Version 1.0

2018-Jun-27



Android Architecture

Homework

Presenter: Arturo Sanchez

Mobile Apps Company

3301 Windy Ridge Parkway, Suite 400, Atlanta, GA 30339

# **Android Architecture**

## Introduction

### Background

Android is a Mobile Operating System developed by Google, based on modified version of the Linux Kernel and other open source software and designed primarily for touchscreen mobile devices such as smartphones and tablets.

## Android Platform Layers



|  |  |
| --- | --- |
| Layer | Description |
| Linux Kernel | It is the foundation of the Android platform, because all of the functionality it is done by this layer (passed through the Android Runtime (ART).  Allows device manufacturers to develop hardware drivers for a well-known kernel. |
| Hardware Abstraction Layer (HAL) | Provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.  Consists of multiple modules, each of which implements an interface for a specific type of hardware component (Example camera or Bluetooth) |
| Android Runtime | It is an application runtime environment used by the Android Operating System.  It’s optimizes the execution of applications by continually profiling application each time they run and dynamically compiling frequently executed short segments of their bytecode into native machine code. |
| Native C/C++ Libraries | It contains the libraries exposed for the Java Framework API to use.  You can use the Android NDK to access some of these native platform libraries directly from your native code. |
| Java API Framework | All of the Android OS features are available through APIs written in the Java language.  The Apps use this API Framework based on Managers to get the desired functionality.  Example.- If you want to use the Location of the device you use the Location Manager. |
| System Apps | Android comes with a set of core apps for several basic functionality.  A third-party app can become the user’s default app because there is no special status among the system apps. |

## Android Components

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

* Activities
* Services
* Broadcast receivers
* Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

### Activities

An activity is the entry point for interacting with the user.

It represents a single screen with a user interface.

An activity facilitates the following key interactions between system and app:

* Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
* Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
* Helping the app handle having its process killed so the user can return to activities with their previous state restored.
* Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

### Services

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.

A Service is a component that runs in the background to perform long-running operations or to perform work for remote processes.

A service does not provide a user interface.

There are two type of services:

* Started Services
  + Tell the system to keep them running until their work is completed
* Bound Services
  + Run because some other app (or the system) has said that it wants to make use of the service.
* A service is implemented as a subclass of Service.

### BroadCast Receivers

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.

Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that are not currently running

Although broadcast receivers do not display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object

### Content Providers

A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access.

Through the content provider, other apps can query or modify the data if the content provider allows it.

To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme. Thus, an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities, which can in turn use them to access the data.

Content providers are also useful for reading and writing data that is private to your app and not shared.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions.

## Lint

Android Studio provides a code scanning tool called lint that can help you to identify and correct problems with the structural quality of your code without your having to execute the app or write test cases.

Each problem detected by the tool is reported with a description message and a severity level, so that you can quickly prioritize the critical improvements that need to be made. In addition, you can lower the severity level of a problem to ignore issues that are not relevant to your project, or raise the severity level to highlight specific problems.

It is available both as a command line tool, as well as integrated with Eclipse, and IntelliJ.

Here are some examples of the types of errors that it looks for:

* Missing translations (and unused translations)
* Layout performance problems (all the issues the old layoutopt tool used to find, and more)
* Unused resources
* Inconsistent array sizes (when arrays are defined in multiple configurations)
* Accessibility and internationalization problems (hardcoded strings, missing contentDescription, etc)
* Icon problems (like missing densities, duplicate icons, wrong sizes, etc.)
* Usability problems (like not specifying an input type on a text field)
* Manifest errors
* And many more.

## Units of Measure

The Android framework supports a specific set of units of measurement to control the place and size of visual objects. These are known as dimensions and dimension values.

There are many times when visual components are set to stretch automatically or place themselves automatically relative to a container or other object.

Whenever you set the placement of an object yourself, or set its size explicitly, you need to know how to use these units of measurement. You can use them either in XML layout files, or programmatically with Java.

**Density-independent Pixels (dp)**

An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. When running on a higher density screen, the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi. Likewise, when on a lower density screen, the number of pixels used for 1dp is scaled down. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Using dp units (instead of px units) is a simple solution to making the view dimensions in your layout resize properly for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.

**Scale-independent Pixels (sp)**

This is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and the user's preference.

**Points (pt)**

1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

**Pixels (px)**

Corresponds to actual pixels on the screen. This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

**Millimeters (mm)**

Based on the physical size of the screen.

**Inches (in)**

Based on the physical size of the screen.