

# Android cross-platform support study

**Name:** Xinwen Li

## About this study

Android has been most popular developing platform provide useful services and features in recent years. Android apps run within the Dalvik Virtual Machine which is similar to Java VM optimized for devices with limited memory and processing capacity. Android app with Java source code are compiled into the Java byte code using the Java compiler. However, this approach introduces difficulties for developers familiar with other languages rather than Java. Then people dive into a solution build runtime library support developing language other than Java. This solution allow developers develop application use same source code for different platforms. For example, Xamarin allow developers develop application for Android, IOS, and OS X using C# and the .NET framework.

While it brings beneficial allow developers use same source code for different platforms, it also introduces difficulties reverse engineering the application with third-party language support runtime libraries. This study focus on third-party runtime libraries on Android platform.

## Cross-platform fundamentals

### Pros and cons of cross-platform support

Pros:

- Allow developers leverage existing language skills.
- Code re-use.

Cons:

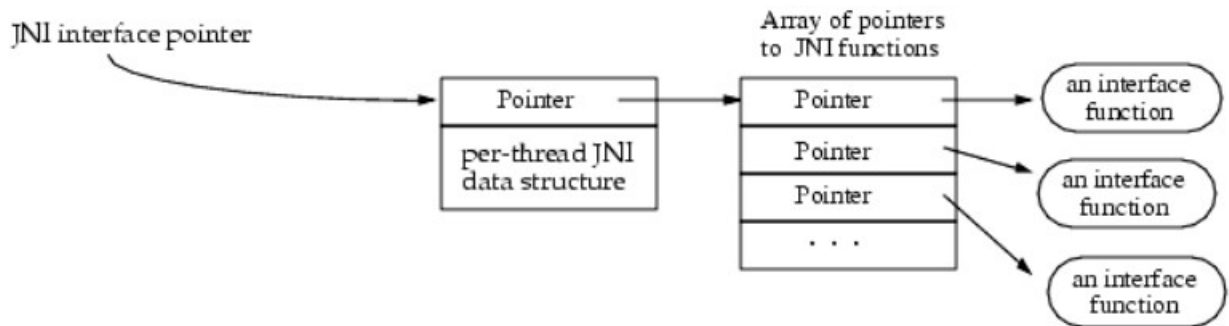
- Third-party architecture licensing requirement.
- Update delay.
- Performance and bugs.
- Understand memory manage and API calling.
- Extra runtime libraries stored.

### How does cross-platform work?

The trick is utilize Java Native Interface (JNI). JNI is a framework that allows non-Java code to be called by a Java code running inside a JVM.

## JNI Interface Functions and Pointers

Native code access Java VM features by calling JNI functions. Functions are pointed by an array of pointers pointed by JNI interface pointer. See figure below:



Native methods receive JNI interface pointer as an argument. For the same Java thread, multiple native method calls are guaranteed to get the same interface pointer. For different different Java threads, it's possible to get different JNI interface pointers.

## Compiling, Loading and linking Native Methods

Since the Java VM is multithreaded, native libraries should also be compiled and linked with multithread aware native compilers. e.g -D\_REENTRANT flag for gcc compiler.

Native methods are loaded with System.loadLibrary method. The system will follow a standard convert library name to a native dynamic linking library name. For example, Solaris system converts libXXX to libXXX.so and Win32 system converts libXXX to LibXXX.dll.

If operating system does not support dynamic linking, the VM completes the System.loadLibrary call to prelink the native methods with VM.

## Native Method Arguments

JNI interface pointer is the first argument of type JNIEnv. The second argument depends on method type. If native method is static, the second argument refers to its Java Class. If the method is nonstatic, the second argument refers to the object. The possible remaining arguments depends on the native method arguments.

## Referencing Java Objects

Primitive types such as int, char are copied between Java and native code. Arbitrary Java objects are passed by reference. The VM keeps track of all objects that have been passed to native code. The native code must inform VM the objects are no longer needed.

