社区说 Community Talks

扩展 Android 构建流程

基于新版 Variant/Artifact APIs

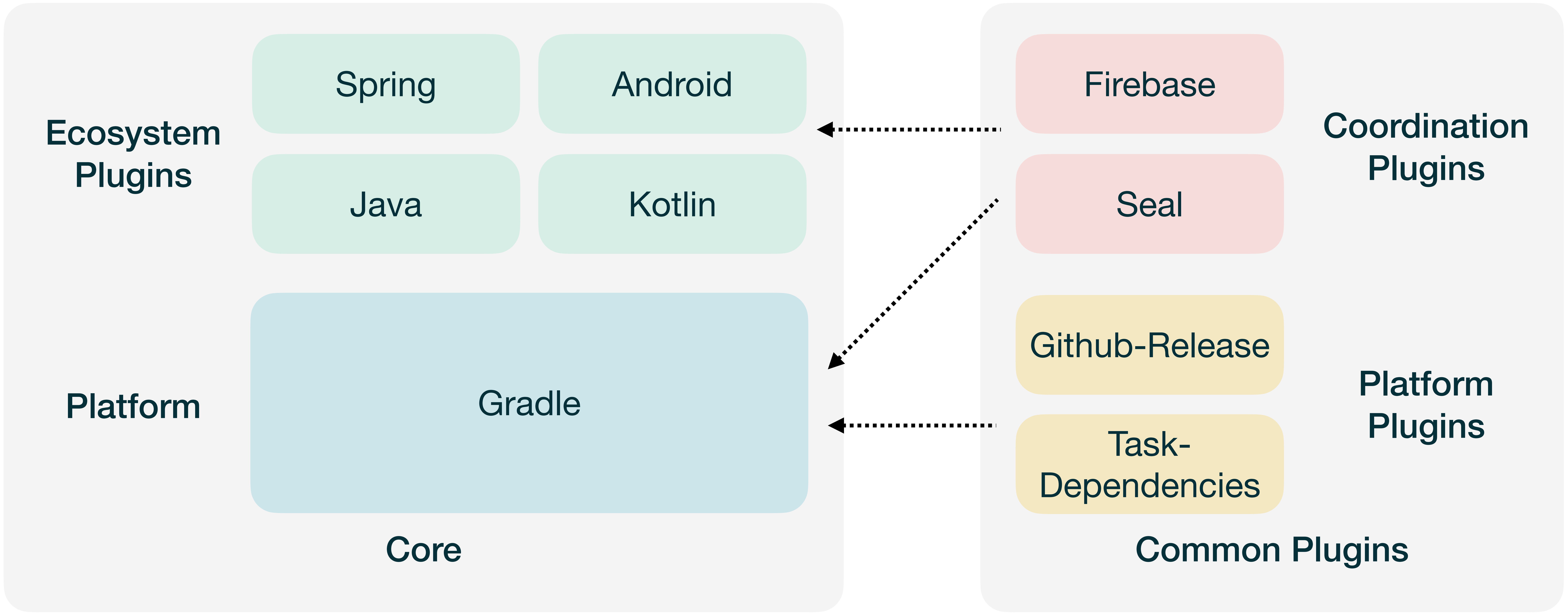
By 2BAB

Google Developers

目录

Google Developers

环境分层


Ecosystem Plugins
Spring
Android
Java
Kotlin
Platform
Gradle
Core
Firebase
Seal
Coordination Plugins
Github-Release
Task-Dependencies
Platform Plugins
Common Plugins

场景举例

- 快速生成 APK 多渠道包
- 基于版本号给所有图片打上盲水印
- 修复 Manifest 合并冲突
- 无痕埋点

扩展形式

脚本
Script

init.gradle.kts
build.gradle.kts
settings.gradle.kts

脚本插件
Script Plugin

maven.gradle.kts
firebase.gradle.kts
apply(from =
"script-plugin.gradle.kts")

二进制插件
Binary Plugin

Plugins { id(“abc”) }
apply(plugin = “abc”)

Google Developers

环境

- AGP: **7.0.3** / 7.1.0-beta04
- Gradle: **7.3**
- Kotlin: **1.5.31**
- 仅测试 **Application** Plugin

目录

什么是 Variant

```
buildTypes {
    getByName("debug") {
        isMinifyEnabled = false
    }
    getByName("release") {
        isMinifyEnabled = true
        proguardFiles(
            getDefaultProguardFile("…")
        )
    }
}
```

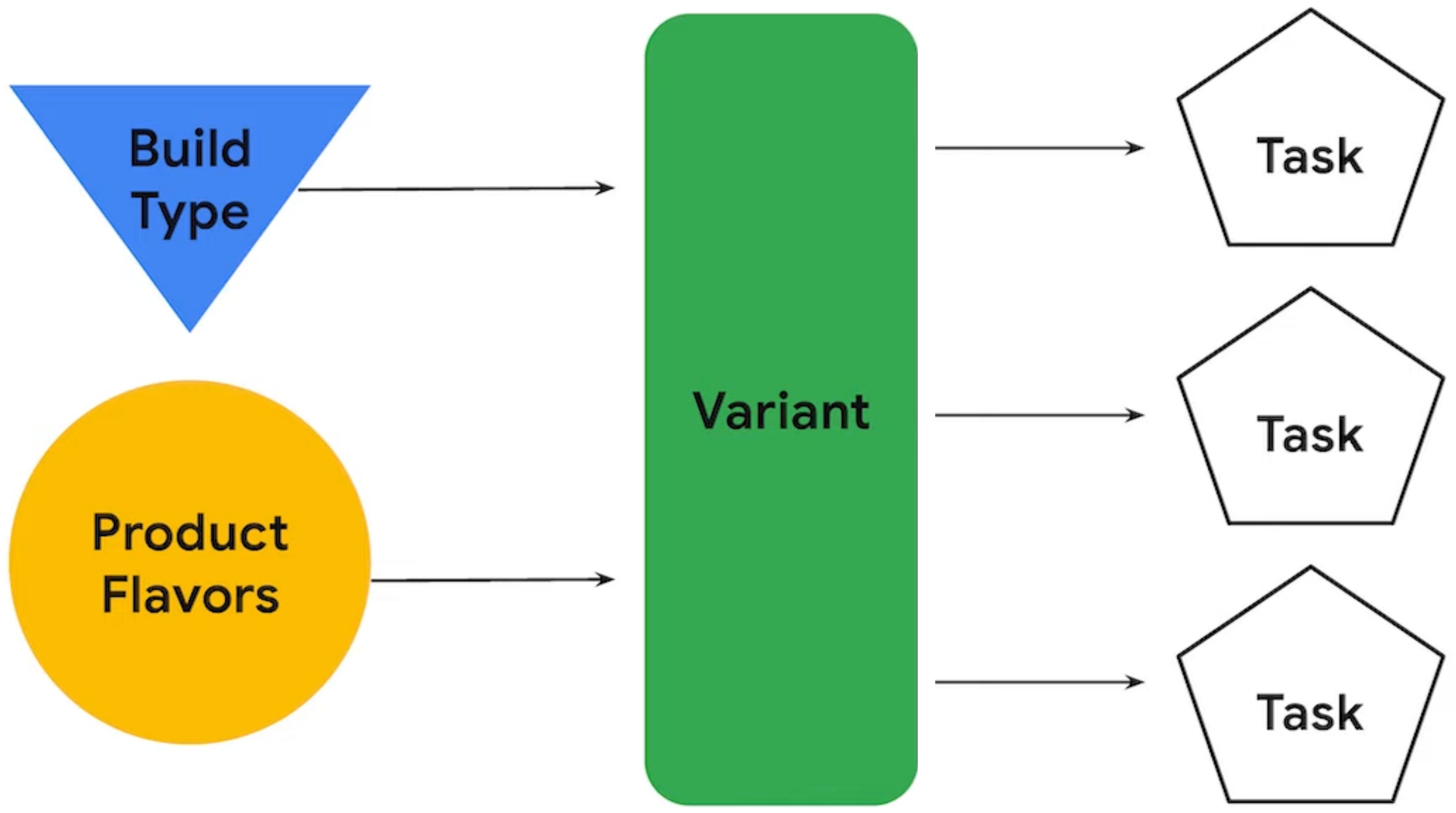
×

```
flavorDimensions += "server"
productFlavors {
    create("staging") {
        dimension = "server"
        applicationIdSuffix = ".staging"
        versionNameSuffix = "-staging"
    }
    create("production") {
        dimension = "server"
        applicationIdSuffix = ".production"
        versionNameSuffix = "-production"
        versionCode = 2
    }
}
```

