MEGAZONE
Cloud Innovator

JVM VS GraalVM

2022.02
DxLabs 가영권

INDEX

1

JVM이란 무엇인가?

JVM이란?

JVM이란, Java Virtual Machine의 줄임말이며 Java Byte Code를 OS에 맞게 해석 해주는 역할을 한다.

(Java Byte Code를 실행할 수 있는 주체이다.)

일반적으로 인터프리터나 JIT 컴파일 방식으로 다른 컴퓨터 위에서 Byte Code를 실행할 수 있도록 구현되나 jop 자바 프로세서처럼 하드웨어와 소프트웨어를 혼합해 구현하는 경우도 있다.

Java Byte Code는 플랫폼(EX) 운영체제)에 독립적이며 모든 자바 가상 머신은 자바 가상 머신 규격에 정의된 대로 Java Byte Code를 실행한다. (자바가 운영체제에 구애 받지 않고 프로그램을 실행할 수 있도록 도와준다.)

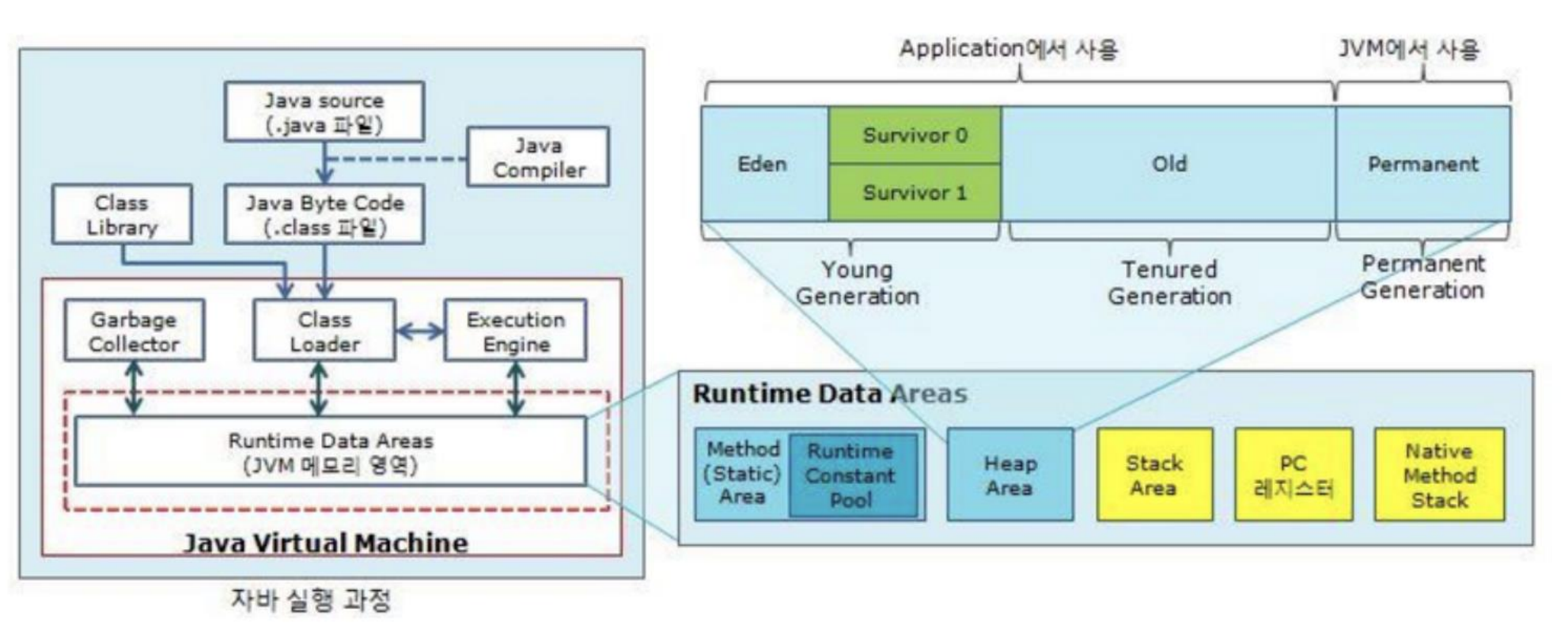
따라서 표준 자바 API까지 동일한 동작을 하도록 구현한 상태에서는 이론적으로 모든 자바 프로그램은 CPU나 운영 체제의 종류와 무관하게 동일하게 동작할 것을 보장한다.

