EBook Gratis

APRENDIZAJE android-gradle

Free unaffiliated eBook created from
Stack Overflow contributors.

#android-gradle

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: android-gradle

It is an unofficial and free android-gradle ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official android-gradle.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con android-gradle

Observaciones

Que es android-gradle

`android-gradle` es un complemento de `gradle` mantenido oficialmente por el equipo de desarrolladores de Google Tools y es la herramienta de compilación oficial desde el anuncio el 16 de mayo de 2013 en Google I / O.

Aprende lo básico leyendo Configura tu construcción con Gradle .

Principales características

Las principales características del Android Gradle Plugin son:

- Gestión de la dependencia
- Proyectos modulares con bibliotecas.
- Variantes a través de sabores y tipos de construcción
- IDE construcciones independientes

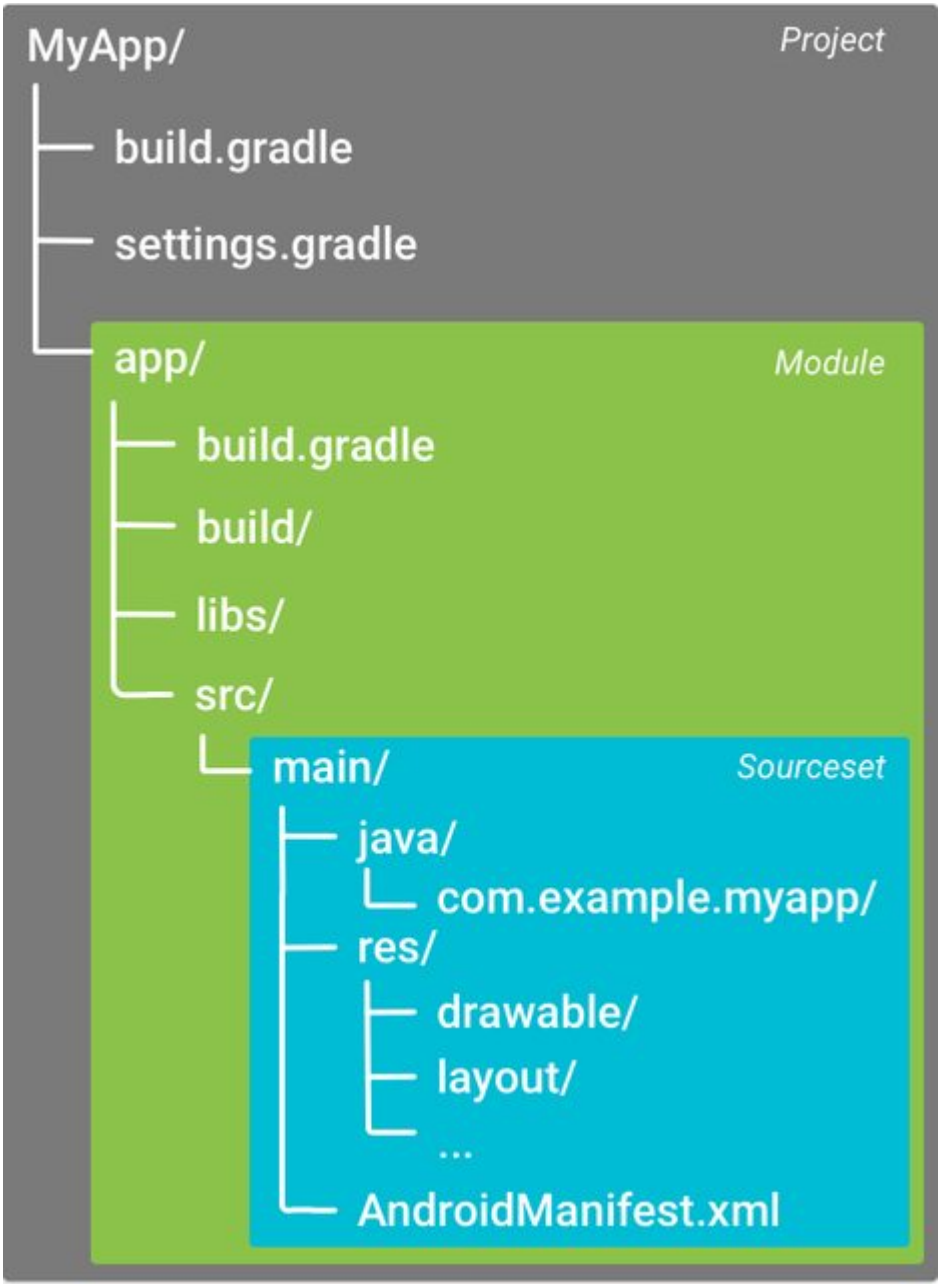
Visión general

1. Descarga e instala Android Studio
2. Ábrelo y crea un nuevo proyecto con todas las configuraciones predeterminadas.

   En teoría, puede instalar Gradle directamente, construir los archivos de configuración y la estructura de directorios por usted mismo. En la práctica nadie hace eso.

Estructura del proyecto

Una estructura de carpetas de proyectos normalmente se ve así:



plugin `android-gradle`

Un proyecto de Gradle generalmente se divide en subproyectos o *módulos,* cada uno de los cuales contiene un script de compilación dedicado.