- stagingDebug
- stagingRelease

- productionDebug
- productionRelease

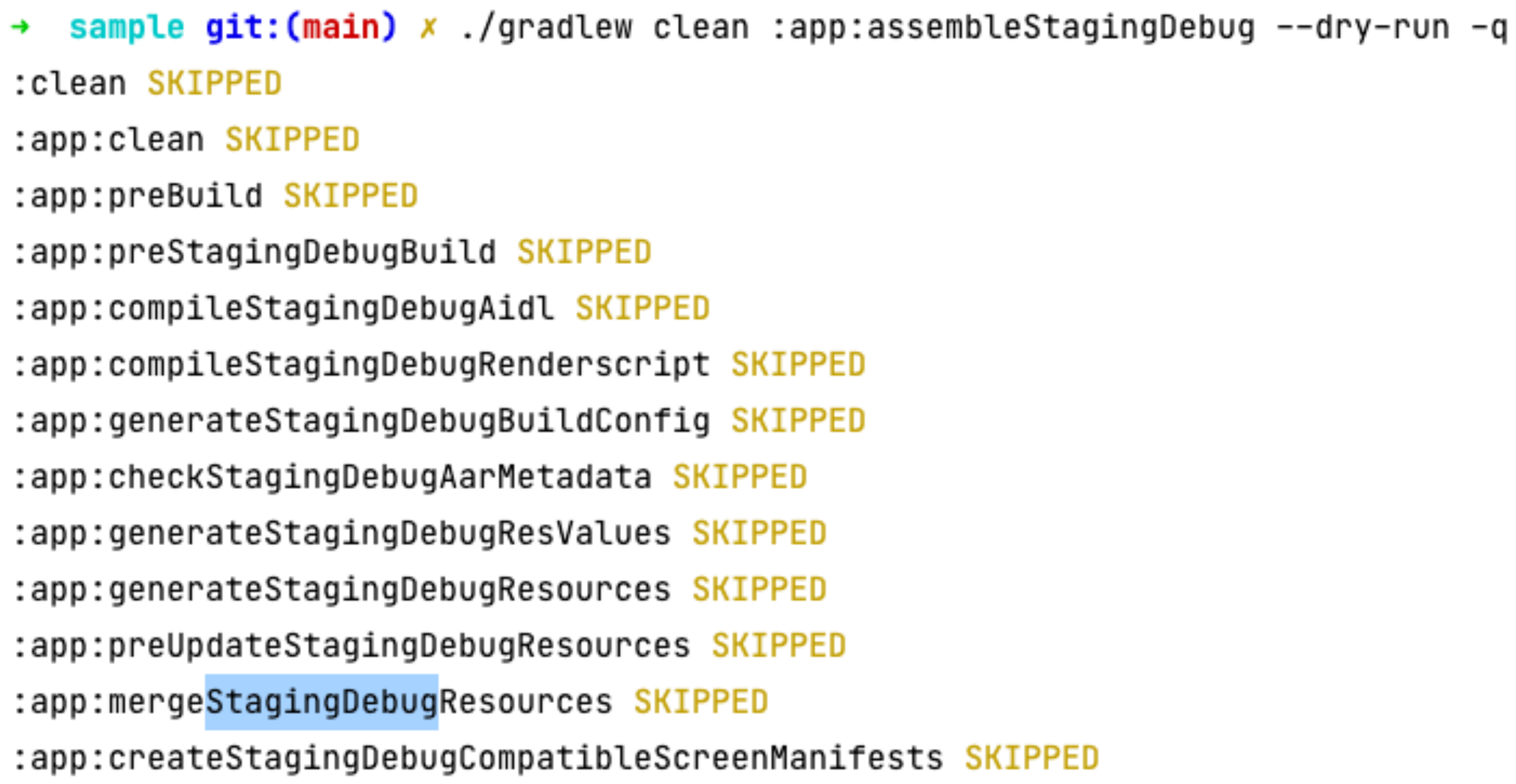
什么是 Variant



可感知变体的任务
variant-aware task

Google Developers

什么是 Variant



```
➜  sample git:(main) ✗ ./gradlew clean :app:assembleStagingDebug --dry-run -q
:clean SKIPPED
:app:clean SKIPPED
:app:preBuild SKIPPED
:app:preStagingDebugBuild SKIPPED
:app:compileStagingDebugAidl SKIPPED
:app:compileStagingDebugRenderscript SKIPPED
:app:generateStagingDebugBuildConfig SKIPPED
:app:checkStagingDebugAarMetadata SKIPPED
:app:generateStagingDebugResValues SKIPPED
:app:generateStagingDebugResources SKIPPED
:app:preUpdateStagingDebugResources SKIPPED
:app:mergeStagingDebugResources SKIPPED
:app:createStagingDebugCompatibleScreenManifests SKIPPED
```

什么是 Variant API

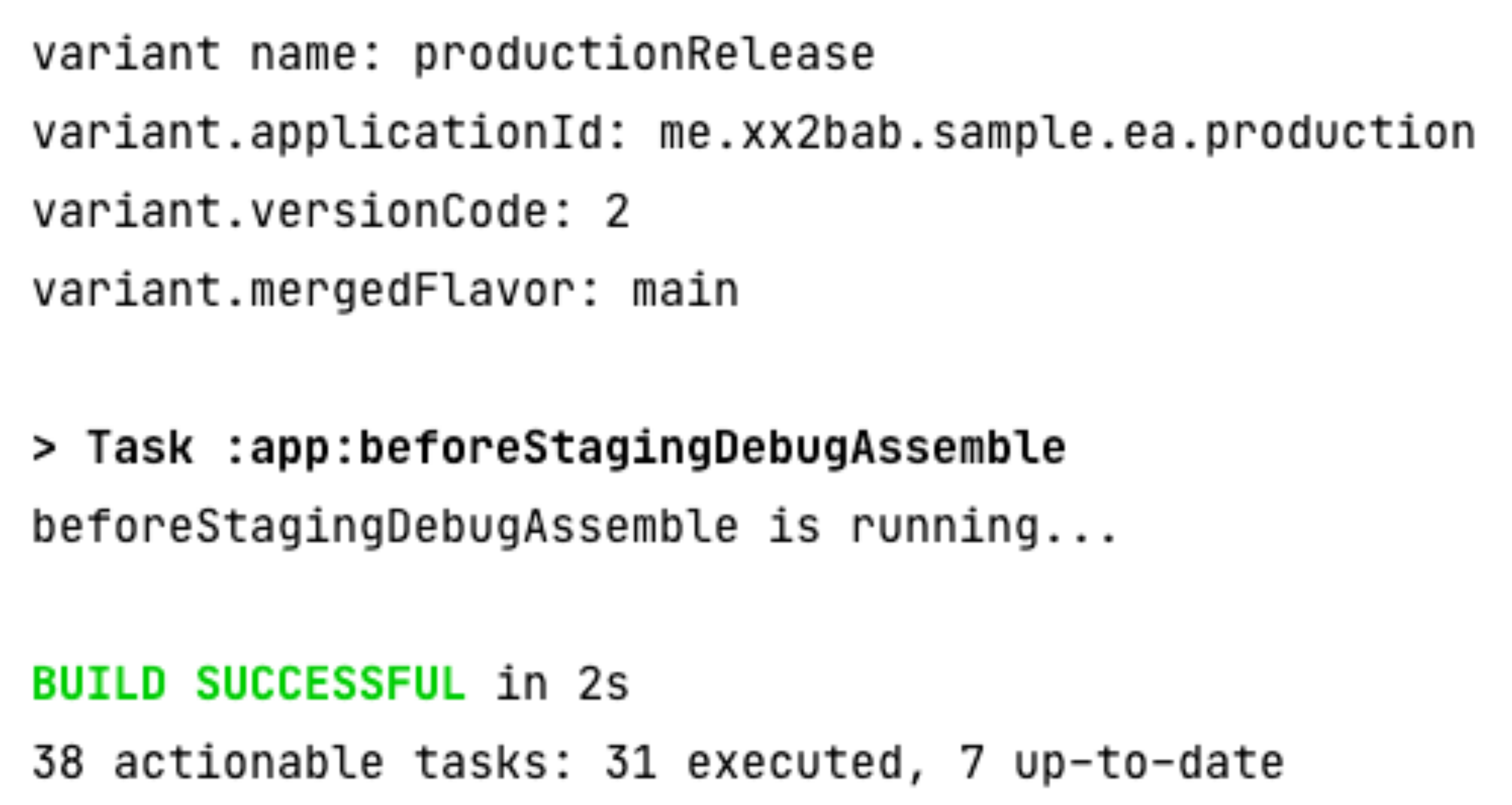
```
applicationVariants.all { variant ->
    variant.outputs.all { output ->
        def appId = variant.applicationId// com.exampleFree.app
        def versionName = variant.versionName
        def versionCode = variant.versionCode // e.g 1.0
        def flavorName = variant.flavorName // e. g. Free
        def buildType = variant.buildType.name // e. g. debug
        def variantName = variant.name // e. g. FreeDebug

        //customize your app name by using variables
        output.outputFileName = "${variantName}.apk"
    }
}}
```

Variant API v1 - 获取已配置内容

```
val android = project.extensions.getByType(AppExtension::class.java)
android.applicationVariants.configureEach {
    val variant: ApplicationVariant = this

    // Configurations (Reflect the DSL models)
    project.logger.lifecycle("variant name: ${variant.name}")
    project.logger.lifecycle("variant.applicationId: ${variant.applicationId}")
    project.logger.lifecycle("variant.versionCode: ${variant.versionCode}")
    project.logger.lifecycle("variant.mergedFlavor: ${variant.mergedFlavor.name}")

    // Task Providers
    val beforeAssemble = project.tasks.register(
        "before${variant.name.capitalize()}Assemble"
    ) {
        doFirst { project.logger.lifecycle("${this.name} is running...") }
    }
    variant.assembleProvider.configure {
        dependsOn(beforeAssemble)
    }
    …
}
```

Google Developers

Variant API v1 - 获取已配置内容



```
variant name: productionRelease
variant.applicationId: me.xx2bab.sample.ea.production
variant.versionCode: 2
variant.mergedFlavor: main

> Task :app:beforeStagingDebugAssemble
beforeStagingDebugAssemble is running...

BUILD SUCCESSFUL in 2s
38 actionable tasks: 31 executed, 7 up-to-date
```

Variant API v1 - 二次配置

```
val android = project.extensions.getByType(AppExtension::class.java)
android.applicationVariants.configureEach {
    …
    if (variant.name.contains("release", true)
        && variant.name.contains("production", true)
    ) {
        (variant.mergedFlavor as MergedFlavor).setSigningConfig(…)
    }

}
```

- productionDebug: signature A
- productionRelease: signature B

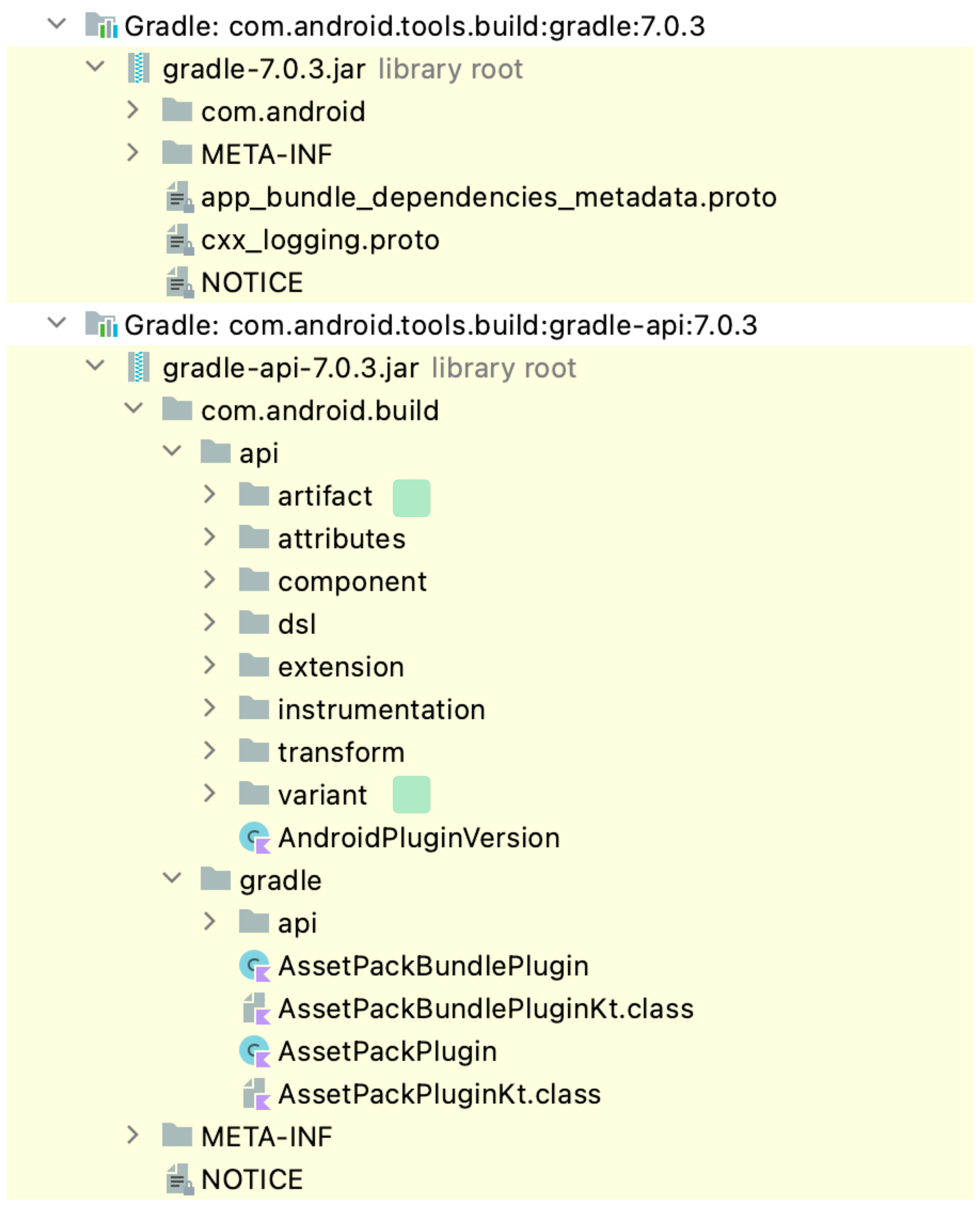
Artifact API v1 - 入口

```
val android = project.extensions.getByType(AppExtension::class.java)
android.applicationVariants.configureEach {
    …
  ▢ variant.outputs.forEach { output ->
        val file = output.outputFile
        if (file.extension == "apk") {
            …
        }
    }
}
```

