소프트웨어 실습 2
1장 참고 자료

Lee Yang Min

Dalvik VS ART

1. Android Runtime(ART)
   - 왜 나왔을까?
   - 안드로이드에서 사용되던 Dalvik VM의 한계점을 극복하기 위해 새로 개발된 런타임이다.

2. JVM, Dalvik VM, ART 비교
   - 2.1 JVM
     - ByteCode → interpret → 각 플랫폼에 맞는 기계어로 번역 → 프로그램 실행 (Byte 코드가 machine 코드가 아님에 주의할 것)
     - 장점 : WORA(write once read anywhere), OS에 구애 받지 않고 해당 OS에 맞는 기계어로 번역됨
     - 단점 : Native 언어들에 비해 속도가 느리다.
   - 2.2 Dalvik VM
     - Dex file을 Dalvik machine 위에 올리는 방식
     - 라이선스 문제로 인해 JVM 대신 Java 코드를 이용할 수 있도록 개발됨
     - Bytecode → interpret → 실행 → 개선 → 네이티브 코드로 변환(Dalvik의 JIT, Just in TIME 컴파일 방식, 특정 임계크기를 넘으면 Byte 코드를 기계어로 미리 변경)
     - 장점 : 다양한 하드웨어나 아키텍처에서 실행 할 수 있는 장점(기존 JVM의 특징)
     - 단점 : 성능과 배터리에 악영향, JIT compile 된 코드가 올라가 메모리가 늘어남

3. ART 방식(Android 4.4 Kitkat 이후)
   - machine 위에서 OAT file을 돌리는 방식, VM이 아닌 런타임 시 사용되는 라이브러리
   - 앱을 설치할 때 완전히 네이티브 코드로 변환되어 설치됨(AOT 컴파일, Ahead-Of-Time 컴파일)
   - 장점 : 코드 Interpret 및 JIT compile 시간을 제거하여 performance가 향상됨
   - 단점 : 설치시점에 소스 코드를 번역하여 설치가 느리고, 파일을 따로 저장하기 때문에 용량이 커짐

JIT VS AOT

3.1 JIT Compiler

- 앱 실행 시 컴파일
- 장점 : 인터프리터 방식에 비해 성능 개선(약 10배 ~ 20배), 설치 시 컴파일을 하지 않기 때문에 AOT보다 설치 속도 빠름
- 단점 : 배터리 소모가 많아지고, 한번에 읽는 방식이기 때문에 화면 전환 시 속도가 느림, 실행 시 컴파일을 하기 때문에 AOT에 비해 실행 속도 느림
- 메모리를 적게 차지함, 즉 용량 작음

3.2 AOT Compiler

- 앱 설치 시 컴파일: 추가 컴파일 작업 없이 바로 실행
- 장점 : 사용자의 대기 시간이 거의 없음
- 단점 : 설치 시 컴파일된 파일을 따로 저장하기 때문에 프로그램 크기가 증가
- 안드로이드 누가 Nougat 버전 부터 JIT와 AOT를 모두 탑재함으로써 최초 설치시에는 무조건 JIT를 사용하도록 하여 설치시간과 메모리 용량을 적게 소모한 뒤, 차후 기기를 사용하지 않을 때나 충전 중일 경우 컴파일을 조금씩 하여, 자주 사용되는 앱을 AOT 방식으로 전환
- 설치 시 컴파일을 완료하기 때문에 JIT에 비해 설치 속도 느림
- 실행 시 컴파일을 하지 않기 때문에 JIT에 비해 실행 속도 빠름
- 용량 큼 (JIT에서 실행 시 컴파일하는 것을 미리 컴파일하여 가지고 있기 때문)