La dependencia del complemento generalmente se declara en el archivo `build.gradle` nivel principal / superior:

```
buildscript {
    // maven repositories for dependencies
    repositories {
        jcenter()
    }
    // build script dependencies
    dependencies {
        // this is the dependency to the android build tools
        classpath 'com.android.tools.build:gradle:2.1.2'
```

```
    }
}

allprojects {
    // maven repositories for all sub-project / modules
    repositories {
        jcenter()
    }
}
```

En este ejemplo, la versión del plugin `android-gradle` es `2.1.2` como se puede ver en esta línea:

```
classpath 'com.android.tools.build:gradle:2.1.2'
```

Módulos

El proyecto se divide en *módulos,* cada uno de los cuales contiene una `build.gradle` comandos `build.gradle` dedicada. El archivo `settings.gradle` enumera estos módulos:

```
include ':app'
```

Los dos puntos `:` se utiliza algo como un delimitador de carpeta.

Para utilizar el complemento, debe aplicarse en la parte superior del archivo `build.gradle` de cada módulo (`app` en el ejemplo).

Para una aplicación de Android:

```
apply plugin: 'com.android.application'
```

Para una biblioteca de Android:

```
apply plugin: 'com.android.library'
```

Y luego configurado en su etiqueta de `android` :

```
android {
  // gradle-android plugin configuration
}
```

Configuración básica de la aplicación Android

El `build.gradle` generado por Android Studio para una aplicación se ve así:

```
apply plugin: 'com.android.application'
```

```
android {
    // setup which version of the SDK to build against and
    // with what version of the build tools
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    // default app configurations
    defaultConfig {
        // this is your app unique ID
        applicationId "com.example.myapp"

        // devices with lower SDK version can't install the app
        minSdkVersion 14
        // target SDK version, should be the last available one and
        // match the compile one
        targetSdkVersion 23

        // integer and string version of your app
        versionCode 1
        versionName "1.0"
    }

    // default build types are "debug" and "release"
    buildTypes {
        release {
            // enable / disable proguard optimization
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

// app dependencies
dependencies {
    // any jar file in the libs folder
    compile fileTree(dir: 'libs', include: ['*.jar'])
    // test dependency
    testCompile 'junit:junit:4.12'
    // runtime dependency on the support library
    compile 'com.android.support:appcompat-v7:24.0.0'
}
```

Configure su compilación con Gradle y aprenda las configuraciones y opciones más avanzadas de Android Gradle Plugin y profundice en el significado de esta configuración.

El `defaultConfig` se llama así porque se puede anular con Product Flavors .

La etiqueta `buildTypes` permite configurar cómo construir su aplicación habilitando la optimización (como proguard), puede aprender más a leer los tipos de compilación . También se puede utilizar para configurar la firma de su aplicación.

También debe aprender más sobre cómo declarar las dependencias . Como ves, la etiqueta de `dependencies` está fuera de la de `android` : esto significa que no está definida por el complemento de Android, pero es estándar de `gradle` .

El envoltorio de Gradle

Android Studio también instalará, de forma predeterminada, una envoltura de Gradle . Esta es una herramienta que puede ejecutar directamente desde la línea de comandos y descargará una versión local específica de gradle la primera vez que la ejecute.

Para iniciar la compilación de la aplicación, puede iniciar la envoltura de Gradle

Linux / Mac:

```
./gradlew assemble
```

Windows:

```
gradlew assemble
```

El script `gradle` el contenedor, contenido en una carpeta de `gradle` en el directorio raíz de su proyecto:

- `gradle-wrapper.jar` : el código del envoltorio para descargar gradle y ejecutarlo
- `gradle-wrapper.properties` define qué versión de gradle debe descargar el contenedor

---

Enlaces externos:

- Documentación oficial de Android Build Tools
- Documentación oficial de Android Gradle Plugin
- Documentación Gradle Stackoverflow
- Documentación oficial de gradle.

Examples

Configuración inicial con Android Studio

Para configurar el uso de Android Gradle Plugin necesitas muchas cosas:

- Java
- gradle
- la estructura de carpetas del proyecto Android
- un manifiesto de Android
- configuración inicial del plugin

La forma más fácil de obtenerlos es seguir estos pasos:

1. Descargue e instale Java OpenJDK versión 6 o 7 (puede usar 8 con configuraciones

adicionales del complemento gradle)
2. Descargar e instalar Android Studio
3. Cree un nuevo proyecto (si necesita ayuda, vea Crear un nuevo proyecto)

Consulte la sección de *comentarios* para más información.

Plugin de Android para Gradle

Como se describe en la sección de comentarios, el sistema de compilación de Android utiliza el complemento de Android para Gradle para admitir la creación de aplicaciones de Android con Gradle.

Puede especificar el complemento de Android para la versión Gradle en el archivo de nivel superior `build.gradle` . La versión del complemento se aplica a todos los módulos integrados en ese proyecto de Android Studio.

```
buildscript {
  ...
  dependencies {
    classpath 'com.android.tools.build:gradle:2.2.0'
  }
}
```

Envoltura Gradle

Como se describe en la sección de comentarios, puede especificar la versión de Gradle utilizada por cada proyecto que edite la referencia de distribución de Gradle en el `gradle/wrapper/gradle-wrapper.properties` .