Artifact -> 工件 / 产物

Google Developers

Variant API v2 - AGP 分包



```
⌄ Gradle: com.android.tools.build:gradle:7.0.3
  ⌄ gradle-7.0.3.jar library root
    › com.android
    › META-INF
      app_bundle_dependencies_metadata.proto
      cxx_logging.proto
      NOTICE
⌄ Gradle: com.android.tools.build:gradle-api:7.0.3
  ⌄ gradle-api-7.0.3.jar library root
    ⌄ com.android.build
      ⌄ api
        › artifact 🟩
        › attributes
        › component
        › dsl
        › extension
        › instrumentation
        › transform
        › variant 🟩
          AndroidPluginVersion
      ⌄ gradle
        › api
          AssetPackBundlePlugin
          AssetPackBundlePluginKt.class
          AssetPackPlugin
          AssetPackPluginKt.class
    › META-INF
      NOTICE
```

:gradle -> 实现细节

:gradle-api -> 公开 API

协同插件的开发理论上

只需要依赖:gradle-api

Variant API v2 - 获取已配置内容

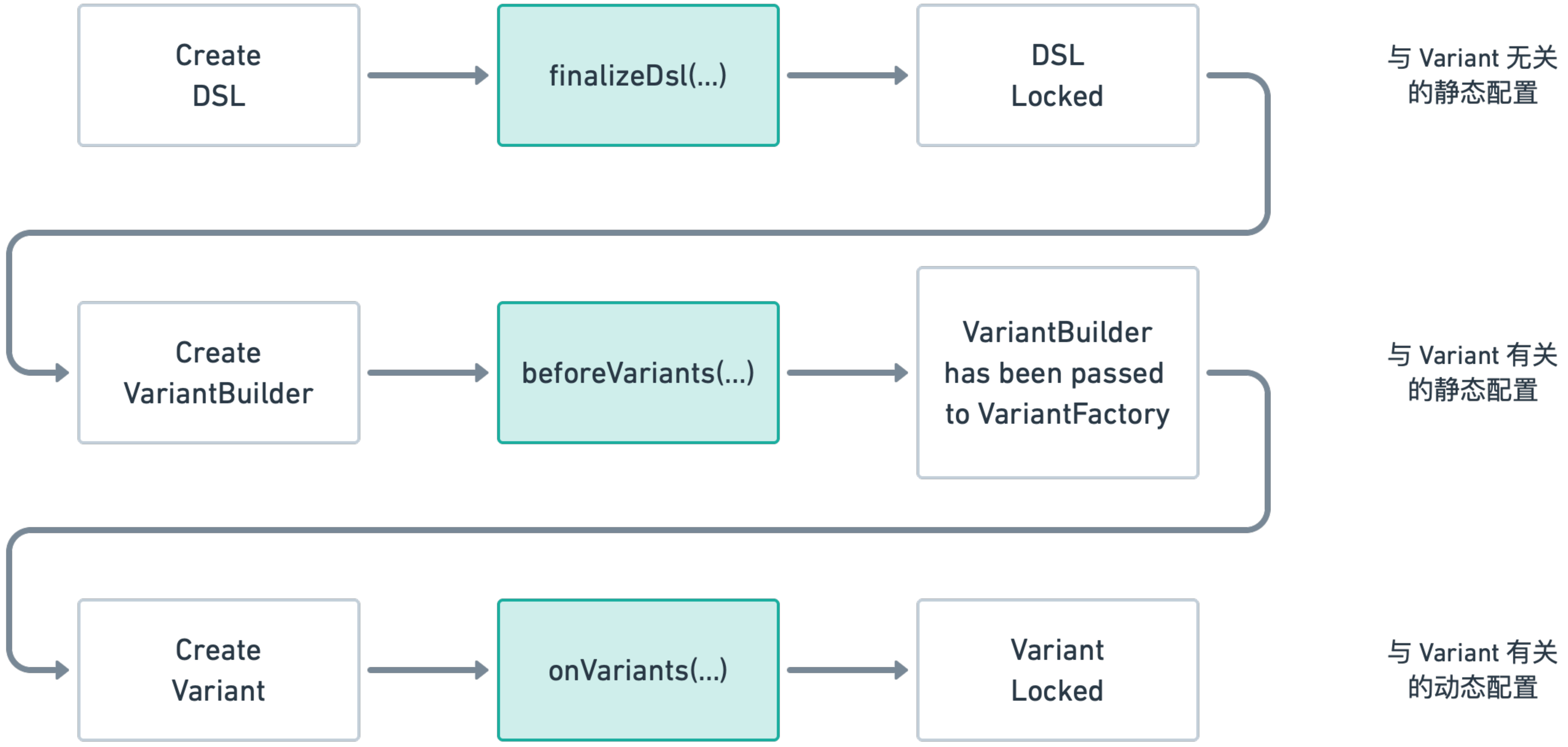
```
val androidExtension = project.extensions
        .getByType(ApplicationAndroidComponentsExtension::class.java)
androidExtension.onVariants { variant ->

    // Configurations (Reflect the DSL models)
    val mainOutput: VariantOutput = variant.outputs.single {
        it.outputType == VariantOutputConfiguration.OutputType.SINGLE
    }
    project.logger.lifecycle("variant name: ${variant.name}")
    project.logger.lifecycle("variant.applicationId: ${variant.namespace.get()}")
    project.logger.lifecycle("variant.versionCode: ${mainOutput.versionCode.get()}")
    project.logger.lifecycle("variant.productFlavors: ${variant.productFlavors.size}")


    // Task Providers are removed from new variant APIs.

    …
}
```

Google Developers

Variant API v2 - 二次配置

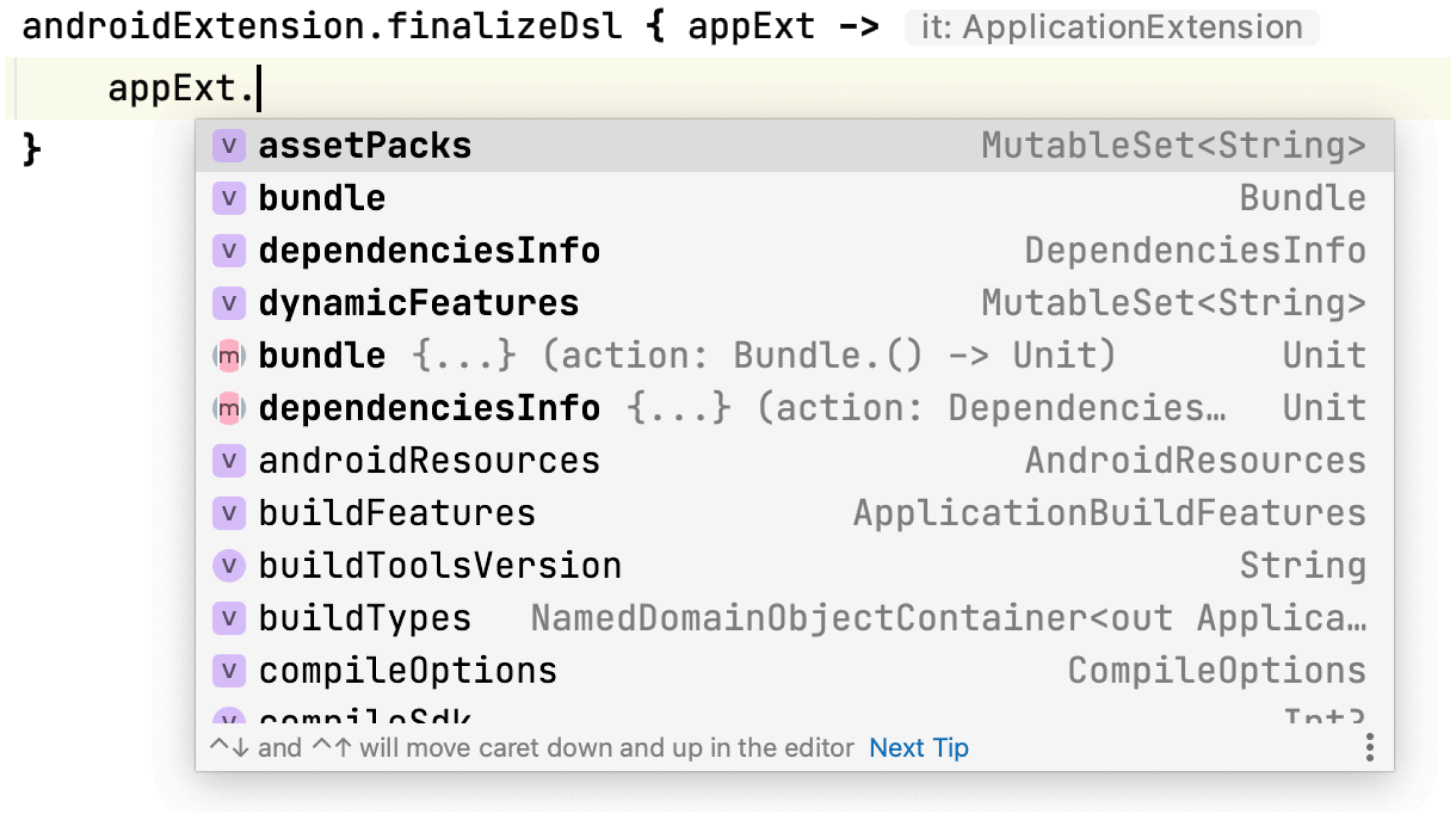
```
androidExtension.onVariants(
    androidExtension
        .selector()
        .withBuildType("release")
        .withFlavor(Pair("server", "production"))
) { variant ->
    val mainOutput: VariantOutput = variant.outputs.single {
        it.outputType == VariantOutputConfiguration.OutputType.SINGLE
    }
    mainOutput.versionName.set("1.1.0")
    variant.androidResources.aaptAdditionalParameters.add("-v")
    // variant.signingConfig?.setConfig(...)
}
```

Variant API v2 - 新生命周期


Create DSL
finalizeDsl(...)
DSL Locked
与 Variant 无关的静态配置
Create VariantBuilder
beforeVariants(...)
VariantBuilder has been passed to VariantFactory
与 Variant 有关的静态配置
Create Variant
onVariants(...)
Variant Locked
与 Variant 有关的动态配置

