Android Crash Course by Kevin McMahon

Last updated on October 29, 2013

Getting Started

IDE

- Android Studio

Packaging

- Gradle
- Maven with Android Maven Plugin
  - Good Intro to Maven series

Key Development Frameworks

- Android Compatibility Library
- Dagger
- Retrofit
- Picasso
- Otto
- OkHttp
- ViewPagerIndicator
- Polaris
- Crouton
- GSON
- Commonsware

Testing Frameworks

- FEST Android
- Robolectric
- Robotium

Guides, Tutorials and Links

Official Google Resources

- Developer Site
- Official Android Training Site
- Android Developers Youtube Channel

Community Resources

- AppDevWiki - Android Frameworks
- #AndroidDev on Google+
- AndroidDevWeekly.com Newsletter
- AndroidWeekly.net Newsletter
- Android Toolbox

Google I/O Sessions

- The World of ListViews
- REST Webservices
- Android Developer Tools
- Android Pro Tips
- Memory Management for Android Apps

Tutorials, Talks and Guides

- 31 Days of Android
- What New Android Developers Need to Know
- Vogella Tutorials
- Understanding LinearLayouts and Gravity
- Android App Anatomy (Talk + Slides)

Open Source Apps

- ShipFaster - demo of Square OSS libraries
- TweetLanes
- Github for Android
- Gaug.es
- Google I/O
- MusicBrainz
- Astrid
- SeriesGuide
- Photup

Quick Refresh of Android System Stack

What is Android

- Application Framework
- Dalvik Virtual Machine
- Customized Linux kernel
- Optimized OpenGL graphics
- Rich development environment

Dalvik

- runs multiple VMs efficiently
- requires a .class to .dex transformation
- JIT (as of Android 2.2)

Each Android Application:

1. Runs in their own process
2. Runs on their own VM

Native

- NDK vs SDK
- What purpose does the NDK serve?

Android Application Concepts

Activities

- orchestrates user interface views
- apps are composed of 1-to-n number of activities
- one activity required to be marked as main and shown first upon launch
- each activity is given a default window to draw in.
- contents of the window is provided by a hierarchy of views

Services

- “headless” activities similar to traditional services in Unix and Windows environments
- possible to bind to an ongoing service and communicate via exposed interface
- runs in main application process but doesn’t block other components or UI

Content Providers

- provides a contract/API for accessing data that an app exposes
- controlled access to data
    - contacts, music, video, pictures, etc

Intents

- Eventing mechanism
- Intent objects are passive data that is of interest to the component that is receiving the intent
- Filterable

Resources

- Images, layout descriptions, binary blobs and string dictionaries
- Abstraction layer which helps decouples code
- Makes managing assets easier
    - Localization
    - Multiple displays
    - Different hardware configurations

Fragments

- A self-contained component with its own UI and lifecycle.
- Can be reused in different parts of an application’s user interface depending on the desired UI flow for a particular device or screen.
- Can be thought of as a mini-activity
- First introduced in Android 3.0 Honeycomb but backwards compatibility via support library exists.
- Android Fragment Transactions

Android Life Cycle

- ***You have to know this cold***
- Application
- Activity
- Saving and restoring state
- Impact on fragments
- Fragment life cycle

Android Project Setup and Configuration

- What API Level Should I Target?

UI View Components

View Hierarchies

- Tree of elements compose views
- XML lends itself nicely

Root View

- DecorView (internal class but represents the device’s viewport)
    - Typical setup
        - FrameLayout
        - LinearLayout with multiple FrameLayouts (title bar in one, screen other)
- Most important node is the FrameLayout w/ “id/content”

Content View

- setContentView() replaces everything under the current content node with tree view specified
- Process of loading and merging called layout inflation
- Not directly from XML but from the binary format Android creates during build
- after inflation the views are added to rendering chain and can be drawn