Por ejemplo:

```
...
distributionUrl = https\://services.gradle.org/distributions/gradle-2.14.1-all.zip
...
```

Lea Empezando con android-gradle en línea: https://riptutorial.com/es/android-gradle/topic/2092/empezando-con-android-gradle

Capítulo 2: Cómo incluir archivos aar en un proyecto en Android

Examples

¿Cómo agregar la dependencia .aar en un módulo?

En un módulo (biblioteca o aplicación) donde necesita el archivo aar que tiene que agregar en su `build.gradle` el repositorio:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

y añadir la dependencia:

```
dependencies {
    compile(name:'nameOfYourAARFileWithoutExtension', ext:'aar')
}
```

Preste atención a la ruta relativa de la carpeta libs que está utilizando en el módulo.

El archivo aar no incluye las dependencias transitivas.

El archivo **aar no contiene las dependencias transitivas** y no tiene un archivo pom que describa las dependencias utilizadas por la biblioteca.

Esto significa que, si está importando un archivo aar utilizando un repositorio de `flatDir` **, tiene que especificar las dependencias también en su proyecto** .

Debe usar un **repositorio de maven** (debe publicar la biblioteca en un repositorio de maven privado o privado), no tendrá el mismo problema.
En este caso, gradle descarga las dependencias utilizando el archivo pom que contiene la lista de dependencias.

Esto funciona con las bibliotecas aar que se publican en un repositorio local o remoto de Maven.
En su caso, parece que la biblioteca no se publicará ni siquiera en un repositorio de maven local.
No puedo encontrar ninguna información definitiva sobre si funcionará en tus circunstancias, pero deberías intentarlo.

Lea Cómo incluir archivos aar en un proyecto en Android en línea:
https://riptutorial.com/es/android-gradle/topic/3037/como-incluir-archivos-aar-en-un-proyecto-en-android

Capítulo 3: Configurar ajustes de firma

Examples

Configurar el build.gradle con la configuración de firma

Puede definir la configuración de firma para firmar el apk en el archivo `build.gradle` .

Puedes definir:

- `storeFile` : el archivo de almacén de claves
- `storePassword` : la contraseña del almacén de claves
- `keyAlias` : un nombre de alias de clave
- `keyPassword` : una contraseña de alias de clave

Usted tiene que **definir** las `signingConfigs` bloquean para crear una configuración de firma:

```
android {
    signingConfigs {

        myConfig {
            storeFile file("myFile.keystore")
            storePassword "myPasswork"
            keyAlias "aKeyAlias"
            keyPassword "myAliasPassword"
        }
    }
    //....
}
```

Luego puedes **asignarlo** a uno o más tipos de compilación.

```
android {

    buildTypes {
        release {
            signingConfig signingConfigs.myConfig
        }
    }
}
```

Definir la configuración de firma en un archivo externo.

Puede definir la configuración de firma en un archivo externo como una `signing.properties` en el directorio raíz de su proyecto.

Por ejemplo, puede definir estas teclas (puede usar sus nombres favoritos):

```
STORE_FILE=myStoreFileLocation
STORE_PASSWORD=myStorePassword
```

```
KEY_ALIAS=myKeyAlias
KEY_PASSWORD=mykeyPassword
```

Luego en tu archivo build.gradle:

```
android {

    signingConfigs {
        release
    }

     buildTypes {
        release {
            signingConfig signingConfigs.release
        }
     }
}
```

Luego puede introducir algunos controles para evitar problemas de gradle en el proceso de construcción.

```
//--------------------------------------------------------------------------------
// Signing
//--------------------------------------------------------------------------------
def Properties props = new Properties()
def propFile = file('../signing.properties')
if (propFile.canRead()) {

    if (props != null && props.containsKey('STORE_FILE') &&
props.containsKey('STORE_PASSWORD') &&
            props.containsKey('KEY_ALIAS') && props.containsKey('KEY_PASSWORD')) {

        android.signingConfigs.release.storeFile = file(props['STORE_FILE'])
        android.signingConfigs.release.storePassword = props['STORE_PASSWORD']
        android.signingConfigs.release.keyAlias = props['KEY_ALIAS']
        android.signingConfigs.release.keyPassword = props['KEY_PASSWORD']
    } else {
        android.buildTypes.release.signingConfig = null
    }
} else {
    android.buildTypes.release.signingConfig = null
}
```

Definir las variables de entorno de configuración de firma.

Puede almacenar las variables de entorno de configuración de información de firma.
Se puede acceder a estos valores con `System.getenv("<VAR-NAME>")`

En tu `build.gradle` puedes definir:

```
signingConfigs {
    release {
        storeFile file(System.getenv("KEYSTORE"))
        storePassword System.getenv("KEYSTORE_PASSWORD")
        keyAlias System.getenv("KEY_ALIAS")
```

```
        keyPassword System.getenv("KEY_PASSWORD")
    }
}
```

Defina la configuración de firma en un archivo gradle separado

La forma más sencilla y limpia de agregar una configuración externa es a través de un archivo Gradle separado

construir.gradle

```
apply from: './keystore.gradle'
android{
    signingConfigs {
        release {
            storeFile file(keystore.storeFile)
            storePassword keystore.storePassword
            keyAlias keystore.keyAlias
            keyPassword keystore.keyPassword
        }
    }
}
```

keystore.gradle

```
ext.keystore = [
    storeFile    : "/path/to/your/file",
    storePassword: 'password of the store',
    keyAlias     : 'alias_of_the_key',
    keyPassword  : 'password_of_the_key'
]
```

El archivo keystore.gradle puede existir en cualquier parte de su sistema de archivos, puede especificar su ubicación dentro de la `apply from: ''` en la parte superior de su archivo de gradle o al final de su proyecto principal, el archivo build.gradle.

