Introduction to Android Programming

Curzel Federico

Android Programming

Part I - Android

- Introduction to Android
- The Dalvik Virtual Machine

Part II - Android Programming

- Java and ARM
- General lines about Android Programming
- Packing Application
- Sample Application in Java

Part III - Python as an alternative to Java

- Python on Android
- Java vs Python in Android context
- A sample application

Part I - Android

- **Introduction to Android**
- **The Dalvik Virtual Machine**

Introduction to Android

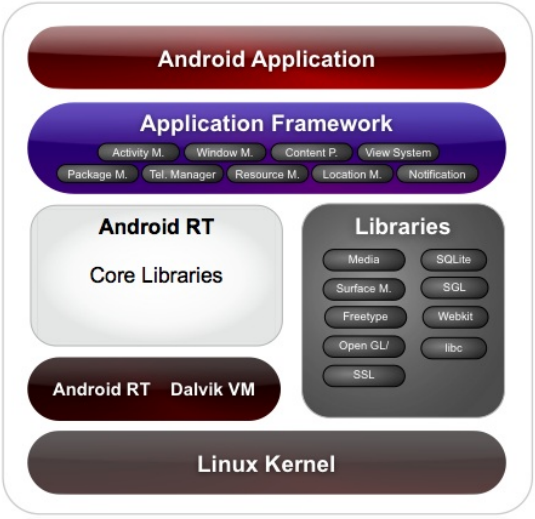
Introduction

Android is a Linux-Based OS which target-platform are mobile devices.
Nowadays, over the 60% of smartphones in the world are running Android.
It is open source, it has a good SDK, and it is Linux based.
This made developers quickly start to love Android.
Android is under ongoing development by Google, which also provides the Android SDK.
Each Android version corresponds to a specific API "level", more API levels are compatible each other.

Life, Death and Miracles

When	What
2003	Android development starts.
2005	Google buys Android Inc..
oct 2008	HTC Dream is the first Android phone to be commercialized.
apr 2009	Android 1.5 "Cupcake" released, based on Kernel Linux 2.6.
oct 2011	Android 4.0 "Ice Cream Sandwich" released, based on Kernel Linux 3.0.
q1 2012	More than 800.000 Apps on the Play Store.
q3 2012	1.5 Million new device activation per day.
q2 2013	Over 900 Million total activation.

Internal Structure



Linux Kernel

Provides basic functionality to the upper levels :

- Hardware Abstraction Layer
- C\C++ APIs.
- GCC.
- Memory Manager
- Scheduler

Dalvik Virtual Machine

See the following chapter for this.

Android Runtime

Consist of core C\C++\Java libraries used by the DVM.

- LibC
- OpenGL ES
- Webkit
- SQLite
- ...

Application Framework

- Development libraries
- Sdk Interface
- Windows Provider
- Package Manager

The Dalvik Virtual Machine

Introduction

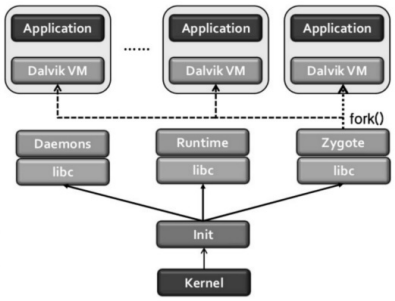
An OS offers a set of common APIs on every machine on which the system runs.
That allows to run the same kind of executable file on different platforms, or OS version.
This is possible in a lot of ways, such as cross-compiling the source, or execute it on a Virtual Machine.
In Android, users application are executed trough the Dalvik Virtual Machine.

Which are its functions?

- Executes **Dalvik Bytecode**.
- Manages **multiple instance** of itself (Like the JVM does.)
- Abstracts the problem of **memory menagement** to the OS.
- Manages multiple instance of itself (Like the JVM does.)
- Provides a **sandbox** where applications can run *without* interact each other.

Applications run on the DVM just like in a "sandbox".