또한, 가비지 컬렉터를 사용한 메모리 관리도 자동으로 수행하며, 다른 하드웨어와 다르게 레지스터 기반이 아닌 스택 기반으로 동작한다.

1

JVM이란 무엇인가?

JVM Architecture



1

JVM이란 무엇인가?

자바 프로그램의 실행 단계

자바는 운영체제 독립적으로 JVM 환경에서 동작할 수 있도록 설계가 되어 있는 동적 언어이다.

프로그램이 실행되면 JVM은 OS로부터 해당 프로그램이 필요로 하는 메모리를 할당 받는다.

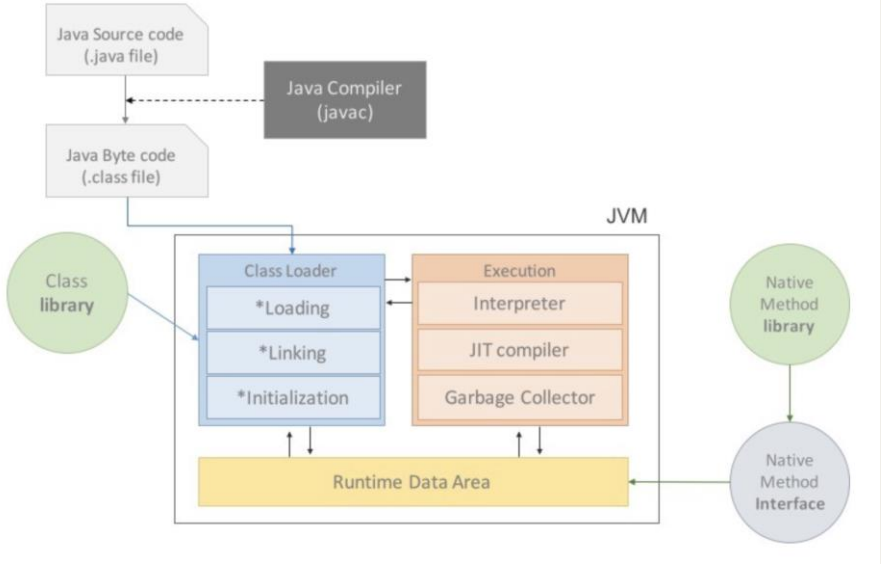
Java compiler는 .java 파일을 .class 라는 Java Byte Code로 변환시켜준다.

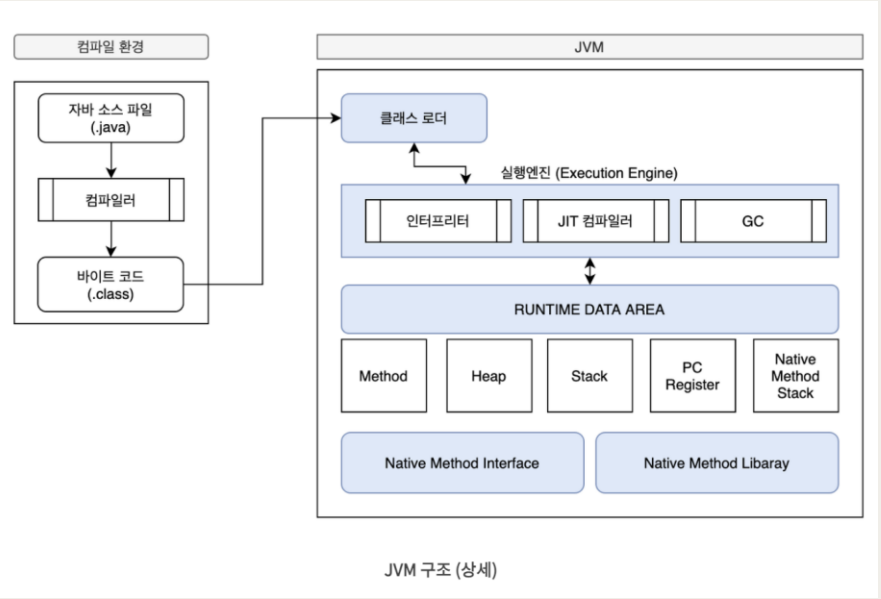
Byte Code는 기계어가 아니기 때문에 운영체제(OS)에서 바로 실행되지 않는다.