## Accessing Java Objects, Primitive Arrays, Fields and Methods

- Java Objects  
Java VM creates a registry for each transition of control from Java to a native method to implement reference. JNI provides a set of accessor functions on references.
- Primitive Arrays  
JNI provide a set of functions copy primitive array elements between a segment of a Java array and a native memory buffer.
- Fields and Methods  
JNI identify methods and fields by their symbolic names and type signatures. For example: to call method `f` in class `cls`, the native code first obtain a method ID:  
`jmethodID method_id = env->GetMethodID(cls, "f", "(Ljava/lang/String;)D");`  
The native code can then use the method ID repeatedly:  
`jdouble result = env->CallDoubleMethod(obj, method_id, 10, str);`

### For more JNI information

Please refer: <https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html>

## APK decode and native library identify

### What is inside an APK file?

An Apk file are a zip file containing resources and assembled java code. If we use apktool decode APK file, we usually get following part:

- AndroidManifest.xml  
Every app project must have one at the root of the project source set describing important information of the app. The information includes: app package name, components of the app, the permission the app need, features app requires.
- smali  
Contains Kotlin/Java code that disassembles from APK's DEX files. Each .smali file corresponds to a Kotlin/Java class.
- res  
Resources are the additional files and static content the app code uses, such as layout, bitmaps.
- lib  
If app includes native code, this directory contains APK's native libraries (.so files).

## How to identify native library exist and in use?

- Native libraries exist  
If we decode/unzip APK file, we can find .so file.
- Native libraries in use  
According to JNI working flow, Native methods are loaded with System.loadLibrary method. So if native libraries is in use, System.loadLibrary method must be called either from code or JVM.

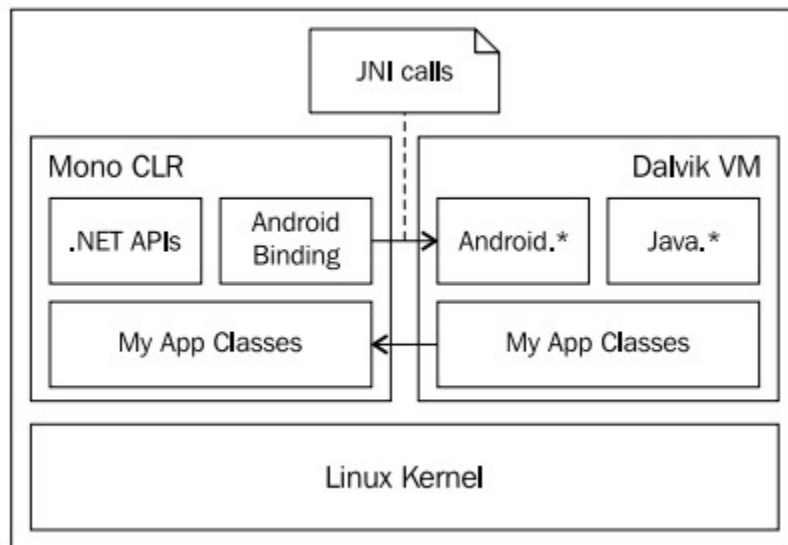
## Native library identify implementation

Please see native\_lib\_identify.py

## Current popular cross-platform framework for Android

### Xamarin.Android Architecture and Mono

Xamarin.Android is a product application provides software developing environment leverage Mono allow developers develop applications for Android, IOS and OS X using C# and .Net framework. Mono is one of cross-platform implementation support C# language. Mono is open source, includes C# compiler, and a Common Language Runtime (CLR) that is binary compatible with Microsoft .NET. These Mono execution environment and Android Runtime (ART) virtual machine are running side-by-side. See below figure:



Both ART and Mono Runtime run on top of the Linux kernel and expose APIs to the user code. Developers access the underlying system using these APIs with their code. For example, within Mono Runtime, developers can use System.IO, System.Net in .NET class libraries to access underlying Linux system facilities. Within ART, most of system facilities such as Graphics, Audio, are not directly available to native applications. Developers using Java.\* or Android.\* namespaces to access the system facilities.

## Managed Callable Warppers

Managed Callable Warppers (MCW) are a JNI bridge used any time the Android runtime needs to invoke managed code. MCW represents how Java Interfaces (The entire `Android.*` and related namespace) can be implemented and how virtual methods are overridden. MCW is responsible converting between managed classes, Android types and ART methods via JNI. Each MCW holds a Java global reference accessible through `Android.Runtime.IJavaObject.Handle` property. This reference provides a mapping between Java instance and managed instance.

## Java Activation

Whenever a Java object is exposed to managed code and a MCW needs to be constructed to manage the JNI handle, usually the constructor is invoked automatically. There are scenarios constructor must be manually provided but we do not expand here.