Normalmente, es una buena idea ignorar este archivo del sistema de control de versiones como git si está ubicado dentro de su repositorio.

También es una buena idea proporcionar un ejemplo de `keystore.gradle.sample` que los desarrolladores que ingresan al proyecto cambiarían de nombre y rellenarían en su máquina de desarrollo. Este archivo siempre estaría contenido dentro del repositorio en la ubicación correcta.

Lea Configurar ajustes de firma en línea: https://riptutorial.com/es/android-gradle/topic/5249/configurar-ajustes-de-firma

Capítulo 4: Configurar los sabores del producto

Observaciones

Los sabores de producto admiten las mismas propiedades que `defaultConfig` esto se debe a que defaultConfig realmente pertenece a la clase ProductFlavor. Esto significa que puede proporcionar la configuración básica para todos los sabores en el bloque `defaultConfig {}` , y cada sabor puede anular cualquiera de estos valores predeterminados, como el `pplicationId` .

Examples

Cómo configurar el archivo build.gradle

```
android {
    ...
    defaultConfig {...}
    buildTypes {...}
    productFlavors {
        demo {
            applicationId "com.example.myapp.demo"
            versionName "1.0-demo"
        }
        full {
            applicationId "com.example.myapp.full"
            versionName "1.0-full"
        }
    }
}
```

Constantes de Sabor y Recursos en build.gradle

Puede usar gradle para tener constantes de `BuildConfig` y valores de `res` por sabor. Simplemente agregue el valor al sabor que desea admitir.

```
android {
    defaultConfig {
        resValue "string", "app_name", "Full App"
        buildConfigField "boolean", "isDemo", "false"
    }
    productFlavors {
        demo {
            resValue "String", "app_name", "Demo App"
            buildConfigField "boolean", "isDemo", "true"
        }
        full {
            // use default values
        }
    }
}
```

Gradle hará toda la fusión / anulación por usted. El código generado también le permitirá ver de dónde provienen los valores, por ejemplo,

```
<!-- Values from default config. -->
<string name="app_name" translatable="false">Default Name</string>
```

y

```
public final class BuildConfig {
    public static final String VERSION_NAME = "1.0";
    // Fields from product flavor: demo
    public static final boolean isDemo = true;
}
```

Usando la Dimensión del Sabor

Cuando la aplicación se basa en más de un criterio, en lugar de crear muchos sabores, puede definir las dimensiones del sabor.

Las dimensiones del sabor definen el producto cartesiano que se utilizará para producir variantes.

Ejemplo:

```
flavorDimensions("dimA", "dimB")

productFlavors {

    row1 {
        ...
        dimension = "dimA"
    }
    row2 {
        ...
        dimension = "dimA"
    }
    row3 {
         ...
        dimension = "dimA"
    }

    col1 {
        ...
        dimension = "dimB"
    }
    col2 {
        ...
        dimension = "dimB"
    }
    col3 {
         ...
        dimension = "dimB"
    }
}
```

Esta configuración producirá 18 (3 *3* 2) variantes (si tiene los 2 tipos de compilación estándar:

`debug` y `release`). Se crearán las siguientes variantes de compilación:

```
row1-col1-debug
row1-col2-debug
row1-col3-debug
row1-col1-release
row1-col2-release
row1-col3-release

row2-col1-debug
row2-col2-debug
row2-col3-debug
row2-col1-release
row2-col2-release
row2-col3-release

row3-col1-debug
row3-col2-debug
row3-col3-debug
row3-col1-release
row3-col2-release
row3-col3-release
```

El **orden de la dimensión** está definido por `android.flavorDimensions` y las **unidades cuyo sabor reemplaza al otro** , lo que es importante para los recursos cuando un valor en un sabor reemplaza un valor definido en un sabor de prioridad más baja.

La dimensión de sabor se define con mayor prioridad primero. Así que en este caso:

```
dimA > dimB > defaultConfig
```

También hay una carpeta de origen de "combinación de sabor" disponible cuando se utiliza más de una dimensión de sabor. Por ejemplo `src/flavor1Flavor2/` .

- Tenga en cuenta que esto es para todas las combinaciones de todas las dimensiones.
- Su prioridad es más alta que los conjuntos de fuentes de un solo sabor, pero más baja que los tipos de compilación.

Añadir dependencias para los sabores.

Puede agregar diferentes dependencias para un sabor de producto específico.