Class Loader는 변환된 Byte code(.class) 파일을 JVM 내로 class를 로드하고, OS가 Byte Code를 이해할 수 있도록 자바 인터프리터, JIT 컴파일 방식을 통해 Byte Code를 기계어로 해석한다.(Execution Engine을 통해 해석)

이러한 과정을 거쳐 Java가 운영체제에 구애 받지 않고 프로그램을 실행할 수 있도록 한다.

JVM은 각 운영 체제에 맞도록 되어있기 때문에 운영 체제에 종속적이다.





JVM 구조 (상세)

2

JVM은 어떻게 동작하는가?

JVM 메모리 구조

JVM의 구조는 크게 보면,

Garbage Collector, Execution Engine, Class Loader, Runtime Data Area로, 4가지로 나눌 수 있다.

2

JVM은 어떻게 동작하는가?

JVM 메모리 구조

(1) **Class Loader**

변환된 Byte code(.class) 파일을 JVM 내로 class를 로드하고 Link작업을 통해 배치 등 일련의 작업을 한다. 또 런타임시 class를 load한다.

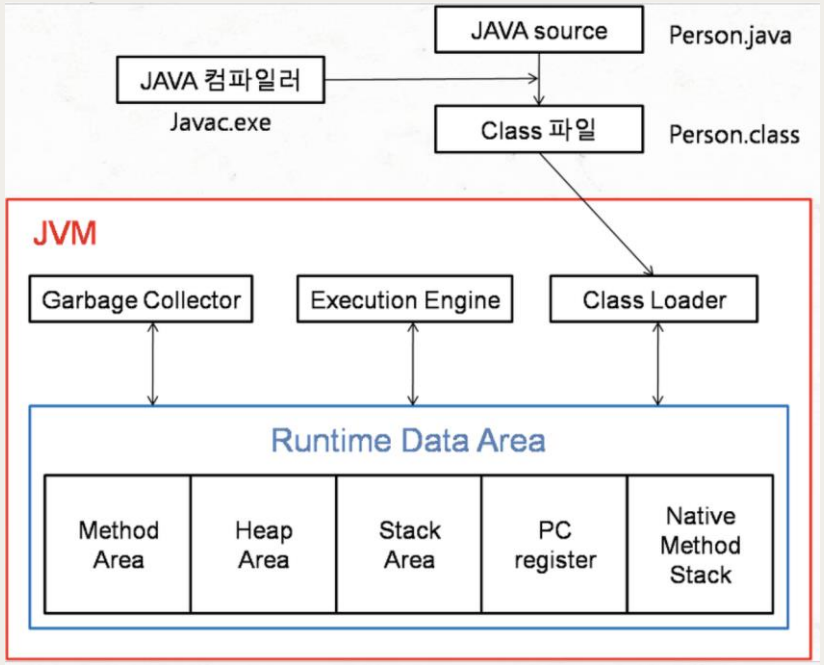
(2) **Execution Engine**

Class Loader를 통해 JVM 내부로 넘어와 Runtime Data Area(JVM 메모리)에 배치된 ByteCode들을 명령어 단위로 실행시킨다.

최초 JVM이 나왔을 당시에는 인터프리터 방식이었기 때문에 속도가 느리다는 단점이 있었지만 JIT 컴파일러 방식을 통해 이 점을 보완하였다.

JIT는 바이트 코드를 어셈블러 같은 네이티브 코드로 바꿈으로써 실행이 빠르지만 역시 변환하는데 비용이 발생하였다.

이 같은 이유로 JVM은 모든 코드를 JIT 컴파일러 방식으로 실행하지 않고, 인터프리터 방식을 사용하다가 일정한 기준이 넘어가면 JIT 컴파일러 방식으로 실행한다.



2

JVM은 어떻게 동작하는가?

