eBook Gratuit

APPRENEZ android-gradle

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#android-gradle

Table des matières

À propos

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

Chapitre 1: Commencer avec android-gradle

Remarques

Qu'est-ce qu'android-gradle

`android-gradle` est un `gradle` in `gradle` officiellement mis à jour par l'équipe de développement de Google Tools et constitue l'outil de création officiel depuis l'annonce du 16 mai 2013 sur les E / S Google.

Apprenez les bases en lisant Configurer votre build avec Gradle .

Caractéristiques principales

Les principales caractéristiques du plug-in Android Gradle sont les suivantes:

- Gestion des dépendances
- Projets modulaires avec bibliothèques
- Variantes à travers les saveurs et les types de construction
- Constructions indépendantes IDE

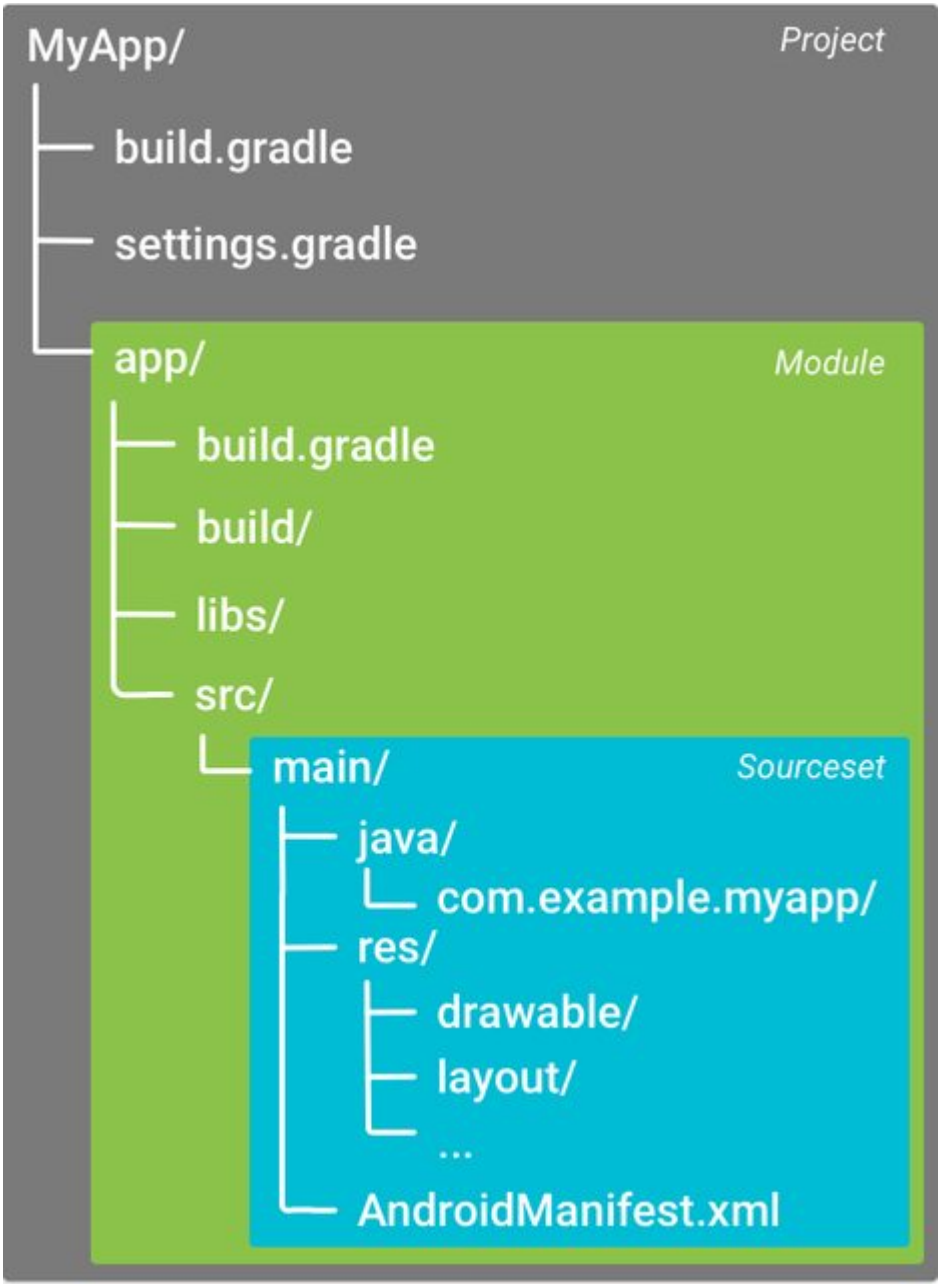
Vue d'ensemble

1. Téléchargez et installez Android Studio
2. l'ouvrir et créer un nouveau projet avec tous les paramètres par défaut

   En théorie, vous pouvez installer directement gradle, créer vous-même les fichiers de configuration et la structure des répertoires. En pratique, personne ne le fait.

Structure du projet

Une structure de dossier de projet ressemble généralement à ceci:



`android-gradle` plugin

Un projet de graduation est généralement divisé en sous-projets ou en *modules* contenant chacun un script de construction dédié.

La dépendance du plugin est généralement déclarée dans le fichier `build.gradle` principal / de niveau `build.gradle` :

```
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()
    }
}
```

Dans cet exemple, la version du plugin `android-gradle` est `2.1.2` comme vous pouvez le voir sur cette ligne:

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

Modules

Le projet est divisé en *modules* contenant chacun un script `build.gradle` dédié. Le fichier `settings.gradle` liste ces modules:

```
include ':app'
```

Les deux points `:` sont utilisés comme un délimiteur de dossier.

Pour utiliser le plug-in, il doit être appliqué en haut du fichier `build.gradle` de chaque module (`app` dans l'exemple).

Pour une application Android:

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

Pour une bibliothèque Android:

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

Et puis configuré dans sa balise `android` :

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

Configuration de base de l'application Android

Le `build.gradle` généré par Android Studio pour une application ressemble à ceci:

```
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'
}
```

Configurez votre build avec Gradle pour vous familiariser avec les paramètres et options avancés du plug-in Android Gradle et approfondir la signification de ce paramètre.

Le `defaultConfig` est appelé comme cela, car il peut être remplacé par des arômes de produit .

La balise `buildTypes` vous permet de configurer la manière de créer votre application permettant une optimisation (comme proguard), vous pouvez en apprendre davantage sur la lecture des types de construction . Il peut également être utilisé pour configurer la signature de votre application.

Vous devriez également en apprendre davantage sur la façon de déclarer les dépendances . Comme vous le voyez les `dependencies` balise est en dehors de l' `android` un: cela signifie qu'il est pas défini par le plugin Android , mais il est la norme `gradle` .

Le Gradle Wrapper

Android Studio installera également par défaut un wrapper gradle . C'est un outil que vous pouvez exécuter directement à partir de la ligne de commande et il va télécharger une version spécifique de gradle la première fois que vous l'exécutez.

Pour lancer la compilation de l'application, vous pouvez ensuite lancer le wrapper gradle

Linux / Mac:

```
./gradlew assemble
```

Les fenêtres:

```
gradlew assemble
```

Le script lance le wrapper, contenu dans un dossier de `gradle` dans le répertoire racine de votre projet:

- `gradle-wrapper.jar` : le code du wrapper pour télécharger gradle et l'exécuter
- `gradle-wrapper.properties` définit quelle version de dégradé le wrapper doit télécharger

---

Liens externes:

- Documentation officielle sur Android Build Tools
- Documentation officielle du plugin Android Gradle
- Stackoverflow gradle documentation
- Documentation officielle

Examples

Configuration initiale avec Android Studio

Pour configurer l'utilisation de Android Gradle Plugin, vous avez besoin de beaucoup de choses:

- Java
- gradle
- la structure du dossier de projet Android
- un manifeste Android
- configuration initiale du plugin

La manière la plus simple de tous les obtenir est de suivre ces étapes:

1. Télécharger et installer Java OpenJDK version 6 ou 7 (vous pouvez utiliser 8 avec des

paramètres supplémentaires du plug-in Gradle)
2. Téléchargez et installez Android Studio
3. Créer un nouveau projet (si vous avez besoin d'aide, voir Création d'un nouveau projet)

Vérifiez la section *Remarques* pour plus d'informations.

Plugin Android pour Gradle

Comme décrit dans la section Remarques, le système de génération Android utilise le plug-in Android pour Gradle pour prendre en charge la création d'applications Android avec Gradle.

Vous pouvez spécifier le plug-in Android pour la version Gradle dans le fichier `build.gradle` niveau `build.gradle` . La version du plugin s'applique à tous les modules construits dans ce projet Android Studio.

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

Gradle wrapper

Comme décrit dans la section Remarques, vous pouvez spécifier la version Gradle utilisée par chaque projet pour éditer la référence de distribution Gradle dans le `gradle/wrapper/gradle-wrapper.properties` .

Par exemple:

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

Lire Commencer avec android-gradle en ligne: https://riptutorial.com/fr/android-gradle/topic/2092/commencer-avec-android-gradle

Chapitre 2: Code rétractable et ressources

Remarques

Pour rendre votre fichier APK aussi petit que possible, vous devez activer la réduction pour supprimer le code et les ressources inutilisés dans votre version.

Examples

Réduire le code avec ProGuard

Pour activer la réduction du code avec ProGuard, ajoutez `minifyEnabled` au type de génération approprié dans votre fichier `build.gradle` .

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

où:

- `minifyEnabled true` : Activer le code réduit
- La `getDefaultProguardFile('proguard-android.txt')` obtient les paramètres ProGuard par défaut du SDK Android
- Le fichier `proguard-rules.pro` est l'endroit où vous pouvez ajouter des règles ProGuard personnalisées.

Réduire les ressources

Pour activer la réduction des ressources, définissez la propriété `shrinkResources` sur true dans votre fichier `build.gradle` .

```
android {
    ...

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

Faites attention car la réduction des ressources **ne fonctionne que si vous** réduisez le code .

Vous pouvez personnaliser les ressources à conserver ou à supprimer en créant un fichier XML comme celui-ci:

```
<?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" />
```

Enregistrez ce fichier dans le dossier `res/raw` .

Supprimer les ressources alternatives inutilisées

Toutes les bibliothèques sont fournies avec des ressources qui ne sont pas nécessairement utiles pour votre application. Par exemple, les services Google Play sont fournis avec des traductions pour les langues que votre propre application ne prend même pas en charge.

Vous pouvez configurer le fichier build.gradle pour spécifier la ressource que vous souhaitez conserver.
Par exemple:

```
defaultConfig {
    // ...

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

Lire Code rétractable et ressources en ligne: https://riptutorial.com/fr/android-gradle/topic/5257/code-retractable-et-ressources

Chapitre 3: Comment inclure des fichiers AAR dans un projet sous Android

Examples

Comment ajouter une dépendance .aar dans un module?

Dans un module (bibliothèque ou application) où vous avez besoin du fichier aar, vous devez ajouter dans votre `build.gradle` le référentiel:

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

et ajoutez la dépendance:

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

Faites attention au chemin relatif du dossier libs que vous utilisez dans le module.

Le fichier aar n'inclut pas les dépendances transitives

Le fichier **aar ne contient pas les dépendances transitives** et ne contient pas de fichier pom décrivant les dépendances utilisées par la bibliothèque.

Cela signifie que si vous importez un fichier aar à l'aide d'un repo `flatDir` **vous devez également spécifier les dépendances dans votre projet** .