Make it efficient



Every Android application runs on a single instance of the Dalvik Virtual Machine.
Zygote is the process that make the creation of new DVM instace efficient.

1. The Boot-loader loads the kernel and start init process.
2. Zygote process starts
3. Zygote creates a new instance of the DVM
   (Also preloads and pre-initializes core library classes.)
4. The system keeps in an idle state the process until the execution request.
5. Once an application execution request occurs,
   Zygote forks itself and create new process with the pre-loaded Dalvik VM.

Not a JVM clone

- **The architecture is Register-Based (instead of Stack-Based, like the JVM).**
  Up to 2^16 avaiable registers.

- **DVM uses the Dalvik Bytecode instead of the Java one.**
  Which is far more pragmatic :

```
                      iload s0
add-int d0, s0, s1    iload s1
                      iadd
                      istore d0
```

  The DVM instructions set has 218 opcodes.
  Compared to the JVM (which uses 200 opcodes) :
- 30% less instructions.
- 25% more code size.

- **Constant Pool**
  The dx compiler significantly reduces the size of the costant pool, by inling them directly in the bytecode.
  This permits the use of one single costant pool, unlike the JVM does.

- **It comes without exceptions handling.**

- **Null references**
  DVM does not specify a null type, the 0 value is used instead.

- **Ambiguos primitive types**
  The JVM is strongly typed, and uses different opcodes for every data-type.
  DVM does not distinguish between int/float/null, it uses aget (and aget-wide for double and long).

- **DVM is far more efficient.**
  A Register-Based architecture is up to 47% more efficient than a Stack-Based one!
  The larger code size involves only 1.07% extra real machine loads per VM instruction, which is negligible.

Part II - Android Programming

- **Java and ARM**
- **General lines about Android Programming**
- **Packing Application**
- **Sample Application in Java**

Java and ARM

Java, over time, has become one of the most widely used programming languages.
J2ME is a runtime and a collection of APIs for software development dedicated to devices with limited resources.
The portability of Java, and all the advantages of J2ME, had soon led it to become a de-facto development standart for mobile devices.

On the other side, the main problem of ARM architecture is the limited computational power.
A solution that has been found was to make cpu natively execute Java Bytecode.
The following is an overview of technologies used in ARM Architectures to treat instructions.

The conclusion is that nowadays there is no more a "special way" to execute Java in ARM CPUs.

Jazelle

Jazelle DBX (Direct Bytecode eXecution) allows some ARM processors to natively execute Java bytecode.
The target of this tecnology is to facilitate the execution on Java ME programs on mobile devices.
The BXJ (Branch and eXchange to Java) instruction attempts to switch to Jazelle state.
If allowed and successful, sets the 'J' bit in the CPSR; otherwise, it "falls through" and acts as a standard BX (Branch) instruction.

Thumb

To improve compiled code-density since 1994 arm processors have featured Thumb instruction set, which have their own state.
When in this state, the processor executes the Thumb instruction set, a compact 16-bit encoding for a subset of the ARM instruction set.
Most of the Thumb instructions are directly mapped to normal ARM instructions.

Thumb2

Both the ARM and the Thumb instructions sets were exdended.
A stated aim for Thumb-2 was to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory.
As you can see below, the same source code can so be compiled as ARM or Thumb2 code.
Recall that the Thumb MOV instruction has no bits to encode "EQ" or "NE", as MOVEQ and MOVNE are ARM instruction.

code	pseudo-code	ARM meaning	Thumb2 meaning
`CMP r0, r1`	--	comparison	comparison
`ITE EQ`	if r0 == r1	--	starts an If-Then-Else block
`MOVEQ r0, r2`	then r0 = r2	conditional	"Then" block
`MOVNE r0, r3`	else r0 = r3	conditional	"Else" block

ThumbEE

ThumbEE, also known as Jazelle RCT (Runtime Compilation Target), was announced in 2005.
It added a fourth processor mode, and made small changes to the Thumb-2 extended Thumb instruction set.
These changes make JIT compilers able to output smaller compiled code without impacting performance.
ThumbEE is a target for languages such as Java, C#, Perl, and Python, which all uses JIT compilers.