Google Developers

Variant API v2 - 新生命周期



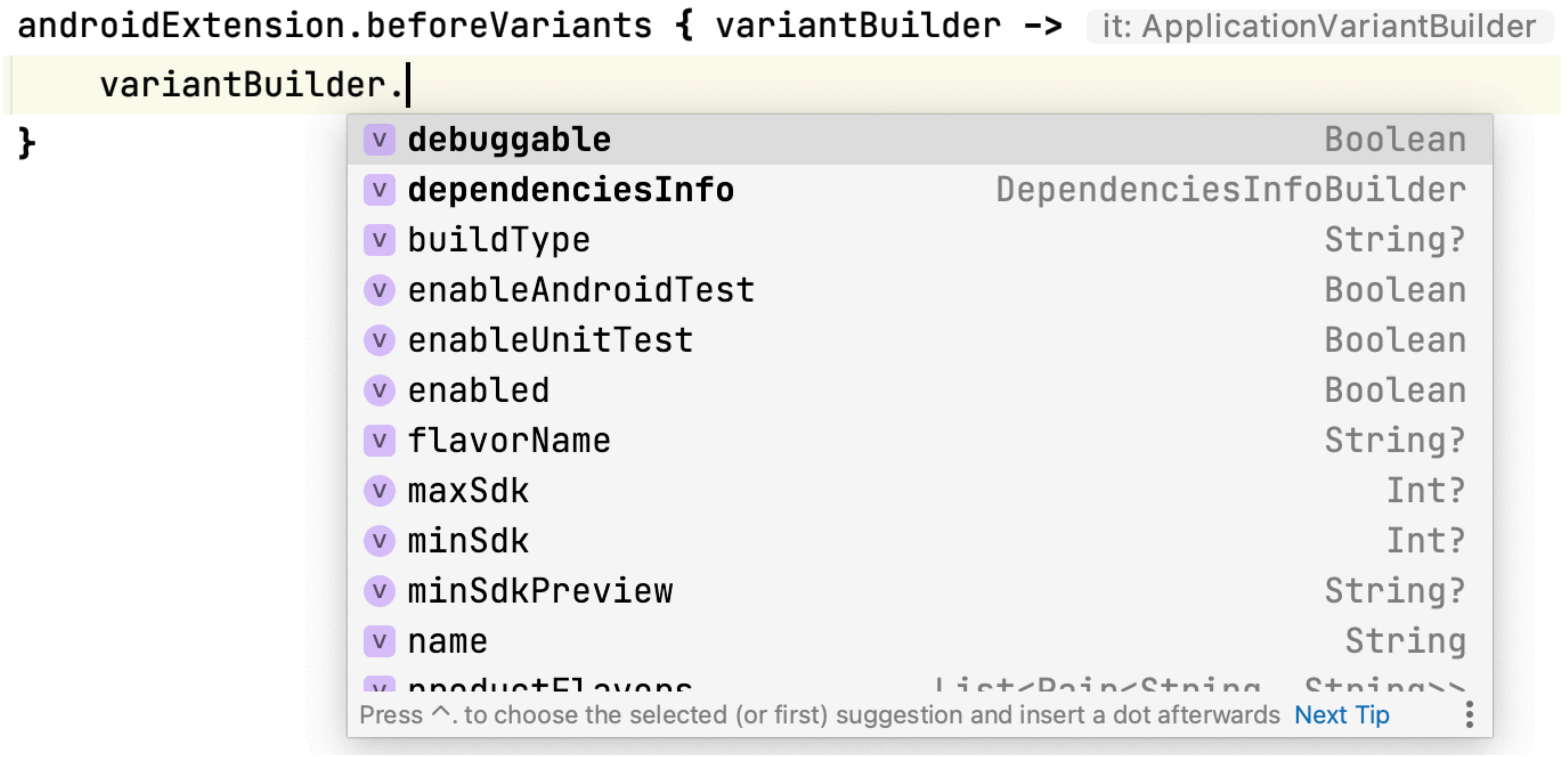
```
androidExtension.finalizeDsl { appExt ->  it: ApplicationExtension
    appExt.
}
```

```
v assetPacks                                          MutableSet<String>
v bundle                                                          Bundle
v dependenciesInfo                                      DependenciesInfo
v dynamicFeatures                                     MutableSet<String>
m bundle {...} (action: Bundle.() -> Unit)                          Unit
m dependenciesInfo {...} (action: Dependencies…                     Unit
v androidResources                                      AndroidResources
v buildFeatures                                 ApplicationBuildFeatures
v buildToolsVersion                                               String
v buildTypes   NamedDomainObjectContainer<out Applica…
v compileOptions                                          CompileOptions
```

^↓ and ^↑ will move caret down and up in the editor Next Tip

Google Developers

Variant API v2 - 新生命周期



```
androidExtension.beforeVariants { variantBuilder ->  it: ApplicationVariantBuilder
    variantBuilder.
}
```

```
v debuggable                          Boolean
v dependenciesInfo                    DependenciesInfoBuilder
v buildType                           String?
v enableAndroidTest                   Boolean
v enableUnitTest                      Boolean
v enabled                             Boolean
v flavorName                          String?
v maxSdk                              Int?
v minSdk                              Int?
v minSdkPreview                       String?
v name                                String
```

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip

Google Developers

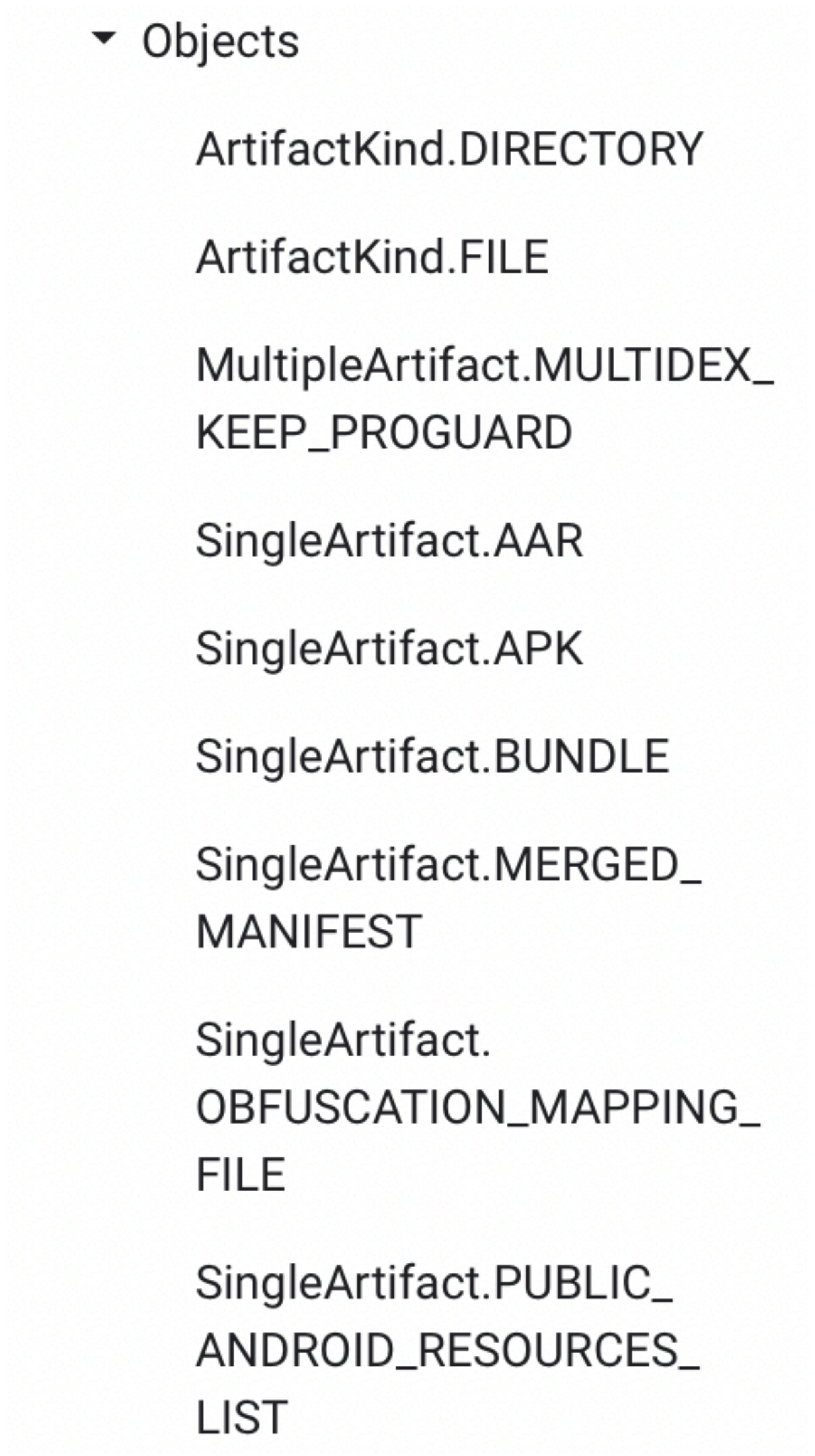
Variant API v2 - 新生命周期

```
androidExtension.beforeVariants(
    androidExtension
        .selector()
        .withName("productionDebug")
) { variantBuilder ->
    variantBuilder.enabled = false
}
```

禁用不需要的组合，加快配置速度

Google Developers

Artifact API v2 - 入口（7.0)



▾ Objects

- ArtifactKind.DIRECTORY
- ArtifactKind.FILE
- MultipleArtifact.MULTIDEX_KEEP_PROGUARD
- SingleArtifact.AAR
- SingleArtifact.APK
- SingleArtifact.BUNDLE
- SingleArtifact.MERGED_MANIFEST
- SingleArtifact.OBFUSCATION_MAPPING_FILE
- SingleArtifact.PUBLIC_ANDROID_RESOURCES_LIST

```
androidExtension.onVariants(
    androidExtension
        .selector()
        .withBuildType("release")
        .withFlavor(Pair("server", "production"))
) { variant ->
   …
   val apkFolderProvider = variant.artifacts.get(SingleArtifact.APK)
}
```

Google Developers

Variant/Artifact API v1/v2 对比

1. API v2 **语义更明确**：隔离出独立的 AndroidComponents extension
2. API v2 的 Variant **生命周期更清晰**：增加了多个回调节点，支持对象锁定
3. API v2 开始实现**动静分离**：每个节点处理特定的一部分配置
4. Artifact 终于有**正式暴露的、稳定的 API**，并且做了包拆分

目录

API v1 - 重命名 APK

```
abstract class RenameApkFile : DefaultTask() {

    @get:InputFile
    lateinit var inputApk: File

    @get:OutputFile
    lateinit var outputApk: File

    @TaskAction
    fun taskAction() {
        inputApk.copyTo(outputApk)
    }
}
```

Google Developers

API v1 - 重命名 APK

```
val android = project.extensions.getByType(AppExtension::class.java)
android.applicationVariants.configureEach {
    val variant: ApplicationVariant = this
    val variantCapitalizedName = variant.name.capitalize()

    variant.outputs.forEach { output ->
        val file = output.outputFile 🟩
        if (file.extension == "apk") {
            // output.outputFileName = "custom-" + variant.veresionName
            val out = File(file.parentFile, "custom-${variant.versionName}")
            val renameApkTask = project.tasks.register(
                "rename${variantCapitalizedName}Apk",
                RenameApkFile::class.java
            ) {
                inputApk = file 🟩
                outputApk = out
                dependsOn(variant.packageApplicationProvider) 🟩
            }
            …
        }
    }
}
```

1

2

API v1 - 扩展插件两要素

Input Artifact

&

Task Dependency

API v1 - 扩展插件两要素

但除了最终产物 APK、AAB、AAR 等以外，其他产物只能按如右步骤获取和使用：

1. 通过执行的命令列表快速定位相关的 AGP Task
2. 阅读和 Debug 源码，找到所需要 Input Artifact
3. 使用 `dependsOn(...)` 等方法插入自定义 Task，确保会在特定时刻执行