Solo use la `<flavorName>Compile 'group:name:xyz'` :

```
android {
    ...
    productFlavors {
        flavor1 {
           //.....
        }
        flavor2 {
           //.....
        }
    }
```

```
}

...
dependencies {

    compile 'com.android.support:appcompat-v7:24.2.0'

    // Add a dependency only for flavor1
    flavor1Compile 'group:name:x.y.z'

    // Add a dependency only for flavor2
    flavor2Compile 'group:name:x.y.z'
}
```

Ejemplo de desarrollo y producción de sabores de productos

```
productFlavors {
        // Define separate dev and prod product flavors.
        dev {
            // dev utilizes minSDKVersion = 21 to allow the Android gradle plugin
            // to pre-dex each module and produce an APK that can be tested on
            // Android Lollipop without time consuming dex merging processes.
            minSdkVersion 21
        }
        prod {
            // The actual minSdkVersion for the application.
            minSdkVersion 15
        }
    }
```

Lea Configurar los sabores del producto en línea: https://riptutorial.com/es/android-gradle/topic/2929/configurar-los-sabores-del-producto

Capítulo 5: Configurar tipos de compilación

Parámetros

Parámetro	Detalle
aplicaciónIdSuffix	Sufijo de ID de aplicación aplicado a esta configuración base
consumerProguardFiles	Los archivos de reglas de ProGuard se incluirán en el AAR publicado.
debuggable	Si este tipo de compilación debería generar un apk debuggable
embedMicroApp	Si una aplicación de Android Wear vinculada debe estar integrada en la variante con este tipo de compilación
jniDebuggable	Si este tipo de compilación está configurado para generar un APK con código nativo que se puede depurar
manifestPlaceholders	Los marcadores de posición manifiestos.
minifyEnabled	Si Minify está habilitado para este tipo de compilación
MultiDexEnabled	Si Multi-Dex está habilitado para esta variante
nombre	Nombre de este tipo de construcción
archivos de proguard	Devuelve los archivos de configuración de ProGuard para ser utilizados
pseudolocales habilitado	Ya sea para generar pseudo locale en el APK
renderscriptDebuggable	Si el tipo de compilación está configurado para generar un apk con código RenderScript debuggable
renderscriptOptimLevel	Nivel de optimización a utilizar por el compilador de renderscript.
encogimientoRecursos	Si la reducción de los recursos no utilizados está habilitada. El valor predeterminado es falso
firmaConfig	La configuracion de firma
testCoverageEnabled	Si la cobertura de prueba está habilitada para este tipo de compilación

Parámetro	Detalle
versionNameSuffix	Sufijo del nombre de la versión
zipAlignEnabled	Si zipalign está habilitado para este tipo de compilación
------	--------
Método	**Detalle**
buildConfigField (tipo, nombre, valor)	Agrega un nuevo campo a la clase BuildConfig generada
consumerProguardFile (proguardFile)	Agrega un archivo de reglas de progreso que se incluirá en el AAR publicado
consumerProguardFiles (proguardFiles)	Agrega archivos de reglas de progreso que se incluirán en el AAR publicado
proguardFile (proguardFile)	Agrega un nuevo archivo de configuración de ProGuard
proguardFiles (proguardFiles)	Agrega nuevos archivos de configuración de ProGuard
valor (tipo, nombre, valor)	Agrega un nuevo recurso generado
valor (tipo, nombre, valor)	Agrega un nuevo recurso generado
setProguardFiles (proguardFileIterable)	Establece los archivos de configuración de ProGuard.
shrinkResources (flag)	Si la reducción de los recursos no utilizados está habilitada. El valor predeterminado es falso

Observaciones

De forma predeterminada, el complemento de Android para gradle configura automáticamente el proyecto para construir tanto una versión de depuración como una versión de lanzamiento de la aplicación.

Esta configuración se realiza a través de un objeto llamado `BuildType`

Documentación oficial:

http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html

Examples

Cómo configurar los tipos de compilación en el build.gradle

Puede crear y configurar tipos de compilación en el archivo `build.gradle` nivel de `build.gradle` dentro del bloque `android {}` .

```
    android {
        ...
        defaultConfig {...}

        buildTypes {
            release {
                minifyEnabled true
                proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
            }

            debug {
                applicationIdSuffix ".debug"
            }
        }
    }
```

Lea Configurar tipos de compilación en línea: https://riptutorial.com/es/android-gradle/topic/3281/configurar-tipos-de-compilacion

Capítulo 6: Configure su construcción con Gradle

Observaciones

El sistema de compilación de Android compila los recursos de la aplicación y el código fuente, y los empaqueta en archivos APK que puede probar, implementar, firmar y distribuir. Android Studio utiliza Gradle, un kit de herramientas de compilación avanzada, para automatizar y administrar el proceso de compilación, mientras que le permite definir configuraciones de compilación personalizadas y flexibles.

Documentacion oficial

https://developer.android.com/studio/build/index.html

Examples

¿Por qué hay dos archivos build.gradle en un proyecto de Android Studio?

`<PROJECT_ROOT>\app\build.gradle` es específico para el módulo de la aplicación.

`<PROJECT_ROOT>\build.gradle` es un "archivo de compilación de nivel superior" donde puede agregar opciones de configuración comunes a todos los subproyectos / módulos.