Vous devriez utiliser un **dépôt maven** (vous devez publier la bibliothèque dans un **dépôt** privé ou public), vous n'aurez pas le même problème.
Dans ce cas, gradle télécharge les dépendances à l'aide du fichier pom qui contiendra la liste des dépendances.

Cela fonctionne avec les bibliothèques AAR qui sont publiées dans un référentiel Maven distant ou local. Dans votre cas, il semble que la bibliothèque ne sera pas publiée dans un référentiel Maven local. Je ne trouve aucune information définitive quant à savoir si cela fonctionnera dans vos circonstances, mais vous devriez essayer.

Lire Comment inclure des fichiers AAR dans un projet sous Android en ligne: https://riptutorial.com/fr/android-gradle/topic/3037/comment-inclure-des-fichiers-aar-dans-un-projet-sous-android

Chapitre 4: Configurer les paramètres de signature

Examples

Configurez le build.gradle avec la configuration de signature

Vous pouvez définir la configuration de signature pour signer le fichier `build.gradle` dans le fichier `build.gradle` .

Vous pouvez définir:

- `storeFile` : le fichier de `storeFile`
- `storePassword` : le mot de passe du `storePassword`
- `keyAlias` : un nom d'alias de clé
- `keyPassword` : un mot de passe alias de clé

Vous devez **définir** le bloc `signingConfigs` pour créer une configuration de signature:

```
android {
    signingConfigs {

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

Vous pouvez ensuite l' **assigner** à un ou plusieurs types de construction.

```
android {

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

Définir la configuration de signature dans un fichier externe

Vous pouvez définir la configuration de signature dans un fichier externe en tant que `signing.properties` dans le répertoire racine de votre projet.

Par exemple, vous pouvez définir ces clés (vous pouvez utiliser vos noms préférés):

```
STORE_FILE=myStoreFileLocation
STORE_PASSWORD=myStorePassword
KEY_ALIAS=myKeyAlias
KEY_PASSWORD=mykeyPassword
```

Ensuite, dans votre fichier build.gradle:

```
android {

    signingConfigs {
        release
    }

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

Ensuite, vous pouvez introduire des vérifications pour éviter les problèmes de graduation dans le processus de génération.

```
//--------------------------------------------------------------------------------
// 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
}
```

Définir les variables d'environnement du paramètre de configuration de signature

Vous pouvez stocker les variables d'environnement du paramètre d'informations de signature. Ces valeurs sont accessibles avec `System.getenv("<VAR-NAME>")`

Dans votre `build.gradle` vous pouvez définir:

```
signingConfigs {
    release {
```

```
        storeFile file(System.getenv("KEYSTORE"))
        storePassword System.getenv("KEYSTORE_PASSWORD")
        keyAlias System.getenv("KEY_ALIAS")
        keyPassword System.getenv("KEY_PASSWORD")
    }
}
```

Définir la configuration de signature dans un fichier distinct

La manière la plus simple et la plus propre d'ajouter une configuration externe consiste à utiliser un fichier Gradle distinct.

build.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'
]
```

Le fichier keystore.gradle peut exister n'importe où dans votre système de fichiers, vous pouvez spécifier son emplacement dans le champ Apply `apply from: ''` en haut de votre fichier de graduation ou à la fin du fichier build.gradle du projet principal.

En règle générale, c'est une bonne idée d'ignorer ce fichier à partir du système de contrôle de version tel que git s'il est situé dans votre dépôt.

Il est également judicieux de fournir un exemple de `keystore.gradle.sample` que les développeurs entrant dans le projet renommeraient et rempliraient leur machine de développement. Ce fichier serait toujours contenu dans le référentiel au bon endroit.