API v1 - 扩展插件两要素

Raw Gradle API

+

Hook

API v1 - 获取合并的 Manifest

```
val processManifestTask = project.tasks
    .withType(ProcessApplicationManifest::class.java).first {
        it.name.contains(variant.name, true)
    }
val postUpdateManifestTask = project.tasks
    .register(
        "postUpdate${variantCapitalizedName}Manifest",
        ManifestAfterMergeTask::class.java
    ) {
        mergedManifest = processManifestTask.mergedManifest
            .get()
            .asFile
    }
// Abuse of finalizedBy()
processManifestTask.finalizedBy(postUpdateManifestTask)
```

API v1 - 获取合并的 Manifest

```kotlin
abstract class ManifestAfterMergeTask : DefaultTask() {

    @get:InputFile
    lateinit var mergedManifest: File

    @TaskAction
    fun afterMerge() {
        val modifiedManifest = mergedManifest.readText()
            .replace("allowBackup=\"true\"", "allowBackup=\"false\"")
        mergedManifest.writeText(modifiedManifest)
    }

}
```

API v1 - 获取所有的 Resources

```kotlin
/**
 * To get all original resources including libraries
 */
fun MergeResources.computeResourceList(): List<File> {
    val resourcesComputer = ReflectionKit.getField(
        MergeResources::class.java,
        this,
        "resourcesComputer"
    ) as DependencyResourcesComputer
    val resourceSets = resourcesComputer.compute(this.processResources, null)
    return resourceSets.mapNotNull { resourceSet ->
        val getSourceFiles = resourceSet.javaClass.methods.find {
            it.name == "getSourceFiles" && it.parameterCount == 0
        }
        val files = getSourceFiles?.invoke(resourceSet)
        @Suppress("UNCHECKED_CAST")
        files as? Iterable<File>
    }.flatten()
}
```

API v1 - 含 APK 文件大小的构建通知

```
// Let's assume below is provided by a 3rd party SDK
abstract class NotificationTask : DefaultTask() {

    @get:Input
  🟩 lateinit var title: String

    @get:InputFile
  🟩 lateinit var releaseNote: File

    @TaskAction
    fun taskAction() {
        val msg = "$title\n${releaseNote.readText()}"
        val channel = "123456789"
        NotificationClient().send(msg, channel)
    }

}
```

```
abstract class ApkSizeObtainTask : DefaultTask() {

    @get:InputFile
    lateinit var apk: File

    @get:OutputFile
    lateinit var releaseNote: File

    @TaskAction
    fun taskAction() {
        val size = apk.length() / 1024.0 / 1024.0
        releaseNote.writeText("Apk - $size MB")
    }

}
```

API v1 - 含 APK 文件大小的构建通知

```
val releaseNoteFile = File(file.parentFile, "release-note.txt")
val apkSizeObtainTask = project.tasks.register(
    "apkSizeObtain$variantCapitalizedName",
    ApkSizeObtainTask::class.java
) {
    apk = file
    releaseNote = releaseNoteFile
    dependsOn(renameApkTask)
}
val notificationTask = project.tasks.register(
    "notify${variantCapitalizedName}Build",
    NotificationTask::class.java
) {
    title = "${project.name} apk is built successfully."
    releaseNote = releaseNoteFile
    dependsOn(apkSizeObtainTask)
}
```

Google Developers

API v1 - 扩展的困难

1. Artifact 部分**只暴露了最终产物**（.../build/outputs），**无其他中间产物**
2. 通过一些 Raw Gradle API 加上 **Hook** 手段使用 AGP 内部任务的成员变量、方法会导致**后期难以维护**
3. 无法在**配置阶段**获得所有任务所需的**输入参数（只能基于 File 做中转**

目录

API v2 - 重命名 APK

```
val renameApkTask = project.tasks.register(
    "rename${variantCapitalizedName}Apk",
    RenameApkFile::class.java
) {
    val apkFolderProvider = variant.artifacts.get(SingleArtifact.APK)
    this.outApk.set(
        File(project.buildDir, "custom-${mainOutput.versionName}")
    )
    this.apkFolder.set(apkFolderProvider)
    this.builtArtifactsLoader.set(variant.artifacts.getBuiltArtifactsLoader())
}
```

API v2 - 重命名 APK

```
abstract class RenameApkFile : DefaultTask() {

    @get:InputFiles
    abstract val apkFolder: DirectoryProperty 🟩

    @get:Internal
    abstract val builtArtifactsLoader: Property<BuiltArtifactsLoader> 🟩

    @get:OutputFile
    abstract val outApk: RegularFileProperty 🟩

    @TaskAction
    fun taskAction() {
        val builtArtifacts = builtArtifactsLoader.get().load(apkFolder.get())
            ?: throw RuntimeException("Cannot load APKs")
        File(builtArtifacts.elements.single().outputFile)
            .copyTo(outApk.get().asFile)
    }
}
```

API v2 - ~~获取~~修改合并的 Manifest

```
val postUpdateTask = project.tasks.register(
    "postUpdate${variantCapitalizedName}Manifest",
    ManifestAfterMergeTask::class.java
)
variant.artifacts // .get(SingleArtifact.MERGED_MANIFEST)
  🟩 .use(postUpdateTask)
  🟩 .wiredWithFiles(
        ManifestAfterMergeTask::mergedManifest,
        ManifestAfterMergeTask::updatedManifest
    )
  🟩 .toTransform(SingleArtifact.MERGED_MANIFEST)
```

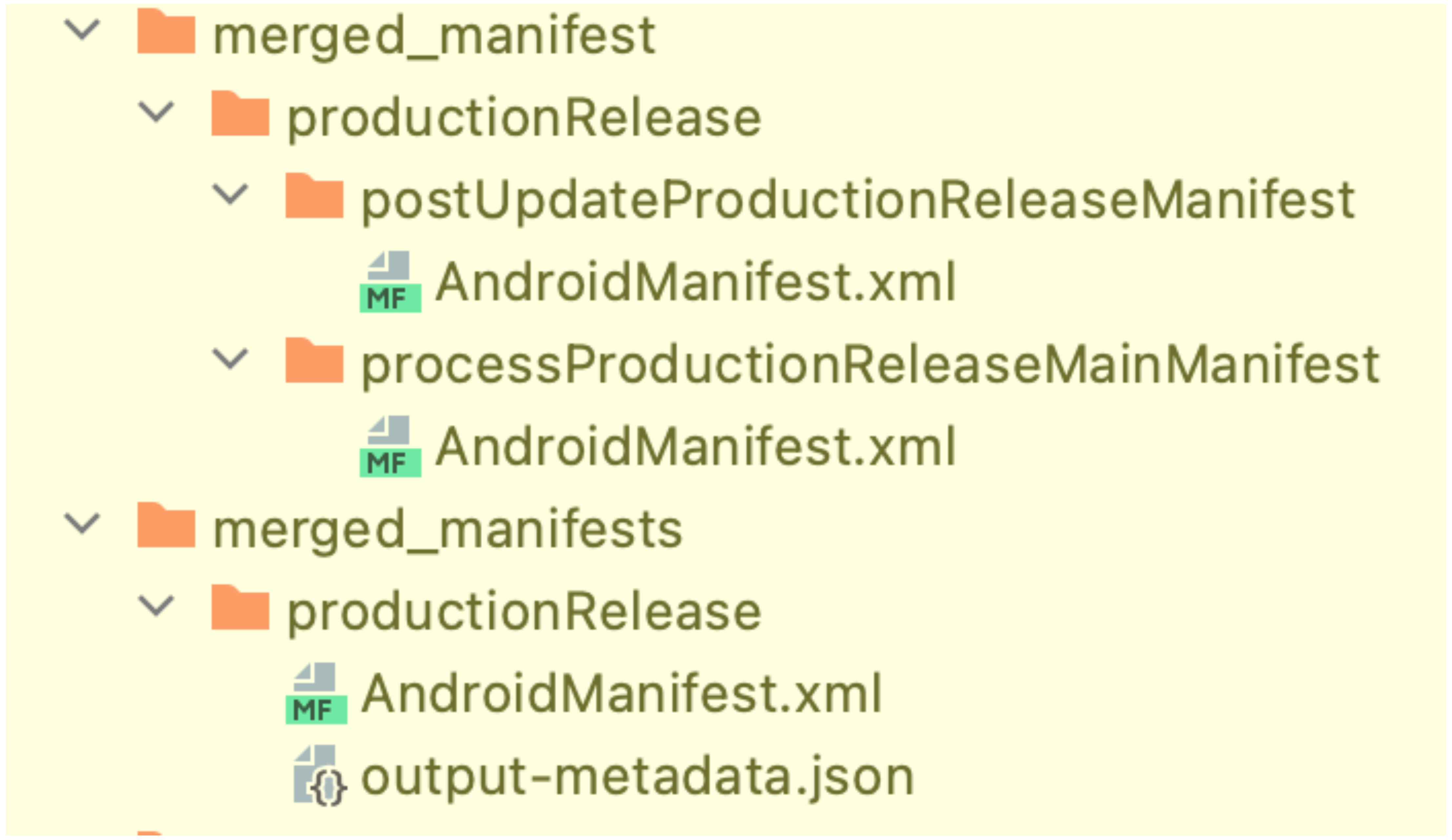
Google Developers

API v2 - 修改合并的 Manifest

```
abstract class ManifestAfterMergeTask : DefaultTask() {

    @get:InputFile
    abstract val mergedManifest: RegularFileProperty

    @get:OutputFile
    abstract val updatedManifest: RegularFileProperty

    @TaskAction
    fun afterMerge() {
        mergedManifest.get().asFile.copyTo(updatedManifest.get().asFile)
    }

}
```

API v2 - 获取合并的 Manifest

Pipleline



```
merged_manifest
  productionRelease
    postUpdateProductionReleaseManifest
      AndroidManifest.xml
    processProductionReleaseMainManifest
      AndroidManifest.xml
merged_manifests
  productionRelease
    AndroidManifest.xml
    output-metadata.json
```