JVM 메모리 구조

(3) **Garbage Collector**

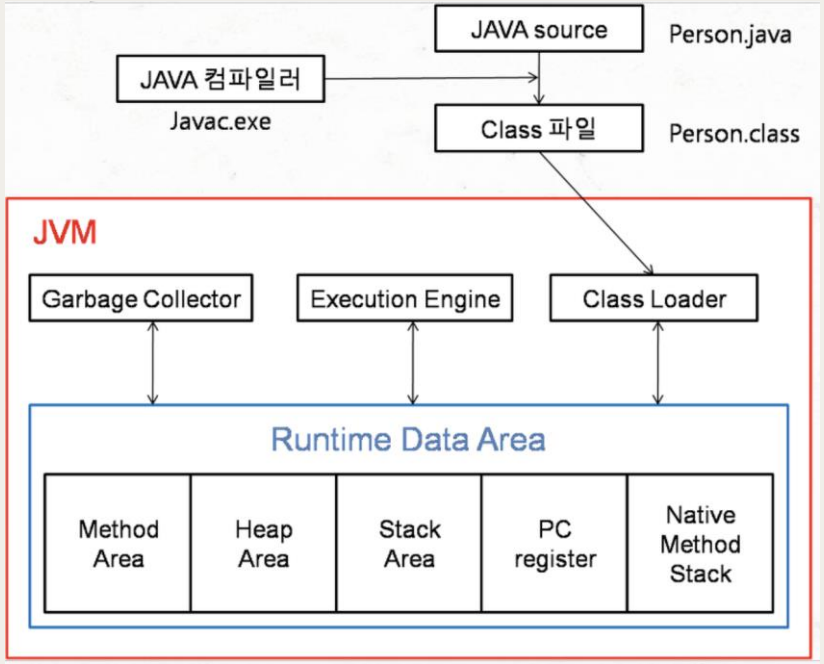
Garbage Collector(GC)는 어플리케이션이 생성한 객체의 생존 여부를 판단하여, 더이상 참조되지 않거나 NULL인 객체의 메모리를 해체시켜 메모리 반납을 한다.

즉, 힙 메모리 영역에 생성된 객체들 중에서 참조되지 않은 객체들을 탐색 후 제거하는 역할을 한다. 이때, GC가 역할을 하는 시간은 언제인지 정확히 알 수 없다.

(4) **Runtime Data Area**

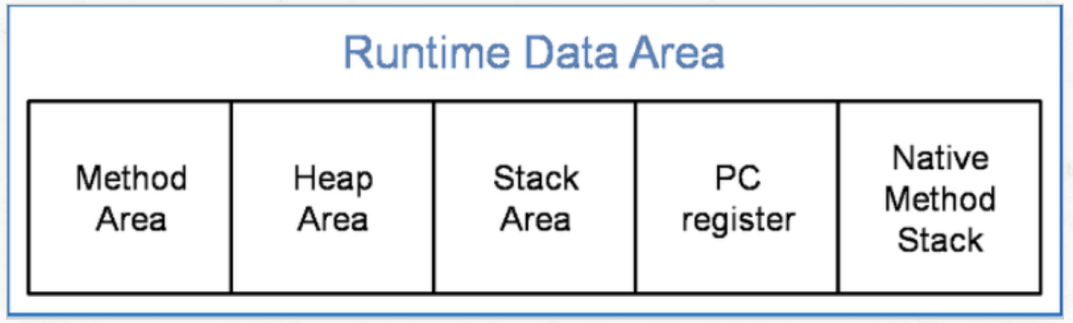
JVM의 메모리 영역으로 Java 애플리케이션을 실행할 때 사용되는 데이터들을 적재하는 영역(할당받은 메모리 영역)이다.

이 영역은 크게 Method Area, Heap Area, Stack Area, PC Register, Native Method Stack로 나눌 수 있다.



2

JVM은 어떻게 동작하는가?



JVM 메모리 구조

Runtime Data Area

1. Method area

모든 쓰레드가 공유하는 메모리 영역이다.

JVM이 읽어들인 클래스와 인터페이스 대한 런타임 상수 풀, 멤버 변수(필드), 클래스 변수(Static 변수), 생성자와 메소드를 저장하는 공간이다.

2. Heap area

JVM이 관리하는 프로그램 상에서 데이터를 저장하기 위해 런타임 시 동적으로 할당하여 사용하는 영역이다. New 연산자로 생성된 객체 또는 객체(인스턴스)와 배열을 저장한다.

또한, 메소드 영역에 로드된 클래스만 생성이 가능하고 Garbage Collector가 참조되지 않는 메모리를 확인하고 제거하는 영역이다.

3. Stack area

메소드를 호출할 때마다 프레임(Frame)을 추가(push)하고 메소드가 종료되면 해당 프레임을 제거(pop)하는 동작을 수행한다.

그리고 메서드 안에서 사용되는 값들을 저장하고, 호출된 메서드의 매개변수, 지역변수, 리턴 값 및 연산 시 일어나는 값들을 임시로 저장한다.