Rendering of Views

- Done in two passes
    - measuring
    - layout
- Best Practices
    - Draw only what is necessary
        - use ViewStub
        - Toggle visibility: View.GONE, View.INVISIBLE, View.VISIBLE
    - Avoid clutter
        - reduce and simplify
        - clean layouts are better for users and better for performance

    - Reuse Views
        - cache
        - ViewHolder

    - Avoid excessive nesting
        - Leverage RelativeLayouts

    - Avoid duplication by using these tags:
        - `<include />`
        - `<merge />`

Arranging Layouts

- layouts vs. layout managers
    - layouts == XML
    - layout managers == ViewGroups

- classes
    - android.View
    - android.Widget

- layout attributes and parameters
    - attributes
        - two different types of attributes
            - view class
            - view parent class (layout managers)
            - Example: TextView
                - class attribute : android:text
                - parent attribute : android:padding

        - Find all of them here: android.R.attr

    - parameters
        - width and height
            - MUST always be present on any Android view
            - can take specific values (100px, 100dp, 100sp)
            - can take special values
                - fill_parent (match_parent) : take as much room as possible
                - wrap_content : take only what you need

        - margin and padding
    - ids
        - + means create this id if you haven’t already

4 main layout managers

FrameLayout

- Everything top left stacked on top of each other

LinearLayout

- supports horiz (default) or vert orientation
- respects order of elements

RelativeLayout

- layout elements relative to each other
- example: android:layout_rightOf

TableLayout

- LinearLayout child
- adds TableRow container to hold table cells
- each cell must be an individual view which can be layout manager or view group

Security

Linux Security Model

- users (UID) and groups (GID) concept
- permissions
    - if you are not explicitly granted the permission, you don’t have it
    - permissions are assigned to each resource
    - resources are typically files
    - defined owner for each resource (UID)
    - defined groupd for each resource (GID)
    - each resource has 3 sets of permissions: owner, group, and world
    - 1 File : RWX for Owner, RWX for Group, RWX for World

Android Security Model

- based on Linux so it follows that the Linux model underlies
- when app is installed
    - new UID is created for the app and applied to app resources (in the Linux sense)
        - Isolation == Separation of Control (SoC)

    - SoC : all data stored by app are given full permissions associated to the app UID and no permissions otherwise
    - UID's are unique to the device. Same app on two different phones have no guarantee that they have the same UID

- You can:
    - run other components under the same UID
    - store data on removable media (SD cards)

- BUT:
    - erodes SoC

Android File System Isolation

- data stored : /data/data/app_package_name
- inside that directory : ./files directory is created and assigned UID and full permissions of the app as app is owner
- isolation
    - By default, when new files are created, permissions are set to give the app's UID full control
    - no other permissions are set

- Four caveats
    - apps that run with the same UID can access each others files
    - root UID can access ANY files on the device
    - data written to external storage (aka SD cards) don't use the Linux permissions-based controls. (data on SD is accessible by anyone)
    - developers CAN change permissions on the files and make them available to others
        - when you create a file, you can modify that default by supplying one or more of the filesystem permission flags to the openFileOutput() method call.

            - MODE_PRIVATE (DEFAULT)
            - MODE_WORLD_WRITABLE (All apps can write to this file)
            - MODE_WORLD_READABLE (All apps can read this file)

        - code: OutputStreamWriter out = new OutputStreamWriter(openFileOutput("scores", MODE_WORLD_READABLE | MODE_WORLD_WRITEABLE));

- Rule of Thumb : assign app resources such as files, in this case, just enough permissions to do what needs to be done and no more.