AArch64e

AArch64 is a 64-bit architecture, main news are :

- New instruction set, A64
  - 31 general-purpose 64-bit registers
  - Instructions are still 32 bits long and mostly the same as A32
  - Most instructions can take 32-bit or 64-bit arguments
  - Addresses assumed to be 64-bit
- Advanced SIMD (NEON) enhanced
  - Has 32x128-bit registers (up from 16), also accessible via VFPv4
  - Supports double-precision floating point
  - Fully IEEE 754 compliant
  - AES encrypt/decrypt and SHA-1/SHA-2 hashing instructions also use these registers
- A new exception system
  - Fewer banked registers and modes

Kernel Linux 3.7 included pathes to support AArch64e.

General lines about Android Programming

Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools.
These include libraries, documentation, sample code, and tutorials.
The SDK also contains a Debugger Tool (ADB) and an Android emulator for x86 systems.
Obvioulsly, there also are tools for packing application, as explained in the next chapter.

Android NDK

The NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++.
This can be helpful in reusing existing code libraries written in these languages.
Most apps don't need the Android NDK.

Android Debug Bridge

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package.
It consists of both client and server-side programs that communicate each other via USB.
A typical use consist of compile a project and run it directly on a physical device.

ADT Bundle

ADT Bundle is the "official" IDE for Android development, base on Eclipse.
I has been create to maintain the same interface of different OS and to keep togheter all the mentioned softwares.

Project Sections

The project folder contains three main directories :

- **Gen**,
  which contains Java Library automatically generated.
  In particular, an "R.java" file, containing exadecimal values refereing to gui elements.
- **Src**,
  which contains the source code of the application.
  It corresponds to the Java default source package.
- **Res**,
  which contains all the resources used by the application.
  Its subdirectory, "drawable", contains all the graphic resource such as image and so on.
  The GUI Layout is defined in a xml file, contained in this directory.

Activity

The concept

An activity is a single, focused thing that the user can do.
Almost all activities interact with the user, so the Activity class takes care of managing graphics resources.
Every application is build around a single instance of the Activity class.
All the activity classes must have a corresponding <activity> tag in their package manifest file.

An application might contains more activities, all these run on a single instance of the DVM.

Some Methods

- **onCreate(), onStart(), onResume(), onDestroy(), ...**
  As easy to understand, these are methods used to interact with the flow of the application lifecycle.
  Override these methods to initialize it on start or modifing it when exiting.

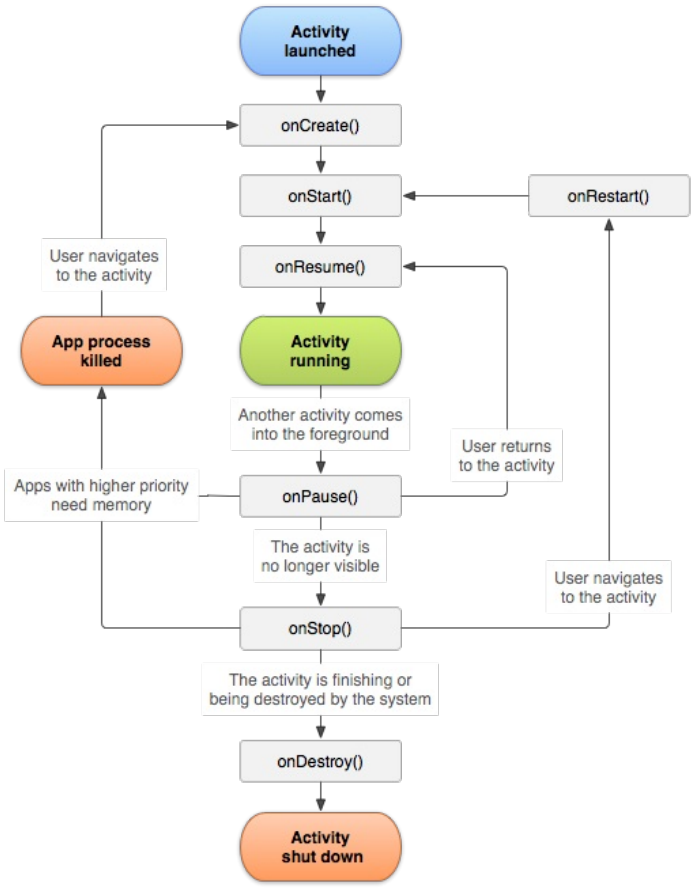
- **setContentView()**
  Set the activity content to an explicit view.
  This view is placed directly into the activity's view hierarchy.
  It can itself be a complex view hierarchy.