## Application Startup

When an activity, service, etc is launched, android will first check to see if there is already a process running to host the activity/service/etc. If no such process, then a new process will be created reading `AndroidManifest.xml` and the type specified in the `/manifest/application/@android:name` attribute is loaded and instantiated. Once instantiated, `ContentProvider.AttachInfo(Context, ProviderInfo)` method will be invoked. `Xamarin.Android` hooks into this by adding a `mono.MonoRuntimeProvider` `ContentProvider` to `AndroidManifest.xml` during the build process. The `mono.MonoRuntimeProvider.attachInfo()` method is responsible for loading the Mono runtime into the process. Any attempts to use Mono prior to this point will fail.

## For more Xamarin.Android Architecture information

For more detailed information, please see <https://docs.microsoft.com/en-us/xamarin/android/internals/architecture>

## How to identify Mono App

- Native library name  
This approach is based on assumption the universal libraries for mono will remain still. This approach is vulnerable if library name get updated in future. Until June 2020, we find below typical mono library names:
  - `libmono-btls-shared.so`
  - `libmono-native.so`
  - `libmono-profiler-log.so`
  - `libmonodroid.so`
  - `libxamarin-app.so`
  - `libxamarin-debug-app-helper.so`
- `AndroidManifest.xml` provider check  
This is a recommended and easy way to classify Mono application. For example, in our test application, we see below provider statement:

```
<provider android:authorities="com.companyname.test_app.mono.MonoRuntimeProvider.__mono_init__"
android:exported="false" android:initOrder="1999999999" android:name="mono.MonoRuntimeProvider"/>
```

- `mono.MonoRuntimeProvider.attachInfo()` method calling scan

While we can check the runtime provider provides the method, If we want to ensure the app process loads the runtime, we can scan this method calling in the code.

- Class name check

This approach is based on assumption the universal mono package name will remain still. This approach is vulnerable if class name get updated in future. Until June 2020, we find mono class name includes:

- `mono.MonoPackageManager_Resources`
- `mono.MonoPackageManager`
- `mono.MonoRuntimeProvider`
- `mono.android.*`
- `mono.java.*`
- `mono.javax.*`

## React and React Native

### React

React is a JavaScript framework developed by Facebook designed for solving the problem with complex user interfaces which had data that changed over time. React serves as the “view” framework part of the development paradigm model-view-controller (MVC).

### Virtual DOM

Once a web page is loaded into a web browser, a Document Object Model (DOM) is created containing that web page. A DOM is of tree structure represent the structure of the current web page and its HTML elements. When action is performed on the web page, usually a new page is created or doing corresponding JavaScript code action. When the DOM is manipulated by JavaScript, this is done using a single page application (SPA). However, DOM manipulation is expensive when mutations are slow. People dive into a solution called two-way data binding. This binding help receive the synchronisation between the application UI state and the data model's state. This solution is achieved by using key-value observing used by Knockout.js. In IOS, this is achieved by checking which the framework Angular by Google. This would require a linking function to look at what data has changed and imperatively make change to the DOM to keep it updated.

React uses a concept called the Virtual DOM instead of two-way data binding. The Virtual DOM selectively renders subtrees of the node in the DOM based on the current state, resulting in a minimal amount of manipulation to keep the page updated. The Virtual DOM is a representation of the actual DOM and is an abstraction which grants the ability to treat the the JavaScript and the DOM as if they were reactive. React will store the state of the application internally and only perform the DOM manipulation when the state has changed. When the state of data model has changes, the virtual DOM and React will re-render the user interface to identify the difference and what should be manipulated. Then the differences are updated into the real DOM.

## React Native

React Native is a JavaScript framework implementation on IOS and android based on React. React Native is not open-sourced yet and runs in an embedded instance of JavaScriptCore (IOS) or V8 (android) inside the application and render to higher-level platform-specific components.

### The core of React Native

React Native is able to render the React Native component to real native Views for Android or UI Views for IOS. This is achieved by the abstraction layer known as the "bridge" which enables React Native to invoke the rendering APIs in Java for android or Objective-C for IOS.

### How to identify React App

- Native library name

This approach is based on assumption the universal libraries for React will remain still. This approach is vulnerable if library name get updated in future. Until June 2020, we find below typical react library names:

- libreactnativeblob.so
- libreactnativejni.so

- Class name check

This approach is based on assumption the universal React package name will remain still. This approach is vulnerable if class name get updated in future. Until June 2020, we find typical react class name includes:

- com.facebook.react.ReactActivity\*
- com.facebook.react.ReactApplication
- com.facebook.react.ReactDelegate
- com.facebook.react.ReactFragment\*
- com.facebook.react.ReactInstanceManager\*
- com.facebook.react.ReactPackage\*
- com.facebook.react.\*

## Flutter

### About Flutter

Flutter is an open-sourced app SDK developed by Google for building cross-platform application support Dart language. Application made in Flutter are build from a single codebase, are compiled to native ARM code.