Application Signing, Attribution and Attestation

- Apps ***must*** be digitally signed before installation on device
- Why?
    - identifies who created the app
    - allows apps signed by the same creator to interact with each other at a higher degree

- Digital Signature
    - cryptographic construct that a developer applies to software to prove that they wrote it.
    - tech version of your hand-written signature
    - made possible by two things:
        - digital certificate identifies each developer (your development "driver's license"
        - private key (effectively a really long and random number)

    - digital certs
        - provided by a CA (DMV in the license metaphor)
            - will have to prove who you say you are
            - can tell that multiple pieces of software are from same digital certificate
            - can identify who wrote it because cert is assocated to identify via the CA

        - self-signed certificate (wax seal).
            - don't have to prove identify
            - can tell that multiple pieces of software are from same digital certificate
            - cannot associate identity to cert

- apps ***DO NOT*** require certs that are provided from a CA
- Debug Signing : APK generated signed with debug key/certificate with default debug credentials created when Android SDK installed
- Release Signing : APK created unsigned and then ADT tool jarsigner used to sign with release credentials
- you can shareUserId between apps and subsequently share data access
- Components in the same package can run in separate processes
    - allows components that are part of different apps but written by the same developer to run in the same process,
    - lets components that are part of the same app to run in different processes.

- Components can run in separate, private process
    - android:process=”:com.example.processIdNumber5” (the colon makes it private)
    - prevents other components from sharing process
    - if component crashes, won’t crash other components. (isolation)

Android Preferences and Database Isolation

SharedPreferences

- KVP
- primitives
- SharedPreferences object in app is a representation of an XML file on the file system
    - stored in ./shared_prefs directory under /data/data/app_package_name directory
    - getSharedPreferences() method takes same args as openFileOutput() so same file permissions apply

SQLite

- more structured data than KVP or flat files
- stored in the ./databases
- openOrCreateDatabase() have same args as openFileOutput() and getSharedPreferences()

Application Permissions

- Uses install-time permission request model
- app specifies in manifest which of these permissions it requires

Steps to Manually Build a Release APK

1. Generate Release Key
   ```
   $ keytool -genkey -v -keystore sampleapp.keystore -alias
   sampleapp -keyalg RSA -keysize 2048 -validity 10000
   ```
2. Build in release mode
   ```
   $ ant release
   ```
3. Sign and Verify
   ```
   $ jarsigner -verbose -keystore keystore/sampleapp.keystore
   bin/sampleapp-release-unsigned.apk sampleapp
   $ jarsigner -verbose -verify bin/sampleapp-release-
   unsigned.apk
   ```
4. Zipalign
   ```
   $ zipalign -v 4 sampleapp-release-unsigned.apk sampleapp-
   release.apk
   ```

Tools

SDK Tools (most useful tools in bold)

- **`android`** : Manages Android virtual devices (AVD) which the emulator uses, creates/updates projects and libraries, updates the SDK components
- **`ddms`** : provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.
- `dmtracedump` : gives you an alternate way of generating graphical call-stack diagrams from trace log files.
- `draw9patch` : Tool which allows you to easily create a ninepatch graphic using a WYSIWYG editor.
- **`emulator`** : A virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device.
- **`hierarchyviewer`** : Allows you to debug and optimize your user interface.
- `hprof-conv` : Tool that converts the HPROF file that is generated by the Android SDK tools to a standard format so you can view the file in a profiling tool of your choice.
- `mksdcard` : Tool that lets you quickly create a FAT32 disk image that you can load in the emulator, to simulate the presence of an SD card in the device.
- `monkey` : A program that runs on your emulator or device and generates pseudo-

random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events.
- `monkey runner` : A tool provides an API for writing programs that control an Android device or emulator from outside of Android code. *Not* related to the monkey tool mentioned above.
- `traceview` : Graphical viewer for execution logs saved by your application.

Build Tools

- **Android Development Tools (ADT)**
  - Includes the tools that compile, transform and package your Android code.
  - Tools used in this process include : `aidl`,`aapt`,`dexdump`,`dx`

- `proguard` : ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names.
- `lint` : Scans Android project sources for potential bugs.
- `zipalign` : Archive alignment tool that provides important optimization to Android application (.apk) files.

Platform Tools

- **`adb`** : Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.
- **`logcat`** : The Android logging system provides a mechanism for collecting and viewing system debug output.