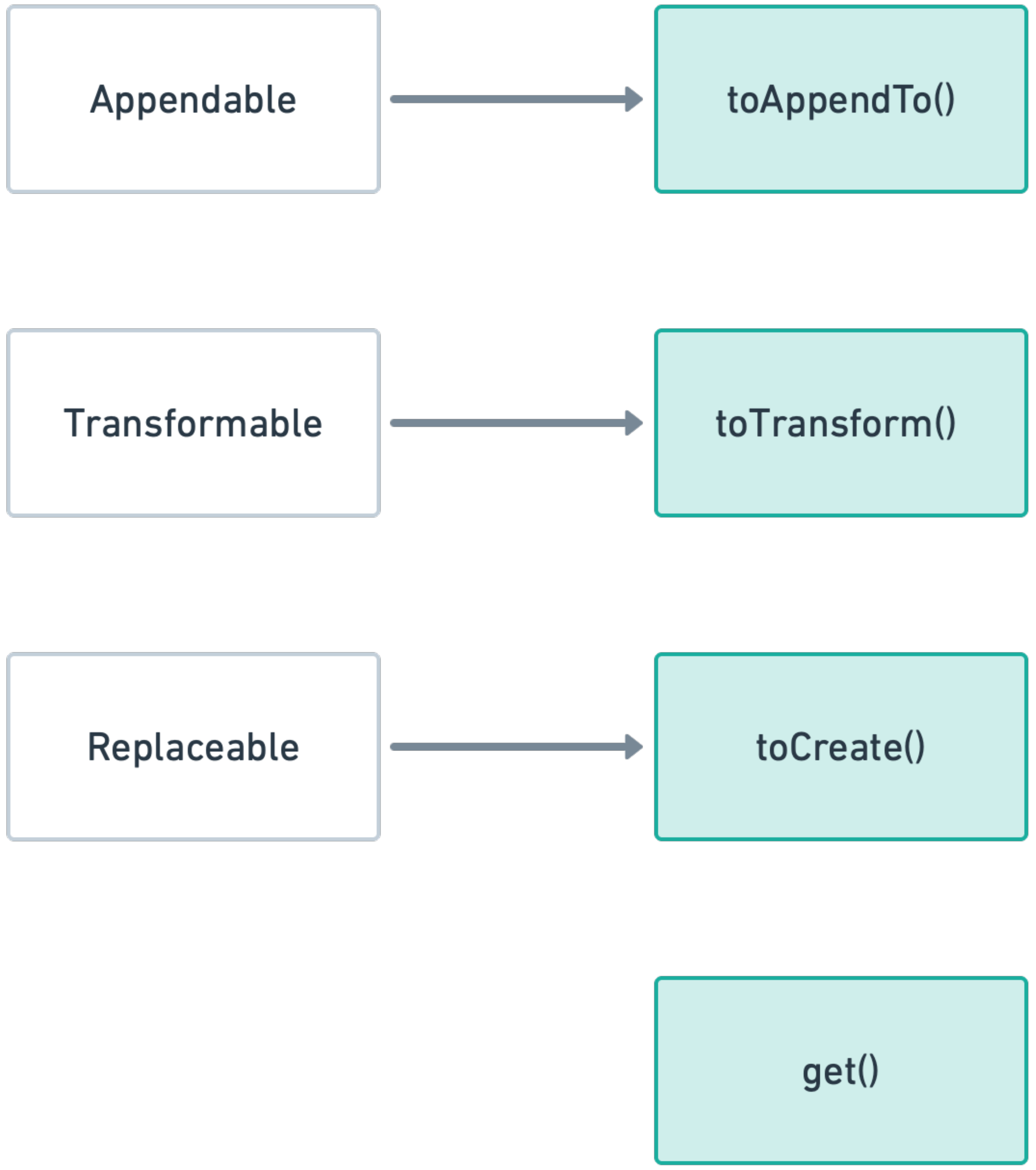
2

1

3

```
variant.artifacts
    .use(postUpdateTask)
    .wiredWithFiles(
        ManifestAfterMergeTask::mergedManifest,
        ManifestAfterMergeTask::updatedManifest
    )
    .toTransform(SingleArtifact.MERGED_MANIFEST)
```

Variant API v2 - 四种操作（7.1.0/7.2.0）

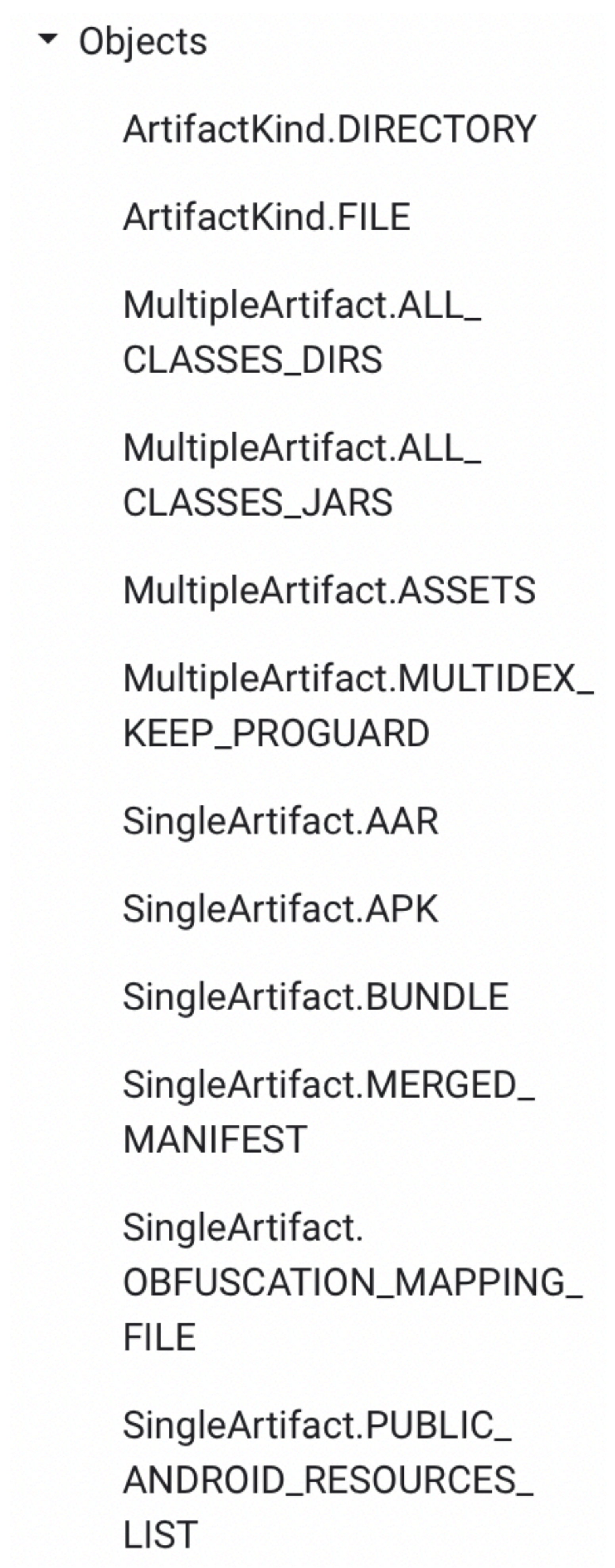


```
variant.artifacts
     .use(xxxxTask)
     .wiredWithFiles(…)
               / wiredWithDirectories(…)
               / wiredWith(…)
     .toXxxx(SingleArtifact.XXX
               / MultipleArtifact.XXX)
```

```
variant.artifacts
     .get(SingleArtifact.XXX
           / MultipleArtifact.XXX)
```

Google Developers

Variant API v2 - 四种操作（7.1.0/7.2.0）

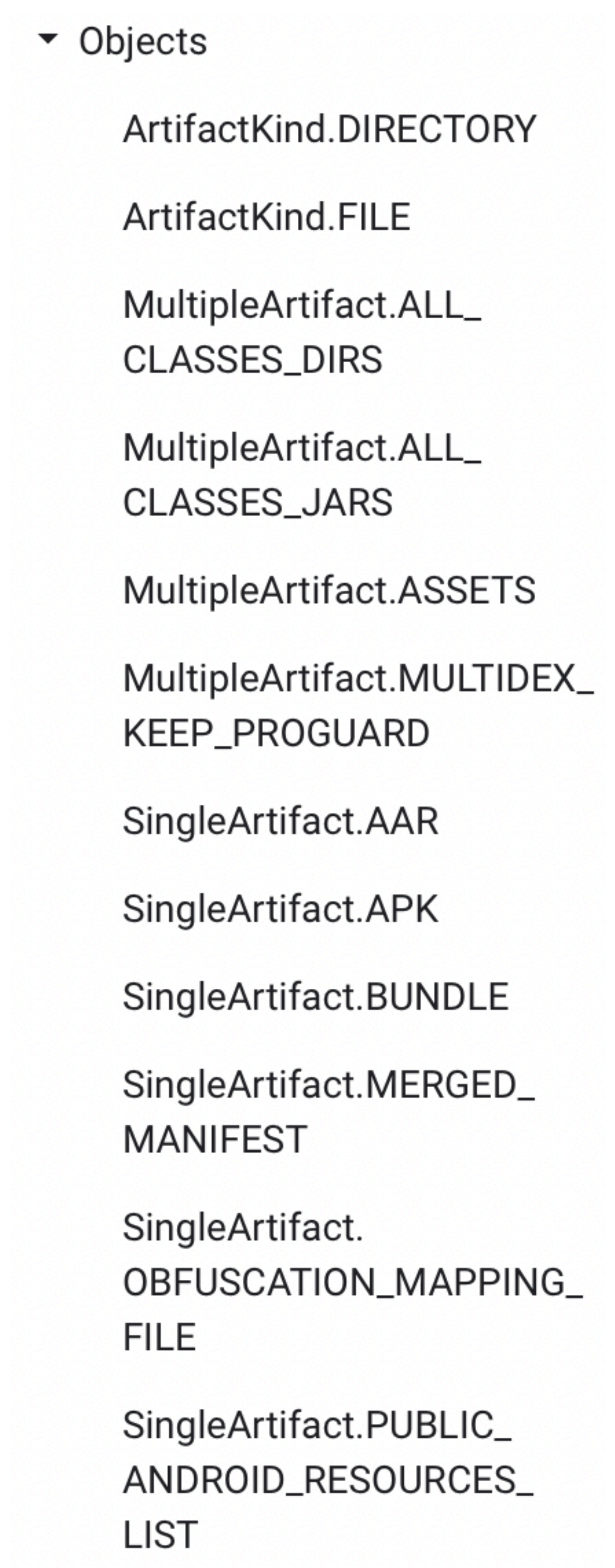


▾ Objects

- ArtifactKind.DIRECTORY
- ArtifactKind.FILE
- MultipleArtifact.ALL_CLASSES_DIRS
- MultipleArtifact.ALL_CLASSES_JARS
- MultipleArtifact.ASSETS
- MultipleArtifact.MULTIDEX_KEEP_PROGUARD
- SingleArtifact.AAR
- SingleArtifact.APK
- SingleArtifact.BUNDLE
- SingleArtifact.MERGED_MANIFEST
- SingleArtifact.OBFUSCATION_MAPPING_FILE
- SingleArtifact.PUBLIC_ANDROID_RESOURCES_LIST

```kotlin
sealed class MultipleArtifact<FileTypeT : FileSystemLocation>(…
   ) : Artifact.Multiple<FileTypeT>(kind, category) {
  @Incubating
  object ALL_CLASSES_JARS:
      MultipleArtifact<RegularFile>(FILE),
      Appendable,
      Transformable,
      Replaceable
  @Incubating
  object ASSETS:
      MultipleArtifact<Directory>(DIRECTORY),
      Appendable,
      Transformable,
      Replaceable
  …
}
```

Variant API v2 - 四种操作（7.1.0/7.2.0）



- Objects
  - ArtifactKind.DIRECTORY
  - ArtifactKind.FILE
  - MultipleArtifact.ALL_CLASSES_DIRS
  - MultipleArtifact.ALL_CLASSES_JARS
  - MultipleArtifact.ASSETS
  - MultipleArtifact.MULTIDEX_KEEP_PROGUARD
  - SingleArtifact.AAR
  - SingleArtifact.APK
  - SingleArtifact.BUNDLE
  - SingleArtifact.MERGED_MANIFEST
  - SingleArtifact.OBFUSCATION_MAPPING_FILE
  - SingleArtifact.PUBLIC_ANDROID_RESOURCES_LIST

```
sealed class SingleArtifact<T : FileSystemLocation>(…)
    : Artifact.Single<T>(kind, category) {

  object APK:
      SingleArtifact<Directory>(DIRECTORY),
      Transformable,
      Replaceable,
      ContainsMany

  object MERGED_MANIFEST:
      SingleArtifact<RegularFile>(FILE,
              Category.INTERMEDIATES, "AndroidManifest.xml"),
      Replaceable,
      Transformable
  …
}
```

