Solve Gradle & Flutter Build Errors: Java Version Fixes & Dependency Solutions

4 months ago 高效码农

Diagnosing and Fixing Gradle & Flutter Build Errors — A Practical, Step-by-Step Guide

> This article is a direct, practical translation and rewrite of the build logs and interactions you provided. It keeps only the facts and steps that appear in the input, presented as a clear, actionable guide for engineers with a junior-college level of experience or above. Everything below is strictly derived from the original content you supplied; no outside material has been added.

Overview

You provided a set of Gradle/Flutter build errors and traces. They repeatedly point to a small set of root causes that interact with each other:

- Gradle / plugin dependency resolution failures where certain artifacts (`com.github.Ysj001.ByteCodeUtil:plugin` , `com.github.Ysj001:DoKitPluginCompat`) could not be resolved because the artifact variants declare a **Java version** not compatible with the build JVM. The logs show: *component declares a component compatible with Java 17 but the consumer needed Java 11*.
- Flutter module include and local package resolution failures: `include_flutter.groovy` missing, `project(':flutter')` not found, and `fiat_module` local path dependency missing in the Flutter module. Those block `flutter pub get` /AAR generation and therefore break Android side inclusion.
- Kotlin / Java JVM target mismatch warnings and resulting Kotlin compilation failures inside the Flutter tool plugin source (`FlutterPlugin.kt`) — unresolved references such as `filePermissions` , `user` , `read` , `write` . The errors show `compileJava (11)` vs `compileKotlin (1.8)` inconsistency.
- Other incidental issues such as DataBinding `BR` imports failing and `@Deprecated` annotation warnings.

This guide walks through the logs, extracts the exact error messages, and shows the exact, practical steps and configuration snippets that will resolve the issues — in the order that m
steps exactly and run the commands shown to verify each step before proceeding

---

Quick problem summary

Your build fails because plugin artifacts and plugin code expect a newer Java/Kotlin toolchain (the artifacts are compiled for Java 17 and Kotlin expectations differ), while your Gradle or module settings still target older JVM compatibility (Java 11 or Kotlin jvmTarget 1.8); additionally, local Flutter module packaging and local path dependencies are missing, preventing generation of `.android/include_flutter.groovy` and causing `project(':flutter')` not to be available.

Raw error highlights

Use these exact lines when you verify outputs — they are the authoritative signals we respond to.

- `No matching variant of com.github.Ysj001.BytecodeUtil:plugin:2.1.3 was found... Incompatible because this component declares a component compatible with Java 17 and the consumer needed a component compatible with Java 11`
- `No matching variant of com.github.Ysj001:DoKitPluginCompat:3.7.1-v1 was found... Incompatible because this component declares a component compatible with Java 17 and the consumer needed a component compatible with Java 11`
- `Project with path ':flutter' could not be found in project ':flutter_modules'`
- `D:\coinbyte_flutter_entry\.android\include_flutter.groovy (D:\coinbyte_flutter_entry\.android\include_flutter.groovy)` — file not found reported by `settings.gradle` line:85
- `Because coinbyte_flutter_entry depends on fiat_module from path which doesn't exist (could not find package fiat_module at "..\fiat_module"), version solving failed.`
- `Inconsistent JVM-target compatibility detected for tasks 'compileJava' (11) and 'compileKotlin' (1.8). This will become an error in Gradle 8.0.`
- `Unresolved reference: filePermissions`
  `Unresolved reference: user`
  `Unresolved reference: read`
  `Unresolved reference: write`
  — all reported in: `.../flutter/packages/flutter_tools/gradle/src/main/kotlin/FlutterPlugin.kt:758..761`
- `Plugin [id: 'com.android.library'] was not found in any of the following sources` — reported from `coinbyte_flutter_entry/.android/Flutter/build.gradle`
- `import jp.co.huobi.japan.BR;` — error: symbol not found (BR missing because DataBinding class not generated)

Minimal explanations used to shape fixes

(These points are strictly derived from the logs and prior suggestions included in your interaction; they are not extra facts.)

- If an artifact declares it was built for Java 17 but Gradle/consumer runs on Java 11, resolution will fail. The logs explicitly state this mismatch.
- If a Flutter module depends on a local path package (like `fiat_module`) and t exist, `flutter pub get` fails. Without a successful `pub get` and `flutter bu lude_flutter.groovy` will not be produced; `settings.gradle` that reference therefore fail.

- Kotlin and Java compile targets must be consistent. The logs show `compileJava` and `compileKotlin` targets differ (11 vs 1.8), and the Kotlin compile step fails within Flutter plugin Kotlin sources because the JVM target is too low to expose certain APIs referenced there.