Si usa otro módulo en su proyecto, como biblioteca local tendrá otro archivo build.gradle: `<PROJECT_ROOT>\module\build.gradle`

El archivo de compilación de nivel superior

El archivo build.gradle de nivel superior, ubicado en el directorio del proyecto raíz, define configuraciones de compilación que se aplican a todos los módulos de su proyecto. De forma predeterminada, el archivo de compilación de nivel superior utiliza el `buildscript {} block` para definir los repositorios y dependencias de Gradle que son comunes a todos los módulos del proyecto. El siguiente ejemplo de código describe la configuración predeterminada y los elementos DSL que puede encontrar en el nivel superior de build.gradle después de crear un nuevo proyecto.

```
buildscript {
    repositories {
        mavenCentral()
    }

    dependencies {
       classpath 'com.android.tools.build:gradle:2.2.0'
       classpath 'com.google.gms:google-services:3.0.0'
    }
```

```
}

ext {
    compileSdkVersion = 23
    buildToolsVersion = "23.0.1"
}
```

El archivo de compilación de nivel de módulo

El archivo build.gradle a nivel de módulo, ubicado en cada directorio `<project>/<module>/` , le permite configurar las configuraciones de compilación para el módulo específico en el que se encuentra. La configuración de estas configuraciones de compilación le permite proporcionar opciones de empaquetado personalizadas, tales como como tipos de compilación y tipos de productos adicionales, y anular la configuración en el archivo `main/ app` manifiesto o en el archivo de nivel superior `build.gradle` .

```
apply plugin: 'com.android.application'


android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion
}

dependencies {
    //.....
}
```

Ejemplo de archivo de nivel superior

```
/**
 * The buildscript {} block is where you configure the repositories and
 * dependencies for Gradle itself--meaning, you should not include dependencies
 * for your modules here. For example, this block includes the Android plugin for
 * Gradle as a dependency because it provides the additional instructions Gradle
 * needs to build Android app modules.
 */

buildscript {

    /**
     * The repositories {} block configures the repositories Gradle uses to
     * search or download the dependencies. Gradle pre-configures support for remote
     * repositories such as JCenter, Maven Central, and Ivy. You can also use local
     * repositories or define your own remote repositories. The code below defines
     * JCenter as the repository Gradle should use to look for its dependencies.
     */

    repositories {
        jcenter()
    }

    /**
     * The dependencies {} block configures the dependencies Gradle needs to use
     * to build your project. The following line adds Android Plugin for Gradle
     * version 2.0.0 as a classpath dependency.
```

```
     */

    dependencies {
        classpath 'com.android.tools.build:gradle:2.0.0'
    }
}

/**
 * The allprojects {} block is where you configure the repositories and
 * dependencies used by all modules in your project, such as third-party plugins
 * or libraries. Dependencies that are not required by all the modules in the
 * project should be configured in module-level build.gradle files. For new
 * projects, Android Studio configures JCenter as the default repository, but it
 * does not configure any dependencies.
 */

allprojects {
   repositories {
       jcenter()
   }
}
```

El ejemplo de archivo de módulo

```
/**
 * The first line in the build configuration applies the Android plugin for
 * Gradle to this build and makes the android {} block available to specify
 * Android-specific build options.
 */

apply plugin: 'com.android.application'

/**
 * The android {} block is where you configure all your Android-specific
 * build options.
 */

android {

  /**
   * compileSdkVersion specifies the Android API level Gradle should use to
   * compile your app. This means your app can use the API features included in
   * this API level and lower.
   *
   * buildToolsVersion specifies the version of the SDK build tools, command-line
   * utilities, and compiler that Gradle should use to build your app. You need to
   * download the build tools using the SDK Manager.
   */

  compileSdkVersion 23
  buildToolsVersion "23.0.3"

  /**
   * The defaultConfig {} block encapsulates default settings and entries for all
   * build variants, and can override some attributes in main/AndroidManifest.xml
   * dynamically from the build system. You can configure product flavors to override
   * these values for different versions of your app.
   */
```

```
defaultConfig {

  /**
   * applicationId uniquely identifies the package for publishing.
   * However, your source code should still reference the package name
   * defined by the package attribute in the main/AndroidManifest.xml file.
   */

  applicationId 'com.example.myapp'

  // Defines the minimum API level required to run the app.
  minSdkVersion 14

  // Specifies the API level used to test the app.
  targetSdkVersion 23

  // Defines the version number of your app.
  versionCode 1

  // Defines a user-friendly version name for your app.
  versionName "1.0"
}

/**
 * The buildTypes {} block is where you can configure multiple build types.
 * By default, the build system defines two build types: debug and release. The
 * debug build type is not explicitly shown in the default build configuration,
 * but it includes debugging tools and is signed with the debug key. The release
 * build type applies Proguard settings and is not signed by default.
 */

buildTypes {

  /**
   * By default, Android Studio configures the release build type to enable code
   * shrinking, using minifyEnabled, and specifies the Proguard settings file.
   */

  release {
      minifyEnabled true // Enables code shrinking for the release build type.
      proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
  }
}

/**
 * The productFlavors {} block is where you can configure multiple product
 * flavors. This allows you to create different versions of your app that can
 * override defaultConfig {} with their own settings. Product flavors are
 * optional, and the build system does not create them by default. This example
 * creates a free and paid product flavor. Each product flavor then specifies
 * its own application ID, so that they can exist on the Google Play Store, or
 * an Android device, simultaneously.
 */

productFlavors {
  free {
    applicationId 'com.example.myapp.free'
  }

  paid {
    applicationId 'com.example.myapp.paid'
```

```
    }
  }
}

/**
 * The dependencies {} block in the module-level build configuration file
 * only specifies dependencies required to build the module itself.
 */

dependencies {
    compile project(":lib")
    compile 'com.android.support:appcompat-v7:24.1.0'
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

Usa archivesBaseName para cambiar el nombre de apk

Puedes usar el `archivesBaseName` para establecer el nombre de apk.

Por ejemplo:

```
  defaultConfig {
      ....
      project.ext.set("archivesBaseName", "MyName-" + defaultConfig.versionName);

  }
```

Obtendrás esta salida.