Provider<T> - 简介

1. 延迟一切计算到需要的时候
2. 例如延迟配置期间的计算到执行期间
3. 可类比 **Supplier<T>** from Java 8 或者 **Lazy<T>** from Dagger
4. Provider<T>#get()
5. Property<T>#set(...)
6. 注意区分原始类型和惰性类型（包装后）
7. 例如 String 和 Property<String>,

   RegularFile 和 RegularFileProperty

Provider<T> - 含 APK 文件大小的构建通知

```
abstract class NotificationTask : DefaultTask() {

    @get:Input
    abstract val title: Property<String> 🟩

    @get:Input
    abstract val releaseNote: Property<String>

    @TaskAction
    fun taskAction() {
        val message = "${title.get()}\n${releaseNote.get()}"🟩
        val channel = "123456789"
        NotificationClient().send(message, channel)
    }

}
```

Google Developers

Provider<T> - 含 APK 文件大小的构建通知

```
project.tasks.register(
    "notify${variantCapitalizedName}Build",
    NotificationTask::class.java
) {
  🟩 title.set("${project.name} apk is built successfully.")
  🟩 releaseNote.set(renameApkTask.map {
        val size = it.outApk.get().asFile.length() / 1024.0 / 1024.0
        "Apk - $size MB"
    })
}
```

计算和引用分离

Google Developers

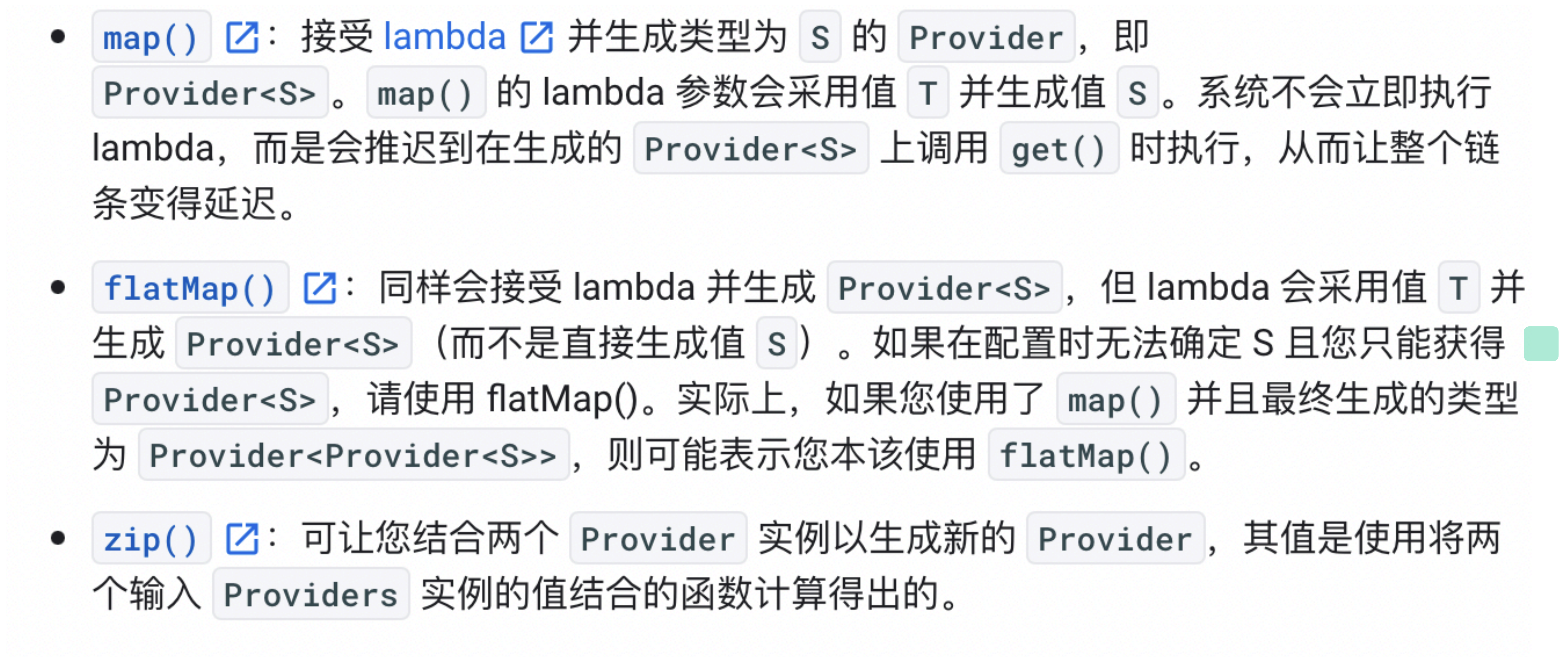
Provider<T> - map/flatMap/zip 变换

```
renameApkTask.map {
    val size = it.outApk.get().asFile.length() / 1024.0 / 1024.0
    "Apk - $size MB"
}
```

map(...)?

Google Developers

Provider<T> - map/flatMap/zip 变换



- `map()` ↗：接受 lambda ↗ 并生成类型为 `S` 的 `Provider`，即 `Provider<S>`。`map()` 的 lambda 参数会采用值 `T` 并生成值 `S`。系统不会立即执行 lambda，而是会推迟到在生成的 `Provider<S>` 上调用 `get()` 时执行，从而让整个链条变得延迟。
- `flatMap()` ↗：同样会接受 lambda 并生成 `Provider<S>`，但 lambda 会采用值 `T` 并生成 `Provider<S>`（而不是直接生成值 `S`）。如果在配置时无法确定 S 且您只能获得 `Provider<S>`，请使用 flatMap()。实际上，如果您使用了 `map()` 并且最终生成的类型为 `Provider<Provider<S>>`，则可能表示您本该使用 `flatMap()`。
- `zip()` ↗：可让您结合两个 `Provider` 实例以生成新的 `Provider`，其值是使用将两个输入 `Providers` 实例的值结合的函数计算得出的。

Lambda 返回 String

(大部分情况下使用 map)

Lambda 返回 Provider<String>

活数据特性（类比 LiveData）

Provider<T> - 自动化依赖处理

```
project.tasks.register(
    "notify${variantCapitalizedName}Build",
    NotificationTask::class.java
) {
    title.set("…")
    releaseNote.set(renameApkTask.map {
        …
    })
    🟩
}
```

为什么没有 dependsOn(…)?

Provider<T> - 自动化依赖处理

```
val notificationTaskProvider: TaskProvider<NotificationTask>
    = project.tasks.register(
    "notify${variantCapitalizedName}Build",
    NotificationTask::class.java
) {…}
```

```
val newProvider = renameApkTaskProvider.map {…}
```

