adding `framework.jar` in android studio

mainuser android 2017-08-102017-10-27 4 Minutes

This post is part of another post (https://kwagjj.wordpress.com/2017/10/27/building-custom-android-sdk-from-aosp-and-adding-it-to-android-studio/) dealing with how to let Android Studio work with a custom android framework where modifications have been made in the AOSP tree.

What is `framework.jar`

I do not have the full grasp of this `jar` file. BTW, a `jar` file is a Java archive format which may contain whatever resources. In the case for `framework.jar` , it would be a huge set of classes.

After an AOSP full image build, a `framework.jar` file can be found in the `/system/framework` directory in the output files. However, in my case the `framework.jar` was an empty file that contained nothing. This can be checked with the `jar tf [jar file]` command like the example below:

```
$ jar tf framework.jar
META-INF/
META-INF/MANIFEST.MF
```

I am not sure if other AOSP builds might create a `framework.jar` file that may actually contain something. If so, then I believe you do not have to read the rest of this section.

Clearly, the `/system/framework.jar` is not the `framework.jar` file that I am looking for. After many googling, one result pointed me to checkout the intermediary files that are saved in the `/out` directory. I found a `jar` file that looked promising and it was `/out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/classes.jar` .

listing the contents of this `jar` file proved that it indeed contained the class that had been custom modified.

```
$ jar tf classes.jar
...
android/speech/tts/TextToSpeech$Engine.class
...
```

One questions that may rise is why is it not named `framework.jar` but `classes.jar` ? According to the google result, the `classes.jar` file is the file which will be renamed/transformed to `framework.jar` . There may be some minor changes applied to the `classes.jar` but it is not critical in which the android studio will not be able to understand.

Adding it to Android Studio

An android studio project has been created with the following tree structure.

```
$ tree . -d -L 2
.
├── app
│   ├── build
│   ├── libs
│   └── src
├── build
│   ├── android-profile
│   ├── generated
│   └── intermediates
├── framework
├── gradle
│   └── wrapper
└── META-INF
```

Adding it to `app/libs`

The `classes.jar` file has been copied to `app/libs` directory and renamed to `framework.jar` .

Adding it to dependencies

In Android Studio after opening the project, go to `File - Project Structure` . In the pop-up window, go to `Dependencies` tab.

Screenshot from 2017-08-10 10:30:35

Click the `+` icon on the right to add a new dependency. Click `Jar Dependency` . Navigate and select `app/libs/framework.jar` file.

Screenshot from 2017-08-10 10:33:22.png

In the newly added `libs/framework.jar` entry, change the Scope to `provided` .

Screenshot from 2017-08-10 10:34:41.png

Click `OK` to save the changes.

The procedure above will allow the project to recognize the custom `framework.jar` . Then the question is which one will the project reference? Will it reference our custom `framework.jar` or will it reference to the official default SDK? Unfortunately, the project will still refer to the default SDK. The next step will take care of this problem.

Adding `app.iml` modifying gradle script

The main problem we now face is telling the project to reference `framework.jar` first instead of the default SDK. The hierarchy of referencing is written in `app.iml` file located in `/app` directory. Take a look and one can recognize a group of `orderEntry` items at the bottom of this file. The order of these `orderEntry` s defines which one to refer first over the other. We can see that the `framework.jar` entry is placed below the `Android SDK` entry.

```
<orderEntry type="jdk" jdkName="Android API 25 Platform" jdkType="Android SDK"/
...
<orderEntry type="library" exported="" name="framework" level="project"/>
```

We need to make sure that the `framework` entry is placed above the `Android SDK` entry. A gradle script can be added to the module's `build.gradle` to make sure of this. In the module's `build.gradle` , please add the following snippet:

```
preBuild {

doLast {
 def imlFile = file( project.name + ".iml")
 println 'Change ' + project.name + '.iml order'
 try {
   def parsedXml = (new XmlParser()).parse(imlFile)
   def jdkNode = parsedXml.component[1].orderEntry.find { it.'@type' == 'jdk' }
   parsedXml.component[1].remove(jdkNode)
   def sdkString = "Android API " + android.compileSdkVersion.substring("androi
   println 'what' + sdkString
   new Node(parsedXml.component[1], 'orderEntry', ['type': 'jdk', 'jdkName': sd
   groovy.xml.XmlUtil.serialize(parsedXml, new FileOutputStream(imlFile))
 } catch (FileNotFoundException e) {
   // nop, iml not found
   println "no iml found"
 }
 }
}
```

Based on observation, a gradle sync or typical build will call `preBuild` directive. This will execute the snipped above (probably at the last moment due to the `doLast` command). The code will read the `app.iml` , save the orderEntry that contains the Android SDK, removes it from the iml file, add a new orderEntry at the last which is identical to the temporarily saved Android SDK orderEntry to the iml file. Eventually, the Android SDK orderEntry will be placed at the very last, thereby guaranteeing the `framework.jar` to be referenced first than the Android SDK.

Force to refer `framework.jar` during Java compile

A few more lines need to be added to the `build.gradle` file. The following code(not the whole but the part after "please add the following") will be read during gradle's configuration phase (https://docs.gradle.org/current/userguide/build_lifecycle.html). What it means is, for the projects including subprojects where this `build.gradle` resides, all the tasks that are type of `JavaCompile` will be executed with an additional compiler argument `-Xbootclasspath/p:app/libs/framework.jar` . The reason why this code is necessary is self explanatory.

```
allprojects {
    repositories {
        jcenter()
    }

    // please add the following!
    gradle.projectsEvaluated {
        tasks.withType(JavaCompile) {
            options.compilerArgs.add('-Xbootclasspath/p:app/libs/framework.jar'
        }
    }
}
```

Result

After gradle sync, the “cannot resolve method” error has vanished.

 Screenshot from 2017-08-10 11:12:10.png

And although I do not have a screenshot, the build of `.apk` file is successful too.

Tagged:
android studio,
framework,
framework.jar

Published by mainuser

View all posts by mainuser

10 thoughts on “adding `framework.jar` in android studio”

Pingback: building custom android SDK from AOSP and adding it to Android Studio – JUST DO IT

Anonymous says:
2018-03-19 at 11:51 am

there're some mistakes …
options.compilerArgs.add('-Xbootclasspath/p:app/libs/framework.jar ')

↳ Reply

mainuser says:
2018-03-19 at 11:54 am
which are in this line is incorrect?

↳ Reply

mainuser says:
2018-03-19 at 11:55 am
I meant "which part in that line is incorrect"

Gaurav says:
2018-09-11 at 11:22 am
After doing this, not able to build APK

↳ Reply

jayanthp50 says:
2019-09-26 at 9:49 am
Hi, currently i'm also unable to build APK after following the above mentioned procedure. Were you able to resolve this issue?

↳ Reply

Anonymous says:
2019-05-31 at 6:32 am
java.lang.ArrayIndexOutOfBoundsException: 65535

↳ Reply

Soumya Ranjan says:
2019-08-27 at 10:57 am
make sure that you have given dependency scope compileOnly/provided not implementation.

↳ Reply

Didier Guillemot says:
2020-02-03 at 3:11 pm
After applying this tutorial, I was also not able to build an APK due to the following error :
"reference to findViewById is ambiguous"
I solved this by removing Activity.class from the archive

↳ Reply

hanhphuclahappy says:
2023-02-20 at 8:57 am
My project don't have .iml files.
How can I solve it ?

This is gradle logs after I build success

> Task :app:preBuild
Change app.iml order
no iml found

↳ Reply