```
MyName-X.X.X-release.apk
```

Lea Configure su construcción con Gradle en línea: https://riptutorial.com/es/android-gradle/topic/2161/configure-su-construccion-con-gradle

Capítulo 7: Declarar dependencias

Examples

Cómo agregar dependencias

El siguiente ejemplo describe cómo declarar tres tipos diferentes de dependencias directas en el archivo `build.gradle` la aplicación / módulo:

```
   android {...}
    ...
    dependencies {
        // The 'compile' configuration tells Gradle to add the dependency to the
        // compilation classpath and include it in the final package.

        // Dependency on the "mylibrary" module from this project
        compile project(":mylibrary")

        // Remote binary dependency
        compile 'com.android.support:appcompat-v7:24.1.0'

        // Local binary dependency
        compile fileTree(dir: 'libs', include: ['*.jar'])
    }
```

Cómo agregar un repositorio

Para descargar dependencias, declare el repositorio para que Gradle pueda encontrarlas. Para hacer esto, agregue un `repositories { ... }` al `build.gradle` la aplicación / módulo en el archivo de nivel superior.

```
repositories {
  // Gradle's Java plugin allows the addition of these two repositories via method calls:
  jcenter()
  mavenCentral()

  maven { url "http://repository.of/dependency" }

  maven {
      credentials {
          username 'xxx'
          password 'xxx'
      }

  url 'http://my.maven
  }
}
```

Dependencias del módulo

En una `gradle build` proyectos `gradle build` , puede tener una dependencia con otro módulo en su

compilación.

Ejemplo:

```
  dependencies {
        // Dependency on the "mylibrary" module from this project
        compile project(":mylibrary")
  }
```

La línea del `compile project(':mylibrary')` declara un módulo de biblioteca local de Android llamado "mylibrary" como una dependencia, y requiere que el sistema de compilación compile e incluya el módulo local al compilar su aplicación.

Dependencias binarias locales

Puede tener una dependencia con un solo archivo jar o varios archivos jar.

Con un solo archivo jar puede agregar:

```
dependencies {
    compile files('libs/local_dependency.jar')
}
```

Es posible agregar un directorio de jars para compilar.

```
dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

La línea de compilación `fileTree(dir: 'libs', include: ['*.jar']`) le dice al sistema de compilación que incluya cualquier archivo JAR dentro del directorio `app/libs/` en el classpath de compilación y en el paquete final de su aplicación.

Si tiene módulos que requieren dependencias binarias locales, copie los archivos JAR para estas dependencias en `<moduleName>/libs` dentro de su proyecto.

Si necesita agregar un **archivo aar** , puede leer más detalles aquí.

Dependencias binarias remotas

Puede agregar dependencias remotas en Gradle utilizando esta estructura:

```
compile 'group:name:version'
```

o esta sintaxis alternativa:

```
compile group: 'xxx', name: 'xxxxx', version: 'xxxx'
```

Por ejemplo:

```
compile 'com.android.support:appcompat-v7:24.1.0'
```

La línea de `compile 'com.android.support:appcompat-v7:24.1.0 '` declara una dependencia de la versión 24.1.0 de la biblioteca de soporte de Android.

Declara Dependencias para Configuraciones

Se pueden agregar dependencias para configuraciones específicas como test / androidTest

```
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
testCompile 'junit:junit:3.8.1'
```

Alternativamente crea tu propia configuración

```
configurations {
    myconfig
}
```

Y luego descargar la dependencia para esta configuración.

```
myconfig group: 'com.mycompany', name: 'my_artifact', version: '1.0.0'
```

Declara dependencias para los sabores.

Se pueden agregar dependencias para sabores de productos específicos de manera similar a las configuraciones de compilación .

```
android {
    ...
    productFlavors {
        flavor1 {
            //...
        }
        flavor2 {
            //...
        }
    }
}

dependencies {
    flavor1Compile 'com.android.support:appcompat-v7:24.1.1'
    flavor1Compile 'com.google.firebase:firebase-crash:9.4.0'

    flavor2Compile 'com.android.support:appcompat-v7:24.1.1'
}
```

Declara las dependencias para los tipos de construcción

Se pueden agregar dependencias para tipos de compilación específicos:

```
android {
```

```
    ...
    buildTypes {
        release {
            //...
        }

        debug {
            //....
        }
    }
}

dependencies {
    debugCompile 'com.android.support:appcompat-v7:24.1.1'
    releaseCompile 'com.google.firebase:firebase-crash:9.4.0'
}
```

Lea Declarar dependencias en línea: https://riptutorial.com/es/android-gradle/topic/3289/declarar-dependencias

Capítulo 8: Gradle - Información de las etiquetas

Examples

Gradle - Información de las etiquetas

Gradle: se utiliza para compilar cualquier software, es un lenguaje específico del dominio que se usa para configurar y completar todos los complementos, las bibliotecas descargadas de los repositorios.