마지막으로, 메서드 수행이 끝나면 프레임별로 삭제한다.

2

JVM은 어떻게 동작하는가?

JVM 메모리 구조

Runtime Data Area

4. PC Register

쓰레드가 시작될 때 생성되며, 생성될 때마다 생성되는 공간으로 쓰레드마다 하나씩 존재한다.

현재 수행 중인 JVM 명령 주소를 갖는다.

프로그램 실행은 CPU에서 인스트럭션(Instruction)을 수행.

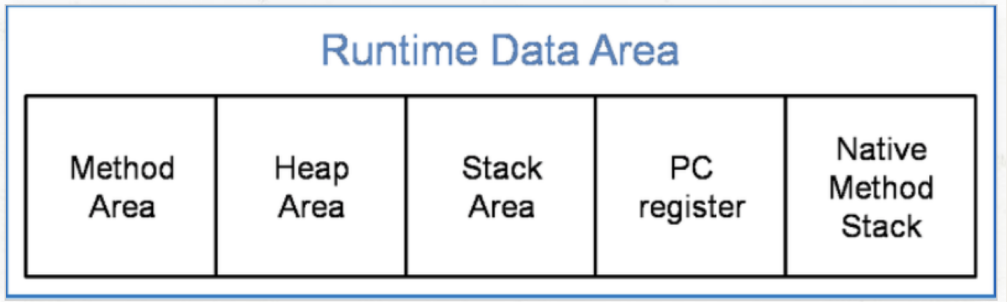
CPU는 인스트럭션을 수행하는 동안 필요한 정보를 CPU 내 기억장치인 레지스터에 저장한다.

연산 결과값을 메모리에 전달하기 전 저장하는 CPU 내의 기억장치

5. Native method stack

자바 외 언어로 작성된 네이티브 코드를 위한 Stack이다.
즉, JNI(Java Native Interface)를 통해 호출되는 C/C++ 등의 코드를 수행하기 위한 스택이다.

네이티브 메소드의 매개변수, 지역변수 등을 바이트 코드로 저장한다.

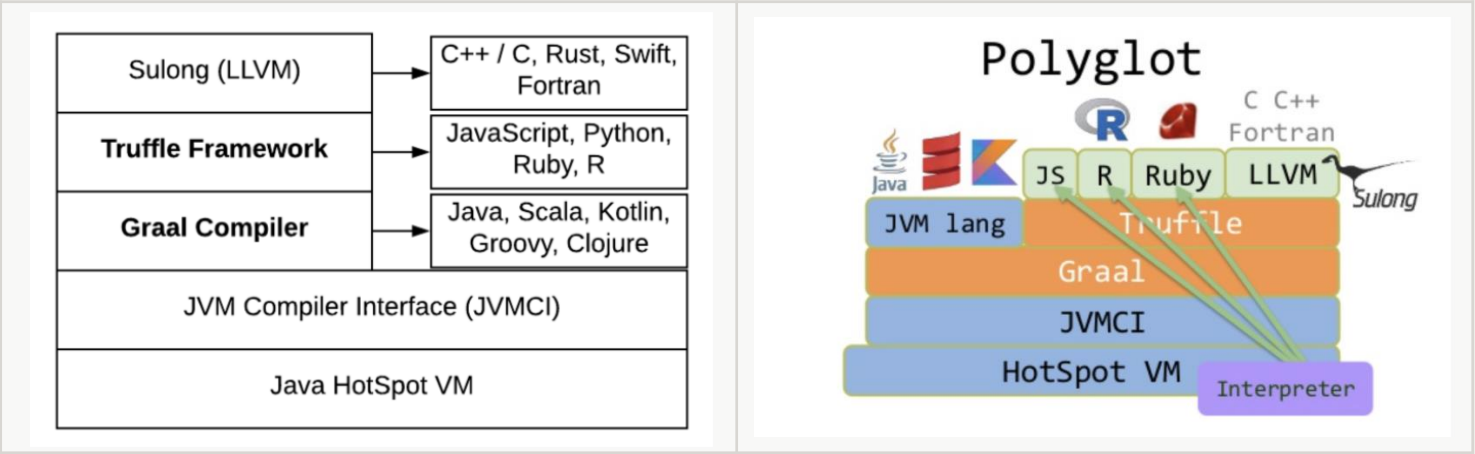


3

GraalVM이란 무엇인가?