- **onActivityResult()** Let's suppose an application needs the user to select a contact from the contect list.
  Calling another application is as simple as this :

```
startActivityForResult(
    new Intent( Intent.ACTION_PICK,
        new Uri( "content://contacts" )
    ),
    PICK_CONTACT_REQUEST
);
```

  Override onAcitvityResult() let you use the result of the requested application, by using an Intent object.

Lifecycle Diagram



Manifest

In software packaging, it is common to list the contents of a distribution in a manifest file.
In java this kind of file is used to specify information used when packing in a jar file :

- store hashes of stored files for signature validation.
  This aspect will be deepened in the next chapter, "Packing Applications".
- sealing jar files (i.e. ensure that only classes from this jar file are loaded in the packages defined in this jar file).
- store version/product/producer information to be readable at runtime

This file also contains information about the permission of the application.
The ability to start a particular Activity can be enforced when it is declared in its manifest's <activity> tag.
Other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start that activity.

Layout

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.
You can declare a layout in two ways:

- Declare UI elements in XML.
  Android provides a straightforward XML vocabulary that corresponds to the View classes, such as those for widgets and layouts.
- Instantiate layout elements at runtime.
  Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework allows you to use either or both of these methods for declaring and managing your application's UI.
For example, you could declare your application's default layouts, screen elements, and their properties in a XML file,
and then modify the state of those object in yout code, at run time.

Packing an Application

Introduction

Compile a programm means translate it in an other language (e.g. C to assembly, or Java to Bytecode).
When compiling an application for Android, our target-language is the Dalvik Bytecode.
The compiled program is stored in a file *.dex. Android applications are packaged in .apk format and stored int the /data/app folder on the Android OS.
APK package contains the .dex files, resource files and so on.

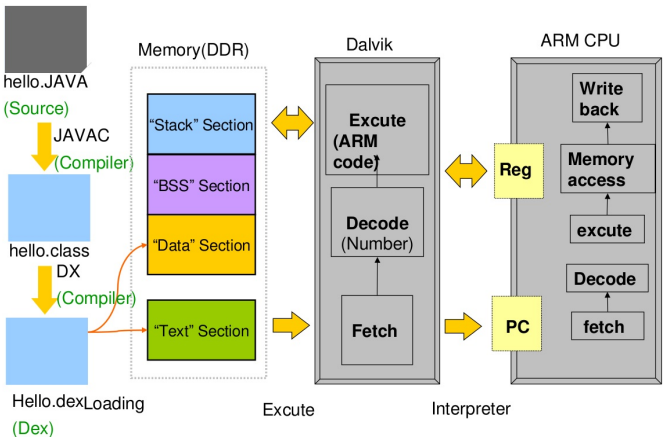
Structure of a DEX file

- **Header**
- **Constant Pool**
  - References to other classes
  - Method names
  - Numerical constants
- **Classes definition**
  - Access flags
  - Class names
- **Data**
  - Method code
  - Info related to methods
  - Variables

Building an APK file

Application Package File (APK) is the file format used to distribute and install application on Android.
Any program, to run on Android, needs to be packed in an apk file.
The apk files contains the application code, the dex file, and all the resources needed.
Those information are stored in a special JAR file (using ZIP), with ".apk" as file extensions.
The application can be compiled using the ADT plugin.
After this, the apk needs to be signed, using Jarsigner, and aligned, using Zipalign.

