**Dalvik vs Android Runtime:**

Dalvik ist ein Just-in-time-Compiler (JIT Compiler). Dabei werden Programme oder Programmteile zur Laufzeit in Maschinencode übersetzt. Da die Kompilierung während der Ausführung des Programms durchgeführt wird, kann sie nicht beliebig aufwendig sein, da dies sonst die Ausführungsgeschwindigkeit des eigentlichen Programms merklich beeinträchtigen könnte. Daher beschränkt man sich meist auf häufig ausgeführte Programmteile. Diese sind typischerweise für den Großteil der Ausführungszeit des Programms verantwortlich, weshalb sich deren Kompilation und Optimierung besonders lohnt.

Die Aufgabe des JIT-Compilers ist es, diese Programmteile zu identifizieren, zu optimieren und anschließend in Maschinencode zu übersetzen, welcher vom Prozessor direkt ausgeführt werden kann. Der erzeugte Code wird meist zwischengespeichert, um ihn zu einem späteren Zeitpunkt der Programmausführung wiederverwenden zu können.

http://de.wikipedia.org/wiki/Just-in-time-Kompilierung

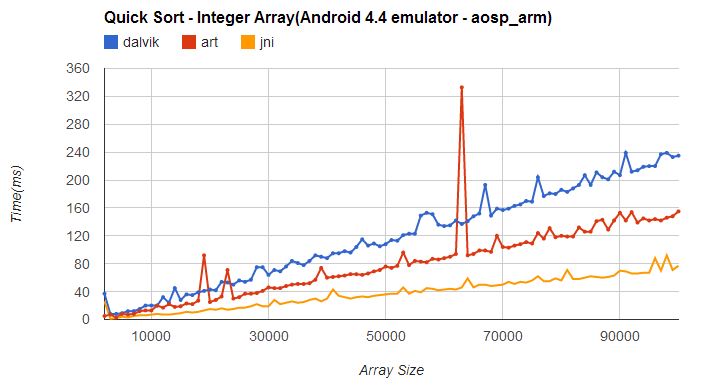
Android Runtime (kurz ART) ist eine Laufzeitumgebung die von Googles mobilem Betriebssystem Android ab Version 5.0 Lollipop eingesetzt wird.

ART löste damit die bisherige virtuelle Maschine Dalvik ab, die bis Version 4.4 (KitKat) im Einsatz war. Laut Google bietet ART eine bessere Performance und niedrigen Energieverbrauch als Dalvik. Dies wird sich in der Praxis in besserer Performance und längerer Akkulaufzeit bemerkbar machen.

http://de.wikipedia.org/wiki/Android\_Runtime

Bei Android Runtime handelt es sich um einen Ahead-of-time-Compiler (AOT-Compiler). Ein AOT-Compiler ist ein Compiler, der im Gegensatz zu Just-in-time-Compilern (JIT-Compiler) Programmcode vor der Ausführung in native Maschinensprache übersetzt. Dies hat den Vorteil, dass dieser Code zur Laufzeit wesentlich schneller ausgeführt wird als auf einem JIT-Compiler, da die Übersetzung bereits durchgeführt wurde. Dabei wird der Java-Bytecode bereits bei der Installation einer App in maschinenlesbaren Code vorkompoliert.

Der Nachteil an AOT-Compilern ist aber, dass dieser Code nicht mehr plattformunabhängig ist, wie es bei JIT-Compilern der Fall ist. AOT-Compiler sind die herkömmlichen Compiler wie sie schon von C eingesetzt wurden. Da Bytecode prinzipiell speichermäßig kleiner als Maschinencode ist, kann die Größe einer installierten App schnell auf bis zu zusätzlich 20 % ansteigen. Außerdem dauert die Installation einer App länger als zuvor, da der Bytecode bereits hier in Maschinencode übersetzt wird.



<http://de.wikipedia.org/wiki/Ahead-of-time-Compiler>

<http://www.droidwiki.de/Android_Runtime>

ART liefert Untersuchungen zufolge, in vielen Apps mehr Leistung als Dalvik. Grund ist, dass der Bytecode durch ART erheblich maschinennäher als Dalvik formuliert wird und somit weniger Rechenaufwand zur Ausführung vonnöten ist. Dies bedeutet, dass die Ressourcen des Geräts effizienter genutzt werden können und die gefühlte Performance ansteigt.

<http://www.androidnext.de/news/android-5-0-art-statt-dalvik/>

ART und Dalvik sind kompatiblel zueinander, das bedeutet Apps, die für Dalvik implementiert wurden, laufen auch auf ART. Die ART benutzt dabei denselben Input Bytecode wie Dalvik, geliefert mit Standard .dex-Dateien als Teil der Android Application Package-Dateien (APK-Dateien), wobei die .odex-Dateien durch Executable and Linkable Format (ELF) – Executables ersetzt werden. Da die Kompilierung nicht bei jeder Ausführung einer Applikation durchgeführt werden muss, wird die Prozessorauslastung verringert und dadurch die Akkulaufzeit verbessert.