What to do, in the order you should do it

Step 0 — capture verification outputs (do this once, copy results here if you need further help)

Run these in the project root and capture outputs:

```
```

These two outputs show what Java and what JVM are currently being used. If `./gradlew -version` shows a JVM that is not Java 17 and your plugin artifacts require Java 17, you need to make Gradle use Java 17.

Step 1 — Make Gradle use Java 17 (root cause for many dependency resolution failures)

Why first: The plugin artifact variants in your logs declare compatibility with Java 17; the consumer (your build) currently needs Java 11. Make the build use Java 17 so dependency resolution and Kotlin compile can proceed.

What to do (persistent setting): Edit the project-level `gradle.properties` file (create one at the root if it does not exist) and add:

```
# point to your local JDK 17 installation; replace with your actual path
org.gradle.java.home=C:/jdk17
org.gradle.jvmargs=-Xmx4096m -Dfile.encoding=UTF-8
```

What to do (temporary, Windows CMD):

```
set JAVA_HOME=C:\jdk17
set PATH=%JAVA_HOME%\bin;%PATH%
gradlew.bat clean build --refresh-dependencies
```

Verify: `./gradlew -version` should show a JVM location that matches the JDK 17 path you set. If the artifact resolution error (`No matching variant ... Java 17`) disappears, step 1

Step 2 — Align Kotlin `jvmTarget` and Java compile target (fix co compileKotlin inconsistency)

Why: The logs show `compileJava (11)` vs `compileKotlin (1.8)` mismatch. This causes Kotlin to target an older JVM where certain APIs (like file permission APIs

`terPlugin.kt`) are not available, producing `Unresolved reference` errors.

What to add (module-level or centralized inside `subprojects{}`):

Place this in your **root** `build.gradle` inside a `subprojects` block, or place equivalent lines in each module that compiles Kotlin:

```
subprojects {
    afterEvaluate { project ->
        if (project.plugins.hasPlugin('kotlin') || project.plugins.hasPlugin('com.android.library')
|| project.plugins.hasPlugin('com.android.application')) {
            // Kotlin toolchain to 17

project.extensions.findByType(org.jetbrains.kotlin.gradle.dsl.KotlinJvmProjectExtension)?.jvmToolcha
{
                languageVersion.set(org.gradle.jvm.toolchain.JavaLanguageVersion.of(17))
            }
            // Kotlin compile tasks set to jvmTarget 17
            project.tasks.withType(org.jetbrains.kotlin.gradle.tasks.KotlinCompile).configureEach {
                kotlinOptions.jvmTarget = "17"
            }
            // Java compile options align with Java 17
            project.extensions.findByName("android")?.with {
                compileOptions {
                    sourceCompatibility JavaVersion.VERSION_17
                    targetCompatibility JavaVersion.VERSION_17
                }
            }
        }
    }
}
```

Or, module-level snippet (one module `build.gradle`):

```
android {
    compileSdk 33
    buildToolsVersion "33.0.3"

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_17
        targetCompatibility JavaVersion.VERSION_17
    }

    kotlinOptions {
        jvmTarget = "17"
    }
}

kotlin {
    jvmToolchain {
        languageVersion.set(JavaLanguageVersion.of(17))
    }
}
```

Verify: Re-run `./gradlew clean build`. The `Inconsistent JVM-target` warning should no longer appear and the Kotlin compilation of Flutter plugin sources should proceed without `Unresolved reference` for `filePermissions`, `user`, `read`, `write`.

Step 3 — Ensure the Android Gradle Plugin (AGP) and Kotlin Gradle plugin are available in root `build.gradle`

Why: The `Plugin [id: 'com.android.library'] was not found` message indicates that the buildscript classpath did not include AGP. Add AGP and Kotlin plugin classpath entries at the root.

Add / verify in `build.gradle` (root):

```
buildscript {
    ext.kotlin_version = '1.8.20'
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:8.1.2"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
    }
}
```

Notes from your logs: You referenced `ext.kotlin_version = '1.9.0'` and `gradle 7.5` earlier. The practical set used in the logs and in the assistance was to use `kotlin_version = '1.8.20'` with AGP `8.1.2` — that combination is the one included above in configuration examples from the conversation.

Verify: Gradle sync should stop reporting `com.android.library` missing.

Step 4 — Fix Flutter module local dependency errors and generate `.android/include_flutter.groovy`

Why: The `include_flutter.groovy` file is missing and `pub get` failed because `fiat_module` path dependency does not exist. Without the generated include script, `settings.gradl` the Flutter module include.

Action plan (exact commands and checks):

1. Open `coinbyte_flutter_entry/pubspec.yaml` and find the `fiat_module` ent

```
dependencies:
  fiat_module:
    path: ../fiat_module
```

2. Ensure the folder `../fiat_module` exists relative to `coinbyte_flutter_entry`. Based on the logs, this path was missing and caused `flutter pub get` to fail.