### About DART

Google created Dart and uses it internally with some of its big projects such as Google AdWords. Dart is used to build mobile, web and server applications fast, portable and reactive. There are some beneficial using Dart:

- Just in time  
Dart is just in time (JIT) compiled. When developer debugging application, the JIT compilation also happens.
- Fast  
Flutter rendering runs at 60 frames per second(fps) and 120fps for capable devices of 120HZ. The more fps, the smoother the app.

### **How to identify Flutter App**

- Native library name  
This approach is based on assumption the universal libraries for Flutter will remain still. This approach is vulnerable if library name get updated in future. Until June 2020, we find below typical react library names:
  - libflutter.so
- Class name check  
This approach is based on assumption the universal Flutter package name will remain still. This approach is vulnerable if class name get updated in future. Until June 2020, we find typical react class name includes:
  - io.flutter.app
  - io.flutter.embedding
  - io.flutter.plugin\*
  - io.flutter.view
  - io.flutter.util
  - io.flutter.\*

## **PhoneGap and Apache Cordova**

### **PhoneGap**

PhoneGap is an application cross-platform framework that enables developers build native applications using HTML and JavaScript.

### **PhoneGap and Apache Cordova**

PhoneGap was originally developed by Nitobi, a company acquired by Adobe. After this company was acquired, Nitobi donated the PhoneDap code base to Apache Software Foundation (ASF) under the project name Cordova.

PhoneGap is a free and open licensed distributin of Apache Cordova.

### **How Cordova access to Native APIs**

When people begin to require cross-platform solution. The most simple cross-platform solution for mobile devices was using HTML as a webapp. However, for mobile developers, many mobile applications need to do more than HTML and web browser can support. For example, building a web application that interactied with the mobile camera. To get around this, Cordova implements a suite of APIs that extend native device capabilities to a web application. Cordova provide a suit of Java APIs that a developer



can leverage to allow a web application running with the Cordova container to access device capabilities outside of the web context. Essentially these APIs are implemented in two parts: a JavaScript library that expose to the native capabilities to the web application and the corresponding native code running in the container that implements the native part of the API. JavaScript access to native APIs made available through the JavaScript to native bridge into the Cordova container.

When a developer implements a feature in an application that uses one of the Cordova APIs, the application calls the API using JavaScript, and then a special layer within the Cordova translates the Cordova API call into the appropriate native API for the particular features.

## How to identify Cordova App

- Class name check

This approach is based on assumption the universal Cordova package name will remain still. This approach is vulnerable if class name get updated in future. Until June 2020, we find typical react class name includes:

- org.apache.cordova.engine
- org.apache.cordova.whitelist
- org.apache.cordova.\*

- JavaScript lib support check

From Apktool decoding, we find one extra directory called "assets" inside Cordova app. This directory include JavaScript source code, HTML file and native API translation providers.

The name of the directory include:

- \assets\www\cordova-js-src\
- \assets\www\css\
- \assets\www\img\
- \assets\www\js\
- \assets\www\cordova\_plugin.js
- \assets\www\cordova.js
- \assets\www\index.html

## Capacitor

### About Capacitor

Capacitor is a cross-platform app runtime that makes it easy to build web apps that run natively on iOS, Android, and the web. Capacitor provides a consistent, web-focused set of APIs that enable an app to stay as close to web-standards as possible, while accessing rich native device features on platforms that support them. Adding native functionality is easy with a simple Plugin API for Swift on iOS, Java on Android, and JavaScript for the web.

Capacitor has compatible support for many existing Cordova libraries.

## How to identify Capacitor App

- Class name check

This approach is based on assumption the universal Capacitor package name will remain still. This approach is vulnerable if class name get updated in future. Until June 2020, we find typical react class name includes:

- `com.getcapacitor.android.android`
- `com.getcapacitor.android.cordova`
- `com.getcapacitor.android.plugin`
- `com.getcapacitor.android.ui`
- `com.getcapacitor.android.util`
- `com.getcapacitor.android.*`
- `capacitor.android.plugins.*`

- JavaScript lib support check

From Apktool decoding, we find one extra directory called "assets" inside Capacitor app. This directory include JavaScript source code, HTML file and native API translation providers.

The name of the directory include:

- `\assets\public\`
- `\assets\capacitor.config.json`