GraalVM이란?

GraalVM이란, JavaScript, Python, Ruby, R, Java, Scala, Clojure, Kotlin과 같은 JVM 기반 언어와 C 및 C ++와 같은 LLVM 기반 언어로 작성된 애플리케이션의 실행을 가속화하기 위한 범용 가상 머신 (Java로 구현 된 HotSpot / OpenJDK를 기반으로 하는 Java VM 및 JDK)이다.



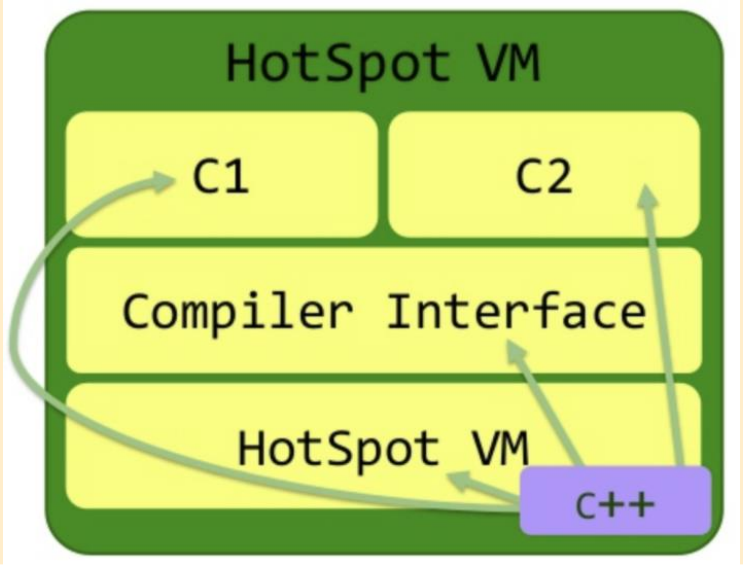
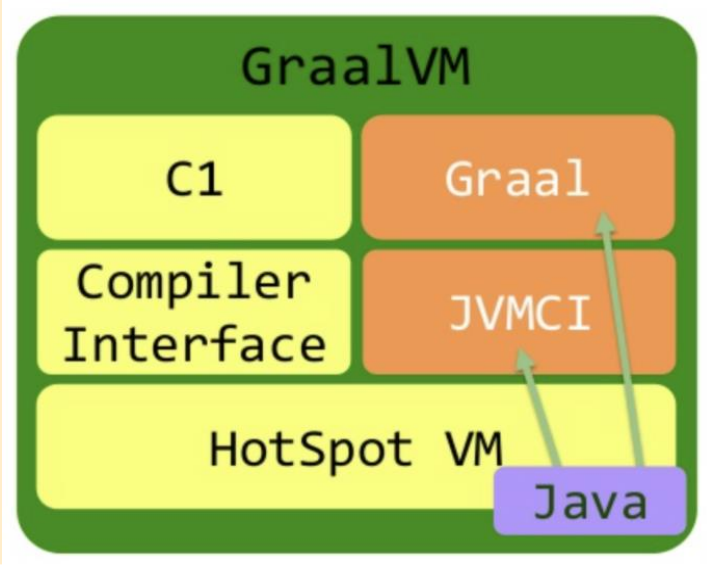
- **HotSpot VM** : Hot한 Spot을 찾아서 해당 부분에서는 JIT 컴파일러를 사용하는 방법
- **JVM Compiler Interface(JVMCI)** : Java로 작성된 컴파일러를 JVM에서 동적 컴파일러로 사용할 수 있도록 Java 기반 JVM 컴파일러. (HotspotVM 과 통신하는 컴파일러)
- **Graal Compiler** : JVMCI를 기반으로 하며 추가 최적화를 통해 더 나은 JIT 컴파일러 구현을 제공
- **Truffle Framework** : 언어 인터프리터 프레임 워크, GraalVM의 다국어를 전담
- **Sulong** : Graal VM에 빌드 된 고성능 LLVM 비트 코드 런타임, Sulong은 Truffle과 Graal을 동적 컴파일러로 사용

3

GraalVM이란 무엇인가?