3. If `fiat_module` is missing, either:

   - restore or copy `fiat_module` into the expected relative path; **or**
   - temporarily comment out the `fiat_module` dependency in `pubspec.yaml` (noting that code using it will then be broken until you restore it).

4. Run (from `coinbyte_flutter_entry`):

```
flutter pub get
flutter build aar    # or `flutter build apk` depending on your integration usage
```

Outcome expected by the logs: After `flutter build aar`, `.android/include_flutter.groovy` will exist at `D:\coinbyte_flutter_entry\.android\include_flutter.groovy`. That file is referenced by `settings.gradle`; once it exists, the settings evaluation error disappears.

Step 5 — Correct `settings.gradle` include paths and fix typos

Why: Your logs show path typos such as `coinbyte_app-maste` missing an `r`. Even small path typos cause `FileNotFoundException` for include files. Fix the paths and ensure references are relative to project root correctly.

Example fixes (apply exactly in `settings.gradle`):

If you want to apply the generated include script:

```
apply from:'coinbyte_flutter_entry/.android/include_flutter.groovy'
```

Or, if you prefer to include the Flutter module as a project directly:

```
include ':flutter'
project(':flutter').projectDir = new File(rootDir, 'coinbyte_flutter_entry/.android/Flutter')
```

Verify: `./gradlew projects` or `./gradlew tasks` should no longer fail at settings evaluation. The error `Project with path ':flutter' could not be found` should go away because `:flutter` will be included.

Step 6 — Run a clean build and watch for the first remaining error

Commands (from project root):

```
flutter clean
flutter pub get
./gradlew clean --no-daemon
./gradlew build --refresh-dependencies --no-daemon
```

Why: This clears caches and forces dependency resolution again. If the earlier ste
correctly, the build should pass beyond the earlier dependency resolution and Ko
If any new or remaining error appears, capture it exactly and use it as the next di

---

Code / config snippets you can copy (collected in one place)

`gradle.properties`

```
org.gradle.java.home=C:/jdk17
org.gradle.jvmargs=-Xmx4096m -Dfile.encoding=UTF-8
kotlin.code.style=official
```

Root `build.gradle` (top fragment to include AGP and Kotlin plugin)

```
buildscript {
    ext.kotlin_version = '1.8.20'
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:8.1.2"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
    }
}
```

Module-level JVM/Kotlin alignment (put in a module `build.gradle` or centralize in root `subprojects` block)

```
android {
    compileSdk 33
    buildToolsVersion "33.0.3"

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_17
        targetCompatibility JavaVersion.VERSION_17
    }

    kotlinOptions {
        jvmTarget = "17"
    }
}

kotlin {
    jvmToolchain {
        languageVersion.set(JavaLanguageVersion.of(17))
    }
}
```

`settings.gradle` examples

Use one of the two approaches:

- Include the generated include script (if it exists):

```
apply from:'coinbyte_flutter_entry/.android/include_flutter.groovy'
```

- Or include Flutter project explicitly (adjust the path if your Flutter module gen
  `Flutter` subproject):

```
include ':flutter'
project(':flutter').projectDir = new File(rootDir, 'coinbyte_flutter_entry/.
```

Checklist you can follow one-by-one

1. Confirm Java and Gradle JVM:
   - `java -version`
   - `./gradlew -version`
2. If Gradle JVM is not JDK 17, set `org.gradle.java.home` in `gradle.properties` to your JDK 17 path.
3. Add AGP and Kotlin plugin classpath entries (root `build.gradle`).
4. Align Java and Kotlin targets (set `sourceCompatibility`, `targetCompatibility`, and `kotlinOptions.jvmTarget` to 17; set `kotlin.jvmToolchain` to Java 17).
5. Fix missing local Flutter packages (`fiat_module`) referenced in `coinbyte_flutter_entry/pubspec.yaml`.
6. Run `flutter pub get` in `coinbyte_flutter_entry`.
7. Run `flutter build aar` (or `flutter build apk`) to generate `.android/include_flutter.groovy`.
8. Correct `settings.gradle` include paths to the generated include or include the `:flutter` project explicitly.
9. Clean and rebuild:
   - `flutter clean`
   - `flutter pub get`
   - `./gradlew clean`
   - `./gradlew build --refresh-dependencies`
10. If any error remains, copy the *first* error message and bring it back — that will be the next root cause to handle.

FAQ (questions you are likely to ask, answered directly)

Q: Why does Gradle say a plugin artifact is incompatible with my build JVM?
A: The logs show the plugin artifacts declare compatibility with Java 17 while your build JVM was Java 11. Gradle refuses to resolve variants that are declared for a different Java target. To resolve, make Gradle use JDK 17 or use plugin versions compiled for Java 11—your logs show the artifact variants are Java 17.