Lire Configurer les paramètres de signature en ligne: https://riptutorial.com/fr/android-gradle/topic/5249/configurer-les-parametres-de-signature

Chapitre 5: Configurer les saveurs du produit

Remarques

Les caractéristiques du produit prennent en charge les mêmes propriétés que `defaultConfig` car defaultConfig appartient en fait à la classe ProductFlavor. Cela signifie que vous pouvez fournir la configuration de base pour toutes les `defaultConfig {}` bloc `defaultConfig {}` , et que chaque version peut remplacer l'une de ces valeurs par défaut, par exemple un `pplicationId` .

Examples

Comment configurer le fichier 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 saveur et ressources dans build.gradle

Vous pouvez utiliser gradle pour avoir des constantes `BuildConfig` et des valeurs `res` sur une base par saveur. Ajoutez simplement la valeur à la saveur que vous souhaitez prendre en charge.

```
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 fera toutes les fusions / remplacements pour vous. Le code généré vous permettra

également de voir d'où proviennent les valeurs, par exemple

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

et

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

Utilisation de la dimension saveur

Lorsque l'application est basée sur plusieurs critères, au lieu de créer beaucoup de saveurs, vous pouvez définir des dimensions de saveur.

Les dimensions d'arôme définissent le produit cartésien qui sera utilisé pour produire des variantes.

Exemple:

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

productFlavors {

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

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

Cette configuration produira 18 (3 *3* 2) variantes (si vous avez les 2 types de build standard: `debug`

et `release`). Les variantes de construction suivantes seront créées:

```
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
```

L' **ordre de la dimension** est défini par `android.flavorDimensions` et les **lecteurs dont l'arôme remplace l'autre** , ce qui est important pour les ressources lorsqu'une valeur dans une saveur remplace une valeur définie dans une saveur de priorité inférieure.

La dimension de saveur est définie avec une priorité plus élevée en premier. Donc dans ce cas:

```
dimA > dimB > defaultConfig
```

Il existe également un dossier source "Combinaison de saveurs" disponible lorsque plusieurs dimensions de saveur sont utilisées. Par exemple `src/flavor1Flavor2/` .

- Notez que ceci est pour toutes les combinaisons de toutes les dimensions.
- Sa priorité est supérieure à celle des sources à saveur unique, mais inférieure à celle des types de construction.

Ajouter des dépendances pour les saveurs

Vous pouvez ajouter différentes dépendances pour un produit spécifique.

Utilisez simplement le groupe `<flavorName>Compile 'group:name:xyz'` syntaxe:

```
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'
}
```

Exemple de développement et production d'arômes de produit

