더 자바, 코드를 조작하는 다양한 방법

JVM(Java Virtual Machine)

- 자바 가상 머신으로 자바 바이트 코드(.class)를 OS에 특화된 코드로 변환(인터프리터와 JIT 컴파일러)하여 실행한다.
- 바이트 코드를 실행하는 표준(JVM 자체는 표준)이자 구현체(특정 밴더가 구현한 JVM)이다.
- JVM 스펙: https://docs.oracle.com/javase/specs/jvms/se11/html/
- JVM 밴더: 오라클, 아모존, Azul,
- 특정 플랫폼에 종속적

```
Java11 부터는 Jre를 제공하지 않음
Java9부터 모듈 개념이 들어옴
```

JRE, JDK

오라클에서 만든 자바11 버전부터는 상용으로 사용할 때 유료 오라클에서 만든 open-jdk는 버전 업그레이드도 되고 무료

JVM에서 구동되는 언어로는

```
클로저, 그루비, JRuby, Jython, 코틀린, ...
```

JVM 구조

클로스 로더 시스템

- 로딩: 클래스의 메모리 로딩
- 링크: 클래스간 연결(레퍼런스 연결)
- 초기화: 각 클래스, 변수 초기화

메모리

- 메소드
  - 클래스명, 부모 클래스명(상속 클래스) 정보 저장
  - 공유 영역
- 힙
  - 객체저장, 인스턴스 저장

스택, PC(PC registers), 네이티브 메소드 스택

- 쓰레드 단위로 저장

실행 엔진(Execution Engine)

- 인터프리터: 바이트 코드를 한 줄씩 실행
- JIT(Just In Time) 컴파일러: 인터프리터 효율을 높이기 위해, 인터프리터가 반복되는 코드를 발견하면

JIT 컴파일러로 반복되는 코드를 모두 네이티브 코드로 바꿔둔다. 그 다음부터 인터프리터는 네이티브 코드로 컴파일된 코드를 바로 사용한다.

- GC(Garbage Collector): 더 이상 참조되지 않는 객체를 모아서 정리한다.

클래스 로더 시스템

로딩, 링크, 초기화 순으로 진행된다.

로딩

- 클래스 로더가 .class 파일을 읽고 그 내용에 따라 적절한 바이너리 데이터를 만들고 “ 메소드 ” 영역에 저장
- 메소드 영역에 저장되는 데이터
  - FQCN: Fully Qualified Class Name
  - 클래스 | 인터페이스 | ENUM
  - 메소드와 변수
- 로딩이 끝나면 해당 클래스 타입의 Class 객체를 생성하여 “ 힙 ” 영역에 저장

클래스 로더

- 부트 스트랩 클래스 로더: JAVA_HOME/lib에 있는 코어 자바 API를 제공하며 최산위 우선순위를 가진 클래스 로더
- 플랫폼 클래스 로더(Extension 클래스 로드): JAVA_HOME/lib/ext 폴더 또는 java.ext.dirs 시스템 변수에 해당하는 위치에 있는 클래스를 읽는다
- 애플리케이션 클래스 로더: 애플리케이션 클래스패스(-classpath, java.class.path, 환경 변수의 값에 해당하는 위치)에서 클래스를 읽는다.

 ClassFileNotFoundException은 클래스 로드가 못 찾는 것

링크

Verify, Prepare, Resolve(optional)

- 검증: .class 파일 형식이 유효한지 체크한다.
- Preparation: 클래스 변수(static 변수)와 기본값에 필요한 메모리 확보
- Resolve: 심볼리 메모리 레퍼런스를 메소드 영역에 있는 실제 레퍼런스로 교체한다.

초기화

Static 변수의 값을 할당한다. (static 블럭이 있다면 실행 된다)

바이트 코드 보기

```
$> javap -c -v ${클래스 파일 경로} // -v 애노테이션 정보도 볼 수 있음
Compiled from "CodeManipulationApplication.java"
public class com.repia.java.codemanipulation.CodeManipulationApplication {
  public com.repia.java.codemanipulation.CodeManipulationApplication();
    Code:
       0: aload_0
       1: invokespecial #8                  // Method
java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: ldc           #1                  // class
com/repia/java/codemanipulation/CodeManipulationApplication
       2: aload_0
       3: invokestatic  #16                 // Method
org/springframework/boot/SpringApplication.run:(Ljava/lang/Class;[Ljava/lang
/String;)Lorg/springframework/context/ConfigurableA
pplicationContext;
       6: pop
       7: return
}
```

리플렉션

스프링 Dependency Injection

어떻게 동작할까?

주레피, manipulation

From:
http://125.132.25.164/dokuwiki/ - **문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12**

Permanent link:
http://125.132.25.164/dokuwiki/doku.php?id=wiki:java:%EC%BD%94%EB%93%9C%EC%A1%B0%EC%9E%91&rev=1651142093

Last update: **2022/04/28 19:34**