Utilizar complementos:

```
Apply plugin: 'com.android.application'
```

El complemento es propiedad en forma de valor clave. En la declaración anterior, el complemento denota la clave y la cadena del lado derecho en capas individuales se convierte en su valor.

Gradle es DSL (lenguaje específico del dominio):

Contiene diferentes `blocks:Tags`

```
repositories { }
dependencies {}
android {}
```

Los repositorios y las dependencias se utilizan para configurar los requisitos del código de la aplicación. El bloque de Android se utiliza para agregar código o información específica de Android a la aplicación. También generamos nuestras etiquetas personalizadas y definimos nuestro propio código personalizado, biblioteca e información.

Mediante el uso de la `"task" tag :`

```
task genrateTestDb (depends on: ….) {
 }
```

Archivos Gradle para cualquier aplicación.

`Build.gradle` archivo funciona para todos los proyectos. `Settings.gradle` - define todos los subdirectorios o proyectos incluidos en la aplicación.

`Build.gradle` contiene a continuación:

```
repositories {
mavenCentral()
}
```

La etiqueta de los repositorios de arriba contiene `mevenCentral()` significa que todas las dependencias se descargan de `mevenCentral()` También podemos usar `jcenter()` o cualquier otra fuente. El bloque de dependencias contiene todas las **dependencias de tiempo de compilación** que deben descargarse de los `repositories` .

```
dependencies {
compile 'org.codehous.groovy:groovy-all:2.3.2'
}
```

Arriba está la biblioteca `meven` : sintaxis:

`org.codehous.groovy` -> ID de grupo

`groovy-all` -> order fact id, ese es un nombre que Gradle utiliza para identificar la biblioteca.

`2.3.2'` -> versión

`Settings.gradle` : es una etiqueta de inclusión para todos los subproyectos que se agrega al proyecto.

```
Include 'googlechart', 'chuckgroovy'
```

Lea Gradle - Información de las etiquetas en línea: https://riptutorial.com/es/android-gradle/topic/9439/gradle---informacion-de-las-etiquetas

Capítulo 9: Reducir el código y los recursos

Observaciones

Para hacer que su archivo APK sea lo más pequeño posible, debe habilitar la reducción para eliminar el código y los recursos no utilizados en su versión de lanzamiento.

Examples

Reducir el código con ProGuard

Para habilitar la reducción de código con ProGuard, agregue `minifyEnabled true` al tipo de compilación apropiado en su archivo `build.gradle` .

```
android {
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile(‘proguard-android.txt'),
                    'proguard-rules.pro'
        }
    }
}
```

dónde:

- `minifyEnabled true` : habilita la reducción de código
- El `getDefaultProguardFile('proguard-android.txt')` obtiene la configuración predeterminada de ProGuard desde el SDK de Android
- El archivo `proguard-rules.pro` es donde puede agregar reglas personalizadas de ProGuard

Reducir los recursos

Para habilitar la reducción de recursos, establezca la propiedad `shrinkResources` en true en su archivo `build.gradle` .

```
android {
    ...

    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

Preste atención porque la reducción de recursos **solo funciona** junto con la reducción de código .

Puede personalizar qué recursos mantener o descartar creando un archivo XML como este:

```
<?xml version=1.0" encoding="utf-8"?>
<resources xmlns:tools="http://schemas.android.com/tools"
    tools:keep="@layout/mylayout,@layout/custom_*"
    tools:discard="@layout/unused" />
```

Guarde este archivo en `res/raw` carpeta `res/raw` .

Eliminar recursos alternativos no utilizados

Todas las bibliotecas vienen con recursos que no son necesarios para su aplicación. Por ejemplo, Google Play Services incluye traducciones para los idiomas que su propia aplicación ni siquiera admite.

Puede configurar el archivo build.gradle para especificar qué recurso desea conservar.
Por ejemplo:

```
defaultConfig {
    // ...

    resConfigs "en", "de", "it"
    resConfigs "nodpi", "xhdpi", "xxhdpi", "xxxhdpi"
}
```

Lea Reducir el código y los recursos en línea: https://riptutorial.com/es/android-gradle/topic/5257/reducir-el-codigo-y-los-recursos

Creditos

S. No	Capítulos	Contributors
1	Empezando con android-gradle	Community, Daniele Segato, Gabriele Mariotti
2	Cómo incluir archivos aar en un proyecto en Android	Gabriele Mariotti, JBirdVegas
3	Configurar ajustes de firma	DArkO, Gabriele Mariotti
4	Configurar los sabores del producto	David Medenjak, Gabriele Mariotti, piotrek1543, Tarek El-Mallah
5	Configurar tipos de compilación	Gabriele Mariotti
6	Configure su construcción con Gradle	Gabriele Mariotti
7	Declarar dependencias	4444, cricket_007, Gabriele Mariotti, jitinsharma
8	Gradle - Información de las etiquetas	Chetan Joshi
9	Reducir el código y los recursos	Gabriele Mariotti