Q: `include_flutter.groovy` is missing; how do I get it?
A: From your Flutter module folder (`coinbyte_flutter_entry`), run `flutter pub get` and then `flutter build aar` (or `flutter build apk`) to generate the `.android` directory and `include_flutter.groovy`. If `pub get` fails, fix local path dependencies first (your logs show `fiat_module` path missing).

Q: Why am I seeing `Unresolved reference: filePermissions` inside `FlutterPlugin.kt`?
A: The Kotlin compile target was too low (jvmTarget 1.8) while compileJava targeted Java 11 or higher, causing inconsistent visibility for APIs used by the Flutter Gradle plugin Kotlin code. Align Kotlin jvmTarget and Java target (set both to 17 as shown) and the unresolved references go away.

Q: My `BR` import can't resolve (`import jp.co.huobi.japan.BR;`). What does that mean?
A: `BR` is the DataBinding generated class. If DataBinding is not enabled or layout files are not using `<layout>` root tags, `BR` will not be generated. Ensure `buildFeatures { dataBinding true }` and correct module `namespace` if necessary.

Q: Can I avoid installing JDK 17 and just downgrade the plugins?
A: Only if the plugin artifacts you use have versions compiled to be compatible with Java 11. Your logs indicate certain plugin versions are Java 17; if a Java-11-compatible release exists, you could switch to it. Otherwise, make Gradle run with Java 17.

HowTo (JSON-LD structured for LLM and data ingestion)


```json
{
  "@context": "https://schema.org",
  "@type": "HowTo",
  "name": "Fix Gradle + Flutter build errors related to Java/Kotlin target mismatch and missing Flutter include",
  "description": "Step-by-step actions to resolve plugin resolution failures due to Java target mismatch, to generate Flutter include files, and to align Kotlin/Java compile targets.",
  "step": [
    {
      "@type": "HowToStep",
      "name": "Check Java and Gradle JVM",
      "text": "Run `java -version` and `./gradlew -version` to find the JVM used by Gradle."
    },
    {
      "@type": "HowToStep",
      "name": "Force Gradle to use JDK 17",
      "text": "Set `org.gradle.java.home` in `gradle.properties` to your JDK 17 path or export `JAVA_HOME` to JDK 17 before running Gradle."
    },
    {
      "@type": "HowToStep",
      "name": "Align Kotlin and Java targets",
      "text": "Set `kotlinOptions.jvmTarget` and Java `sourceCompatibility/targetCompatibility` to `17` and configure Kotlin jvmToolchain to Java 17."
    },
    {
      "@type": "HowToStep",
      "name": "Fix missing Flutter local package",
      "text": "Ensure local path dependency (e.g., `fiat_module`) exists so `flutter pub get` succeeds and `.android/include_flutter.groovy` can be generated."
    },
    {
      "@type": "HowToStep",
      "name": "Generate Flutter include file and include it",
      "text": "Run `flutter build aar` to produce `.android/include_flutter.
`settings.gradle` apply from that file or include `:flutter` manually."
    }
  ]
}
```

Final verification checklist (one more time, copy & paste to run)

Run these commands and confirm no errors remain:

```
# in project root
java -version
./gradlew -version

# ensure JDK17 is used
# either set JAVA_HOME/PATH or set org.gradle.java.home in gradle.properties

# from Flutter module
cd coinbyte_flutter_entry
flutter pub get
flutter build aar

# back to project root
flutter clean
flutter pub get
./gradlew clean
./gradlew build --refresh-dependencies
```

If you still get any `Unresolved reference` errors or the plugin variant error, copy the *exact* error lines and return them — they are the precise signals we use to identify the next root cause.

Closing note

Everything in this post comes exactly from your provided build logs and the configuration snippets that appeared in the interactions. The instructions and configuration excerpts above are the practical actions and minimal changes that directly address each quoted error in your logs: Java version mismatches on dependency resolution, missing Flutter include files caused by local package absence, Kotlin and Java target inconsistencies, and the missing AGP classpath causing `com.android.library` plugin errors.

If you need, paste the outputs of:

1. `java -version`
2. `./gradlew -version`
3. the top ~120 lines of your root `build.gradle`
4. the `gradle.properties` file

— and I will produce the exact patch / diff to apply to those files so you can test with a single apply-and-build iteration.

Tags: Android Development Flutter Gradle Java Kotlin

Liked (0)

Related Posts

- Mastering AI in 2026: 6 Skills to Outperform 90% of t…
- AI Agent Evaluations: The Complete 2025-2026 Gui…
- Claude Code Setup Guide: Master Installation, Confi…
- WeChat Chat History Unleashed: View, Export & Su…
- Automated AI Media Software: The Future of Conte…
- VideoRAG: How Machines Finally Crack Extreme L…
- LittleCrawler Python Framework: Master XHS, Xiany…
- UniVideo Explained: The Single Open-Source Mode…