Runtime through the Dalvik Virtual Machine

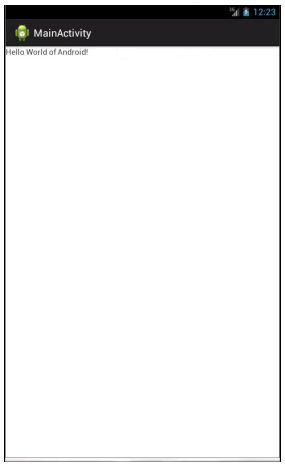


A compiled program is loaded by the DVM and unpacked.
Application memory is divided in various sections (like the 8086 segments).

- Stack Section
- Data Section
- BSS Section
  (A special Part of the data segment, for statically-allocated variables.)
- Text Section (A special segment for strings treatment

A sample Application

Hello World!




```
package com.curzel.sample.helloword;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {

 @Override
 protected void onCreate( Bundle savedInstanceState ) {
  super.onCreate( savedInstanceState );
  TextView text = new TextView( this );
  text.setText( "Hello World of Android!" );
  setContentView( text );
 }
}
```

The Bundle class

A bundle is generally used for exchange data among various Activities of android.
The following code will start up an activity and pass to it some values.

```
Intent intent = new
Intent( getApplicationContext(), SecondActivity.class );
intent.putExtra( "myKey", someValues );
startActivity( intent );
```

Then, in the second activity, you can use the bundle just like an HashMap.

```
Bundle extras = intent.getExtras();
Object someValues = extras.get( "myKey" );
```

Part III - Python as an alternative to Java

- **Python on Android**
- **Java vs Python in Android context**
- **A sample application**

Python on Android

Python is a multi-paradigm programming language, initially developed by Guido Van Rossum, in 1989.
In analogy with Java, Python is both compiled and interpreted.
It is often used as a scripting language, but is also used in a wide range of non-scripting contexts.
Using third-party tools it can also be packaged into standalone executables.

In Pills

- Emphasize the readability
- Concise Syntax
- Multi-Paradigm
- Duck Typing
- No modifier, but Decorators
- Completely Cross-Platform

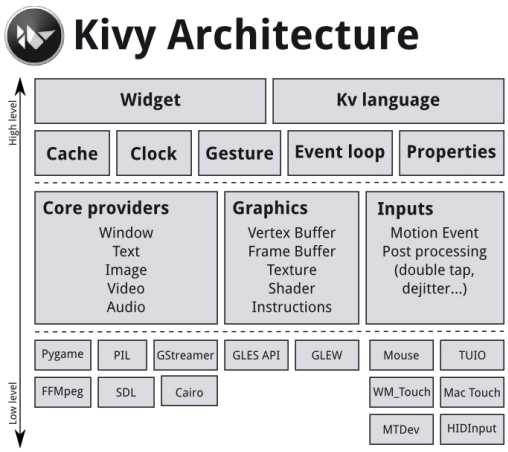
SL4A

As mentioned, Python runs on an interpreter.
This allows to develop multiplatform code, just like Java.
The Scripting Layer for Android (SL4A) allows to run various script directly on Android devices.
Supported languages includes Python, Perl, Ruby, Lua, JavaScript and so on...
SL4A is designed for developers and is still bt an alpha quality software.

Python for Android

Python for android is a project to create your own Python distribution, with all the modules you need.
The distribution is then used to compile an Android application using ADT.
It is easy to access Java classes from python, as the next chapter explains.

