Infrastructure Middleware (Part 2): Android Runtime Execution Environment

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Understand the role of the execution environment in Android’s Runtime layer
Learning Objectives in this Part of the Lesson

• Understand the role of the execution environment in Android’s Runtime layer

• Know the two execution environments that have been part of Android’s Runtime: ART & Dalvik
Learning Objectives in this Part of the Lesson

• Understand the role of the execution environment in Android’s Runtime layer
• Know the two execution environments that have been part of Android’s Runtime: ART & Dalvik

Apps rarely access ART or Dalvik directly, but it’s useful to understand what they do.
Overview of the Android Runtime’s Execution Environment
Android Runtime’s Execution Environment

• Android’s Runtime layer is largely used to execute Java apps on mobile devices
Android Runtime’s Execution Environment

- Android’s Runtime layer is largely used to execute Java apps on mobile devices.
- These apps can also now run in the Chrome browser on laptops & desktops.

See arstechnica.com/gadgets/2016/04/it-looks-like-the-google-play-store-is-headed-to-chrome-os
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel.

See [en.wikipedia.org/wiki/Virtual_machine](en.wikipedia.org/wiki/Virtual_machine)
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel.
- Executes app byte code and/or native code (typically) inside a single Linux process.

See en.wikipedia.org/wiki/Virtual_machine#Process_virtual_machines
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
- Executes app byte code and/or native code (typically) inside a single Linux process
- This code generated is from Java source files by the javac compiler

```
java source files (.java)
class Foo {
    /* ... */
}
```

```
java bytecode files (.class/.jar)
... 
iconst_0
iaload
istore_1
jsr 19
iload_1
...
```

See source.android.com/source/jack.html
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
- Executes app byte code and/or native code (typically) inside a single Linux process
- This environment is created when a process starts & is destroyed when the process exits

See developer.android.com/guide/components/processes-and-threads.html#Processes
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel.
- Executes app byte code and/or native code (typically) inside a single Linux process.
- This environment is created when a process starts & is destroyed when the process exits.
- A process can run apps or system services.
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
- Executes app byte code and/or native code (typically) inside a single Linux process
- This environment is created when a process starts & is destroyed when the process exits
- It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
  - e.g., Intel x86, ARM, emulator, etc.
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
- Android apps typically run in their own process & own instance of the execution environment

See developer.android.com/guide/topics/processes/process-lifecycle.html
Android Runtime’s Execution Environment

- Android’s Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
- Android apps typically run in their own process & own instance of the execution environment
  - Apps can also share the same process

See developer.android.com/guide/topics/manifest/application-element.html#proc
Evolution of the Android Execution Environment
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
- Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode

See en.wikipedia.org/wiki/Dalvik_(software)
Evolution of the Android Execution Environment

• Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)

• Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode

• Dalvik uses a “register machine” model

See source.android.com/devices/tech/dalvik/dalvik-bytecode.html
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik uses a “register machine” model
  - In contrast, the Java platform uses a “stack machine” model

See en.wikipedia.org/wiki/Java_bytecode
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik uses a “register machine” model
    - In contrast, the Java platform uses a “stack machine” model
  - These two types of bytecode are not directly compatible
Evolution of the Android Execution Environment

• Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  • Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    • Dalvik uses a “register machine” model
  • dx program transforms Java bytecode in class files into .dex-formatted bytecodes

See sites.google.com/site/io/dalvik-vm-internals
Evolution of the Android Execution Environment

• Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)

• Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  • Dalvik uses a “register machine” model
  • `dx` program transforms Java bytecode in class files into .dex-formatted bytecodes
  • A Just-in-time (JIT) compiler was added to Dalvik later

See android-developers.blogspot.com/2010/05/dalvik-jit.html
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
- Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik uses a “register machine” model
  - `dx` program transforms Java bytecode in class files into .dex-formatted bytecodes
- A Just-in-time (JIT) compiler was added to Dalvik later
  - A JIT optimizes bytecode dynamically while a program runs

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced with an improved “Android Runtime” (ART)

See source.android.com/devices/tech/dalvik/art.html
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced with an improved “Android Runtime” (ART), e.g.,
    - An “ahead-of-time” (AOT) compiler
      - `dex2oat` compiles .dex files into native code when app is installed

See [source.android.com/devices/tech/dalvik/#AOT_compilation](source.android.com/devices/tech/dalvik/#AOT_compilation)
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced w/an improved “Android Runtime” (ART), e.g.,
    - An “ahead-of-time” (AOT) compiler
    - Better garbage collector (GC)
      - e.g., fewer GC pauses & parallel execution

See source.android.com/devices/tech/dalvik/gc-debug.html#art_gc_overview
Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don’t run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced with an improved “Android Runtime” (ART), e.g.,
    - An “ahead-of-time” (AOT) compiler
    - Better garbage collector
    - A JIT that further optimize ART’s AOT compiled code at runtime

See source.android.com/devices/tech/dalvik/jit-compiler.html
Evolution of the Android Execution Environment

Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features.
Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language

See docs.oracle.com/javase/tutorial/essential/concurrency
Evolution of the Android Execution Environment

• Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
  • Threading & synchronization mechanisms in the Java programming language, e.g.
  • Threads that run computations concurrently

See docs.oracle.com/javase/tutorial/essential/concurrency/threads.html
Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language, e.g.
    - Threads that run computations concurrently
    - Build-in monitor objects w/synchronization & notification features

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language.

My website contains many more digital learning resources related to Android & Java concurrency.

Digital Learning Offerings

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
Associate Chair of Computer Science and Engineering,
Professor of Computer Science, and
Senior Researcher in the Institute for Software Integrated Systems (ISIS)
at Vanderbilt University.

See www.dre.vanderbilt.edu/~schmidt/DigitalLearning
Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
- Threading & synchronization mechanisms in the Java programming language, e.g.

Android’s concurrency features involve capabilities at multiple layers
Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android’s execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language
  - There’s also support for multi-core hardware that’s now widely available for mobile devices

See www.androidauthority.com/fact-or-fiction-android-apps-only-use-one-cpu-core-610352
End of Infrastructure Middleware (Part 2): the Android Runtime Execution Environment