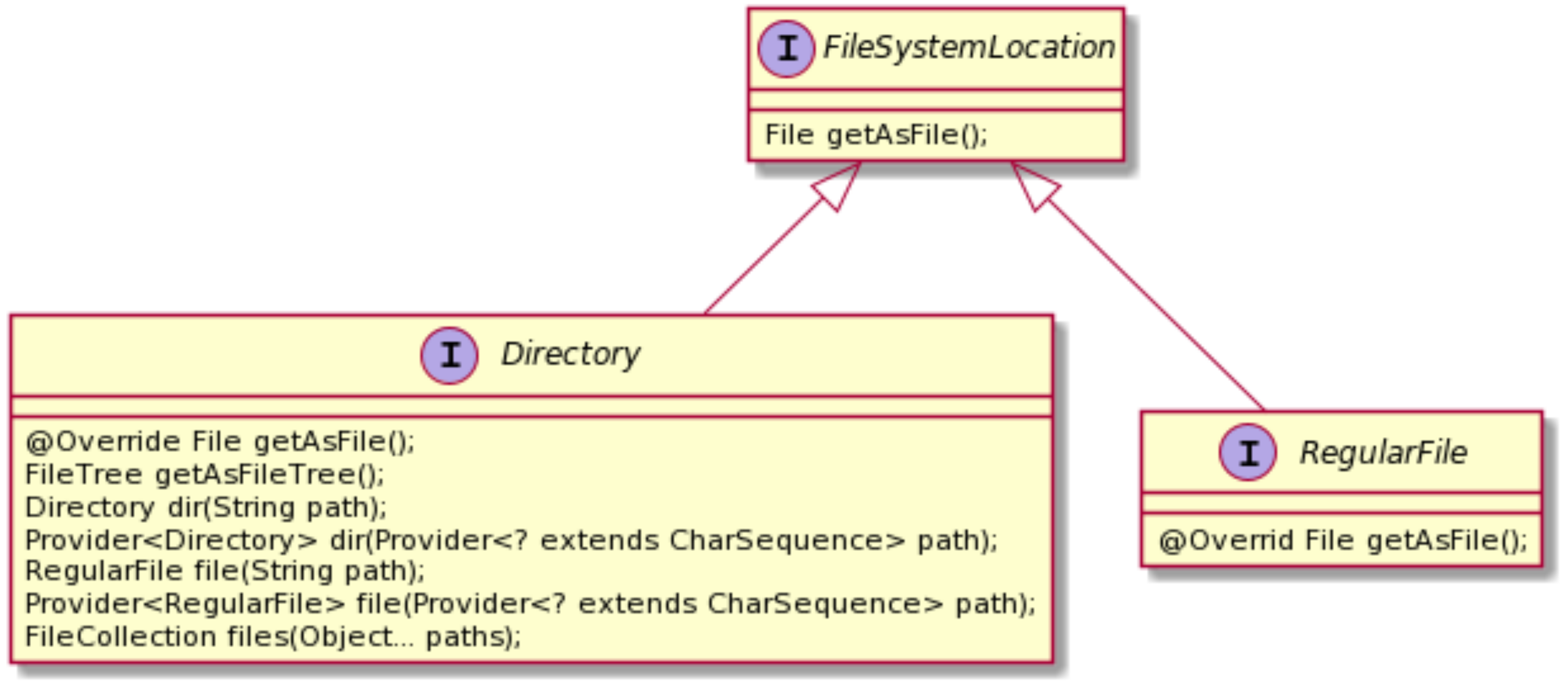
newProvider 会自动带上相关 Task 的依赖

Provider<T> 和文件

- ~~java.io.File~~

- FileTree
- FileCollection

- FileSystemLocation
- RegularFile
- Directory

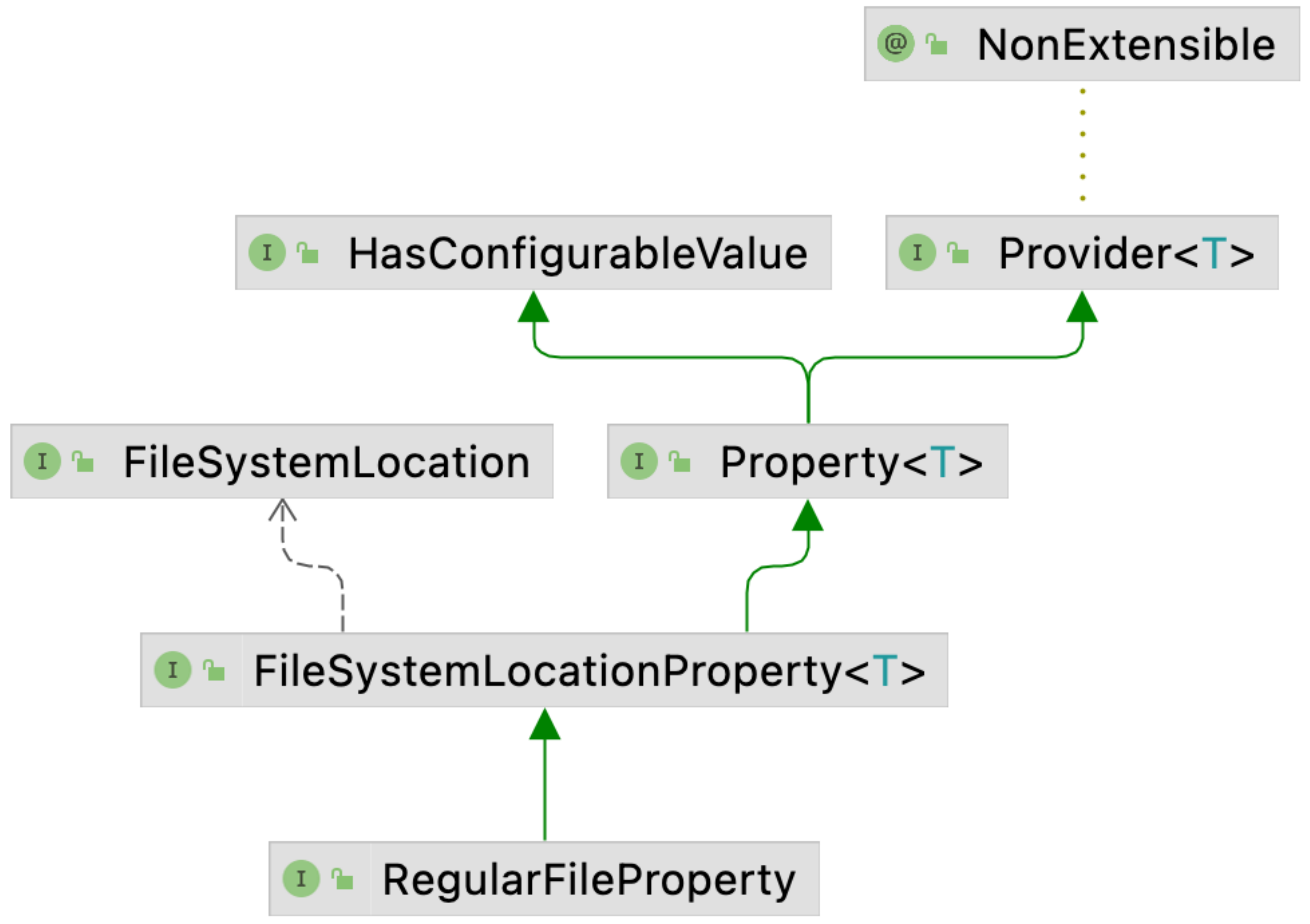


I FileSystemLocation
File getAsFile();
I Directory
@Override File getAsFile();
FileTree getAsFileTree();
Directory dir(String path);
Provider<Directory> dir(Provider<? extends CharSequence> path);
RegularFile file(String path);
Provider<RegularFile> file(Provider<? extends CharSequence> path);
FileCollection files(Object... paths);
I RegularFile
@Overrid File getAsFile();

Provider<T> 和文件

~~Property<RegularFile>~~

RegularFileProperty



```
interface RegularFileProperty extends FileSystemLocationProperty<RegularFile>
```

Provider<T> 和文件

```kotlin
abstract class ManifestAfterMergeTask : DefaultTask() {

    @get:InputFile
    abstract val mergedManifest: RegularFileProperty

    @get:OutputFile
    abstract val updatedManifest: RegularFileProperty

    @TaskAction
    fun afterMerge() {
        mergedManifest.get().asFile.copyTo(updatedManifest.get().asFile)
    }

}
```

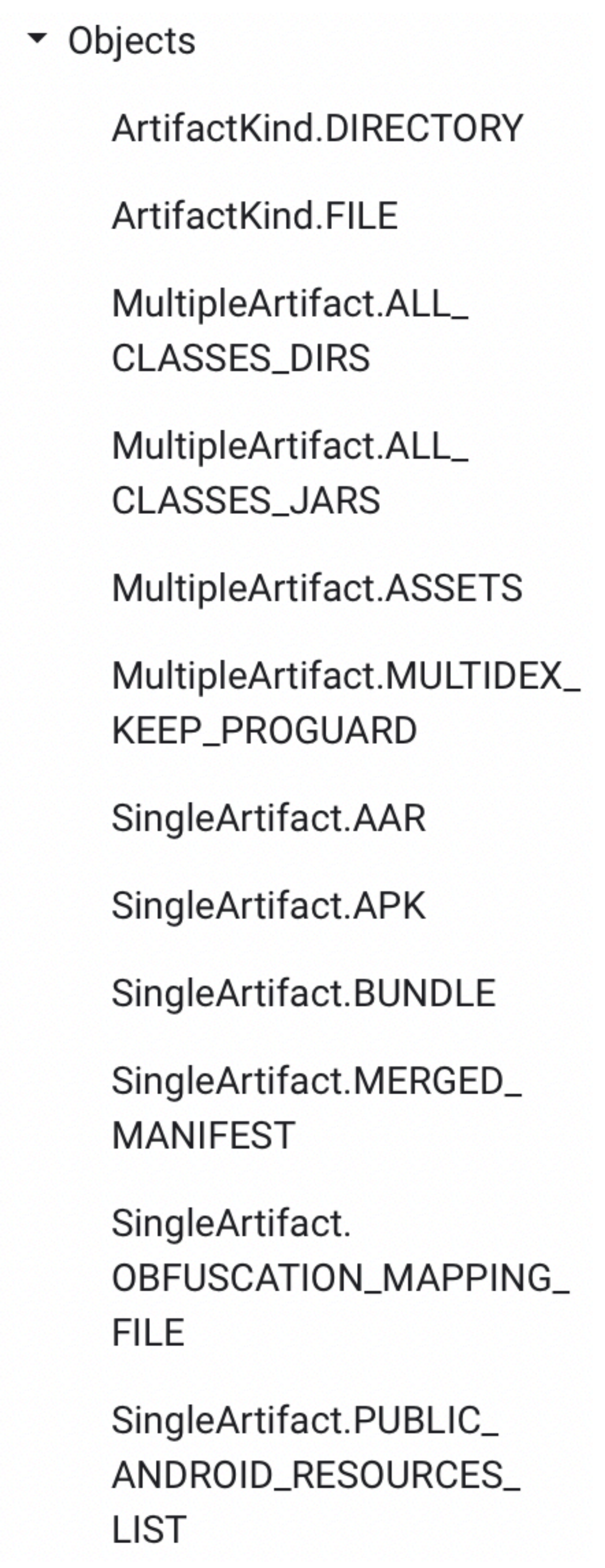
API v2 - 扩展的改进

1. 明确了可公开访问的 API
2. 基于 Provider<T> 接口的 CRUD Pipeline

目录

API v2 很美好，但是...

依旧有很多中间产物无法获取到...


▾ Objects

ArtifactKind.DIRECTORY

ArtifactKind.FILE

MultipleArtifact.ALL_CLASSES_DIRS

MultipleArtifact.ALL_CLASSES_JARS

MultipleArtifact.ASSETS

MultipleArtifact.MULTIDEX_KEEP_PROGUARD

SingleArtifact.AAR

SingleArtifact.APK

SingleArtifact.BUNDLE

SingleArtifact.MERGED_MANIFEST

SingleArtifact.OBFUSCATION_MAPPING_FILE

SingleArtifact.PUBLIC_ANDROID_RESOURCES_LIST

Google Developers

土制 Artifact API

Raw Gradle API

+

Hook

+

Provider<T>

土制 Artifact API - 获取合并前的 Manifests

```
val andExt = project.extensions.getByType(AndroidComponentsExtension::class.java)
andExt.onVariants { variant ->

    // 0. Get Polyfill instance with Project instance
    val polyfill = ApplicationVariantPolyfill(project, variant)

    // 1. Create & Config the hook task.
    val preUpdateTask = project.tasks.register(
        "preUpdate${variant.name.capitalize()}Manifest",
        ManifestBeforeMergeTask::class.java
    ) {
        val p = polyfill.newProvider(ManifestMergeInputProvider::class.java).obtain()
        beforeMergeInputs.set(p)
    }
    // 2. Add it with the action (which plays the role of entry for a hook).
    val beforeMergeAction = ManifestBeforeMergeAction(preUpdateTask)
    polyfill.addAGPTaskAction(beforeMergeAction)

}
```

1

2

土制 Artifact API - 获取合并前的 Manifests

```kotlin
class ManifestMergeInputProvider
    : ApplicationSelfManageableProvider<Provider<Set<FileSystemLocation>>> {
    private lateinit var manifests: Provider<Set<FileSystemLocation>>
    override fun initialize(...) {
        // ProcessApplicationManifest#configure(...)
        manifests = (variant as ApplicationVariantImpl).delegate
            .config
            .variantDependencies
            .getArtifactCollection(
                AndroidArtifacts.ConsumedConfigType.RUNTIME_CLASSPATH,
                AndroidArtifacts.ArtifactScope.ALL,
                AndroidArtifacts.ArtifactType.MANIFEST
            )
            .artifactFiles // FileCollection
            .elements
    }
    override fun obtain(defaultValue: Provider<Set<FileSystemLocation>>?)
        : Provider<Set<FileSystemLocation>> {
        return manifests
    }
}
```

土制 Artifact API - 获取合并前的 Manifests

```kotlin
class ManifestBeforeMergeAction(private val taskProvider: TaskProvider<*>) :
    ApplicationAGPTaskAction {

    override fun orchestrate(…) {
        // `variant.toTaskContainer().processManifestTask` can not guarantee the impl class
        project.afterEvaluate {
            project.tasks.named("process${variantCapitalizedName}MainManifest")
                .apply { configure { it.dependsOn(taskProvider) } }
        }
        project.rootProject.subprojects { subProject ->
            if (subProject == project) {
                return@subprojects
            }
            subProject.tasks.whenTaskAdded { newTask ->
                if (newTask.name == "process${variantCapitalizedName}Manifest"
                    || newTask.name == "extractDeepLinks${variantCapitalizedName}"
                ) {
                    taskProvider.configure { preUpdateTask ->
                        preUpdateTask.dependsOn(newTask)
                    }
                }
            }
        }
    }
}
```

土制 Artifact API - Polyfill

A middleware to assist writing Gradle Plugins for Android build system.

(https://github.com/2BAB/Polyfill)

土制 Artifact API - Seal

A Gradle Plugin to resolve AndroidManifest.xml merge conflicts.

(https://github.com/2BAB/Seal)

更多

Refs

- 本次分享的 Samples
- 扩展 Android Gradle 插件
- What's new in AGP 2021
- From Gradle properties to AGP APIs
- AGP API Ref
- Lazy Configuration
- Intro to Gradle and AGP Build APIs - MAD Skills

KOGE

面向 Kotlin 用户的
Gradle 基础手册

(https://koge.2bab.me/#/zh-cn/)

公众号

Android高效开发

(在菜单查看本次分享的 PPT)