Kivy



Kivy is a cross-platform Python framework for rapid development of application.
Using python-for-android technology to create a python distribution that includes kivy
allows you to run the same application code on any Android device.
All without modify a single line of code.
This is possible because the framework itself is written in pure python.
As many other frameworks do, kivy provides a set of classes for building the GUI,
but also to manage internal process and touch input.
However, Kivy comes with a further kind of approach.
Both GUI layout and widgets can be described using the Kv Language.
This technology is cross between UIML (Layout definition) and XML (Structure definition).
The second approach makes kivy easily extensible.

Java and Python on Android

Accessing Java classes in Python

Both the SDK and the NDK are written in C, C++ and, of course, Java.
The usage of Java classes in a Python program is possible in several ways.
A good solution consist of use Jython, a Python interpreter written in Java.
An other option is using the Pyjnius library, which is what Python-For-Android does :

```
from jnius import autoclass

Stack = autoclass('java.util.Stack')
stack = Stack()
stack.push('hello')
stack.push('world')

print stack.pop() # --> 'world'
print stack.pop() # --> 'hello'
```

Python-For-Android internally mixes Cython, JNI (Java Native Interface) and Pyjnius.
The cost in terms of efficiency is minimal, and provides a more flexible interface.

While Pyjnius sounds to have no limitation, Kivy does not officially supports the whole SDK.
Basic Hardware Interface such as GPS or Bluetooth API are currently *not directly* avaiable through Kivy.

Table of concepts

<table>
<tr><th>How do you...</th><th>Java</th><th>Python</th></tr>
<tr><td>...define a job for the DVM?</td><td>Using the Activity class.</td><td>Using the App class.
onCreate(Bundle) -> build()
onPause(Bundle) -> on_pause()
and so on...</td></tr>
<tr><td>...interact with an other application?</td><td>Using a Bundle object.</td><td>Actually, there is no way of doing this.</td></tr>
<tr><td>...handle touch events?</td><td>Overriding the onTouchEvent method.</td><td>Overriding respectively :
on_touch_down
on_touch_move
on_touch_up
You also can grab a certain touch to a Widget.</td></tr>
<tr><td>...handle multi-touch events?</td><td colspan="2">Using the touch-event unique id.</td></tr>
<tr><td rowspan="2">...make 2D drawing?</td><td colspan="2">In both languages, you will simply need to override the atomic Widget\View class.</td></tr>
<tr><td>You need to extend the View class and override the onDraw(Canvas c) method.</td><td>Every Widget object as a canvas property, which you can treat in different ways :
Using Kv-Language, expecially for static things, in comfortable a separeted file.
Using the Python with statement.
Or using it as a python variable (not so widely used).</td></tr>
<tr><td rowspan="2">...make 3D drawing?</td><td colspan="2">Of course, you will need to pass through OpenGL ES, which APIs are provided by the Application Framework.
In both languages however you will not have the same set of API you will have on a desktop app.</td></tr>
<tr><td>Just use the classes of the OpenGL package in the NDK.
While it not differs a lot from JOGL or Java3D, you will have a different set of APIs.</td><td>Kivy have its own OpenGL wrapper, which offers common APIs on every platform.
3D drawing is currently in a "beta" phase.
However, it does not have the same set of API of PyOpenGL.</td></tr>
</table>

Kipycalc - Programmable Calculator

What is that

As IT student, during the last year I've to work with lots of calculations...
Some frequent tasks were :

- Solve expression with **Complex Number** (Electronics)
- Use of **Derivatives** (Math)
- Use of **Integrals** (Math)
- Operation with **Matrices** (Statistic)
- Use of **Alghoritms** (Almost all subjects)

The problem was that, after the first month, the calculator was not enough.
I didn't want to buy a (costly) programmable calculator, so I start thinking on a Computer Algebra System for Android.
As I heard of python-for-android, I immediately started working on this project.
Today, with Kipycalc you can easely :

- Run python statements.
- Do comlpex calculations.
- Do symbolic calculations.
- Plot functions and expressions.
- Integrate, derivate and evaluate an expression.
- Quickly use "standard" scientific calculator tasks.

Links

This app is open source, and is avaiable on :

- **Github.com**
- **Play Store**

Screenshots