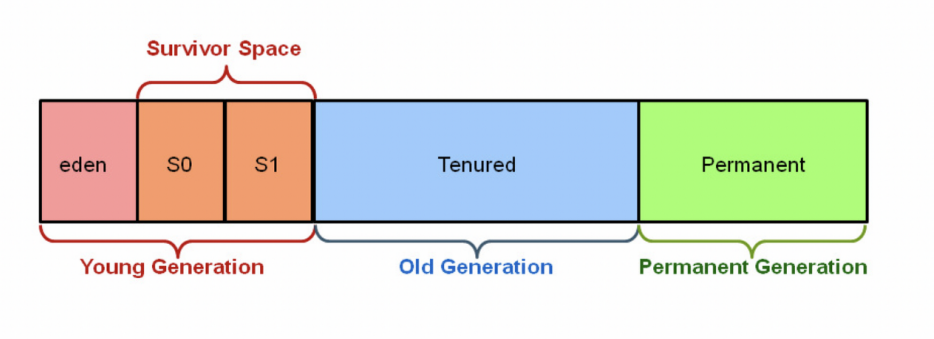
메모리를 더 요청한다고 한다. 이 때 가비지 컬렉터가 자동적으로 실행된다.

메모리를 정리하는 과정이기 때문에, 메모리 사용을 중단한 채로 진행해야 된다.

이를 'Stop-the-World'라고 한다.

Stop-the-World

GC 실행이 시작되면 GC 실행 스레드 이외 스레드는 모든 나머지 작업을 멈춘다고 한다.

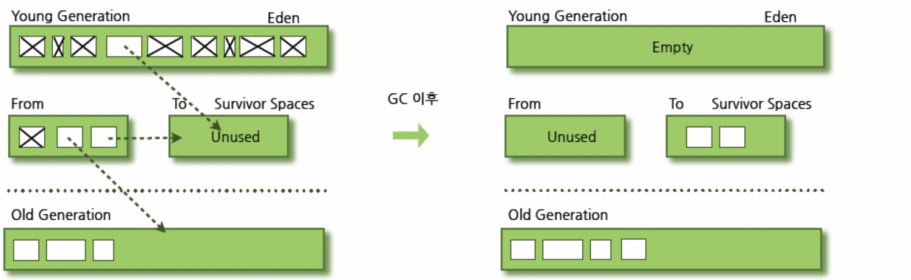


JVM heap 영역

GC의 작업은 위 Young에 대한 부분, Old에대한 부분으로 나뉜다.

Young 영역에서는 새롭게 생성된 객체들이 저장되는 영역,

Old 부분에서는 Young에서 계속 사용되던 객체들이 복사되는 영역이다.



- GC의 동작 방법 -

GC 동작 방법

eden에 객체가 생기면 **eden에 GC가 동작**한다.

여기서 살아남은 애들이 S0으로 이동.

S0에 GC가 동작한다.

여기서 살아남은 애들이 S1로 이동.

이를 반복해 특정 횟수만큼 살아남으면 Old로 이동.

Old영역이 가득차면 **Old영역에 GC가 동작.**

Mark and Sweep!

GC가 닿을 수 있는 모든 변수, 객체를 스캔하며 어떤 객체를 가리키고 있는지 Mark하고, (이 때 StoptheWorld)

Mark되지 않은 객체를 Heap에서 제거하는 Sweep 과정을 거친다.

[가비지 컬렉션(Garbage Collection) 알고리즘의 종류]

Serial GC

mark-sweep-compact 알고리즘을 사용

Old영역에서 살아있는 객체 확인(Mark),

살아있는 객체만을 남긴다.(Sweep)

그리고 난 후 객체들을 앞부분부터 채워, 객체가 존재하는 부분과 존재하지 않는 부분으로 나눈다.(Compaction)

Parallel GC

Serial GC와 같지만 멀티 스레드 이용

Parallel Old GC(Parallel Compacting GC)

Serial GC의 Sweep 대신 Summary를 사용

Summary : 앞서 GC를 수행한 영역에 대해서 별도로 살아있는 객체를 Mark, Sweep보다 조금 더 복잡함.

Concurrent Mark & Sweep GC(이하 CMS)

Initial Mark 단계에서는 살아 있는 객체를 찾는 것으로 끝낸다.(Stop-the-World 시간이 짧음)

후에 찾은 객체에서 참조하는 객체를 Concurrent하게(여러 스레드가 동시에) 따라가는 Concurrent Mark 수행.

그 이후에 Stop-the-World가 실행되고 Concurrent하게 Remark.

애플리케이션의 응답속도가 매우 중요할 때 사용한다.

G1(Garbage First) GC

바둑판의 각 영역에 객체를 할당하고 GC를 실행.

위에서 설명한 Young영역과 Old영역에 대한 개념을 사용하지 않고, 객체를 할당한다.

끝!

참고했어용~

더보기

https://mangkyu.tistory.com/94

'Study && Education > Tech-Seminar' 카테고리의 다른 글

5.14 Tech-Seminar : JAVA...에 대하여 1 **(0)** 2022.05.14