GraalVM 구조

- **GraalVM**은 C2 컴파일러와 HotspotVM 과 통신하는 컴파일러 인터페이스 부분을 java로 작성했다. 여기서 C2 컴파일러를 Graal이라고 말한다.
- **Graal** : GraalVM의 새로운 JIT(just-in-time) Compiler(Java로 작성)

기존 HotSpot VM	Graal VM
	

참고

2개의 JIT compiler를 포함

C1: client compiler
- **C1 디자인: 빠른 시작**
- **코드 최적화의 비중 낮춤**

C2: server compiler
- **C2디자인:약간느린시작**
- **코드 최적화의 비중 낮춤**

3

GraalVM이란 무엇인가?

GraalVM 주요 기능

- **High Performance**
  - 속도나 자원의 사용률에 대해서 성능을 향상
  - 고성능 최적화 Just-In-Time 컴파일러(Java로 작성)

- **Ahead-of-Time Compilation**
  - Natvie Binary에 관련된 내용이다.
  - Natvie 실행 파일을 빌드
  - AOT 컴파일의 경우 Native Image Builder 또는 native-image 기술을 사용하여 Java 코드를 독립 실행형 실행 파일로 미리 컴파일한다.
  - Java의 약점인 인터프리터 실행 환경을 극복하고자 하는 목적으로 미리 타깃 OS에 맞는 실행 파일을 만들겠다는 의미이다.
  - 컴파일의 결과물이 특정 OS에 맞는 실행 파일로 만들어 줄 수 있기 때문에 타깃 OS에 java를 설치할 필요도 없게 되고, 실행 시 JIT가 동작을 할 필요가 없기 때문에 빠르게 실행이 될 수 있는 장점이 있다.

- **Language Choice**
  - GraalVM은 모든 언어를 실행할 수 있게 해 줄 수 있다.(Polyglot 지원, Truffle Framework 사용을 통해..)

- **Advanced Tools**
  - Dashboard, Chrome Debugger, VisualVM,.. 등 지원하는 툴이 있다.

4

GraalVM은 어떻게 동작하는가?

GraalVM, 여러 작동 모드를 제공하는 런타임 환경

- **JVM 런타임 모드**
  - HotSpot JVM에서 프로그램을 실행할 때 GraalVM은 기본적으로 최상위 JIT 컴파일러로 GraalVM 컴파일러를 사용한다.
  - 런타임 시 애플리케이션이 로드되어 JVM에서 정상적으로 실행된다.
  - JVM은 Java 또는 기타 JVM 고유 언어에 대한 바이트 코드를 컴파일러에 전달하고 컴파일러는 이를 기계 코드로 컴파일하고 JVM으로 반환한다.

- **Native Image**
  - Native Image는 Java 코드를 독립 실행형 바이너리 실행 파일 또는 기본 공유 라이브러리로 컴파일하는 기술이다.
  - GraalVM이 제공하는 Native Image는 javac로 컴파일된 클래스 파일 혹은 JAR 파일을 실행 파일로 만드는 Native Image Generator이다.
  - Native Image 빌드 중에 처리되는 Java 바이트 코드에는 필요한 모든 애플리케이션 클래스, 종속성, 타사 종속 라이브러리 및 모든 JDK 클래스가 포함된다.
  - 생성된 자체 포함 기본 실행 파일은 JVM이 필요하지 않은 각 개별 운영 체제 및 시스템 아키텍처에 따라 다르다.

- **Java on Truffle**
  - Java on Truffle은 Truffle 언어 구현 프레임워크로 구축된 JVM(Java Virtual Machine) 사양의 구현이다.
  - 모든 핵심 구성 요소를 포함하고 Java Runtime Environment 라이브러리와 동일한 API를 구현하며 GraalVM의 모든 JAR 및 기본 라이브러리를 재사용하는 완전한 Java VM이다.

5 참고 사이트

JVM

https://www.holaxprogramming.com/2013/07/16/java-jvm-runtime-data-area/

https://steady-coding.tistory.com/305

https://hoonmaro.tistory.com/19

https://limkydev.tistory.com/51

https://medium.com/@lazysoul/jvm-%EC%9D%B4%EB%9E%80-c142b01571f2#.tel2hzjyz

https://swk3169.tistory.com/181

https://minikuma-laboratory.tistory.com/25

GraalVM

https://www.graalvm.org/22.0/docs/

https://www.graalvm.org/22.0/docs/introduction/

https://meetup.toast.com/posts/273

https://giljae.medium.com/graalvm-%EC%86%8C%EA%B0%9C-84ac547f8df2

It started!
Let's growing!
Cloud Native