**Neue Features in ART**

Verbesserte Garbage Collection:

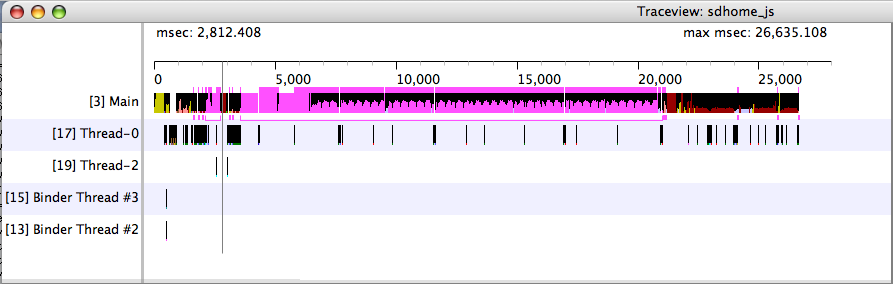
GC kann die Performance einer App beeinflussen, also abgehackte Bilder, langsame Reaktion auf Benutzerinteraktionen oder andere Probleme. Verbesserungen sind unter anderem, dass weniger Ausführungen des Garbace Collector’s gemacht werde und während der GC-Pause trotzdem Anwendungen parallel laufen können. Außerdem wird der GC kürzer ausgeführt für kurzlebige Objekte.

Verbesserungen für Entwicklung und Debugging

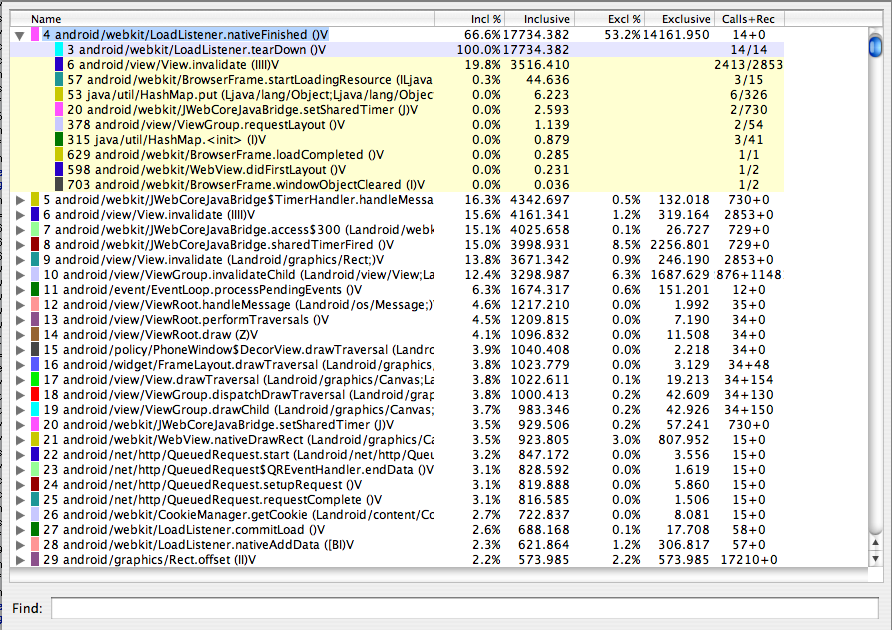
Für Dalvik wurde das Tool Traceview als Profiler verwendet, also zur Analyse des Laufzeitverhaltens von Software. Dieses gibt zwar sinnvolle Informationen, jedoch wird bei jedem Aufruf durch das Tool der Overhead größer, wodurch die Laufzeit beeinflusst wird.

Bei Traceview verwendet die Debug class um Informationen im Code zu protokollieren. Die log-Dateien können geladen werden und in 2 unterschiedlichen Panels angezeigt werden:

Timeline Panel: zeigt wann jeder Thread und jede Methode startet und stoppt



Profile Panel: stellt einen Überblick dar von Ereignissen innerhalb der Methoden.



ART unterstützt einen geeigneten Profiler der diese Einschränkungen nicht hat. Dadurch gibt es eine detaillierte Sicht über die Ausführung der App ohne eine erhebliche Verlangsamung.

Mehr Debugging Features

Man kann anzeigen welche Locks im Stack gehalten werden und zu dem Thread springen, der den Lock hält. Es ist außerdem möglich nachzusehen, wie viele Instanzen einer Klasse existieren und welche Referenzen diese haben. Zusätzlich gibt es sogenannte „method-exit“-events, die anzeigen, welche Werte bestimmte Methoden zurückliefern, und man kann „Watchpoints“ auf Members setzen und die Ausführung unterbrechen falls auf diese zugegriffen wird bzw. diese modifiziert werden.

Verbesserte Diagnose in Exceptions und Crash Reports

ART gibt so viele Details wie möglich falls Exceptions zur Laufzeit auftreten. Zum Beispiel zeigt die java.lang.NullpointerException Informationen darüber was die App mit dem Nullpointer versuchte zu tun, wie auf einen Wert zu schreiben oder der Versuch, eine Methode aufzurufen.

Beispiele:

java.lang.NullPointerException: Attempt to write to field 'int

android.accessibilityservice.AccessibilityServiceInfo.flags' on a null object

reference

java.lang.NullPointerException: Attempt to invoke virtual method

'java.lang.String java.lang.Object.toString()' on a null object reference

<http://source.android.com/devices/tech/dalvik/art.html>

<http://en.wikipedia.org/wiki/Android_Runtime>

http://developer.android.com/tools/debugging/debugging-tracing.html