```
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
        }
    }
```

Lire Configurer les saveurs du produit en ligne: https://riptutorial.com/fr/android-gradle/topic/2929/configurer-les-saveurs-du-produit

Chapitre 6: Configurer les types de construction

Paramètres

Paramètre	Détail
applicationIdSuffix	Suffixe de l'ID d'application appliqué à cette configuration de base
ConsumerProguardFiles	Fichiers de règles ProGuard à inclure dans l'AAR publié
débogueur	Si ce type de construction doit générer un fichier apk pouvant être débogué
EmbedMicroApp	Si une application Android Wear liée doit être intégrée dans variant en utilisant ce type de build
jniDebuggable	Si ce type de génération est configuré pour générer un fichier APK avec du code natif débogeable
manifestPlaceholders	Les espaces réservés manifest
minifyEnabled	Si Minify est activé pour ce type de build
multiDexEnabled	Si Multi-Dex est activé pour cette variante
prénom	Nom de ce type de build
proguardFiles	Retourne les fichiers de configuration ProGuard à utiliser
pseudoLocalesEnabled	Indique s'il faut générer des pseudo-paramètres régionaux dans l'APK
RenderscriptDabuggable	Indique si le type de génération est configuré pour générer un apk avec du code RenderScript pouvant être débogué
renderscriptOptimLevel	Niveau d'optimisation à utiliser par le compilateur renderscript
réduire les ressources	Si la réduction des ressources inutilisées est activée. Le défaut est faux
signatureConfig	La configuration de signature
testCoverageEnabled	Indique si la couverture de test est activée pour ce type de construction

Paramètre	Détail
versionNameSuffix	Suffixe du nom de la version
zipAlignEnabled	Si zipalign est activé pour ce type de construction
------	--------
Méthode	**Détail**
buildConfigField (type, nom, valeur)	Ajoute un nouveau champ à la classe BuildConfig générée
consumerProguardFile (proguardFile)	Ajoute un fichier de règles proguard à inclure dans l'AAR publié
consumerProguardFiles (proguardFiles)	Ajoute les fichiers de règles proguard à inclure dans l'AAR publié
proguardFile (proguardFile)	Ajoute un nouveau fichier de configuration ProGuard
proguardFiles (proguardFiles)	Ajout de nouveaux fichiers de configuration ProGuard
resValue (type, nom, valeur)	Ajoute une nouvelle ressource générée
resValue (type, nom, valeur)	Ajoute une nouvelle ressource générée
setProguardFiles (proguardFileIterable)	Définit les fichiers de configuration ProGuard
shrinkResources (indicateur)	Si la réduction des ressources inutilisées est activée. Le défaut est faux

Remarques

Par défaut, le plug-in Android pour gradle configure automatiquement le projet pour créer une version de débogage et une version finale de l'application.

Cette configuration se fait via un objet appelé `BuildType`

Documentation officielle:

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

Examples

Comment configurer les types de build dans le build.gradle

Vous pouvez créer et configurer des types de construction dans le fichier `build.gradle` niveau du module `build.gradle` dans le bloc `android {}` .

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

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

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

Lire Configurer les types de construction en ligne: https://riptutorial.com/fr/android-gradle/topic/3281/configurer-les-types-de-construction

Chapitre 7: Configurez votre build avec Gradle

Remarques

Le système de génération Android compile les ressources et le code source de l'application et les empaquette dans des fichiers APK que vous pouvez tester, déployer, signer et distribuer. Android Studio utilise Gradle, un kit d'outils de génération avancé, pour automatiser et gérer le processus de génération, tout en vous permettant de définir des configurations de construction personnalisées et flexibles.

Documentation officielle

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

Examples

Pourquoi y a-t-il deux fichiers build.gradle dans un projet Android Studio?

`<PROJECT_ROOT>\app\build.gradle` est spécifique au module d'application.

`<PROJECT_ROOT>\build.gradle` est un "fichier de construction de premier niveau" dans lequel vous pouvez ajouter des options de configuration communes à tous les sous-projets / modules.

Si vous utilisez un autre module dans votre projet, en tant que bibliothèque locale, vous auriez un autre fichier build.gradle: `<PROJECT_ROOT>\module\build.gradle`

Le fichier de construction de niveau supérieur

Le fichier build.gradle de niveau supérieur, situé dans le répertoire du projet racine, définit les configurations de génération qui s'appliquent à tous les modules de votre projet. Par défaut, le fichier de génération de niveau supérieur utilise le `buildscript {} block` pour définir les référentiels et les dépendances Gradle communs à tous les modules du projet. L'exemple de code suivant décrit les paramètres par défaut et les éléments DSL que vous pouvez trouver dans le fichier build.gradle de niveau supérieur après la création d'un nouveau projet.

```
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"
}
```

Le fichier de construction au niveau du module

Le fichier build.gradle au niveau du module, situé dans chaque répertoire `<project>/<module>/` , vous permet de configurer les paramètres de construction du module spécifique dans lequel il se trouve. La configuration de ces paramètres de construction vous permet de en tant que types de construction et variantes de produit supplémentaires, et remplacer les paramètres du manifeste `main/ app` ou du fichier `build.gradle` niveau `build.gradle` .

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


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

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

Exemple de fichier de niveau supérieur

```
/**
 * 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()
   }
}
```

L'exemple de fichier de module

```
/**
 * 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'])
}
```

Utilisez archivesBaseName pour changer le nom apk

Vous pouvez utiliser le `archivesBaseName` pour définir le nom de apk.

Par exemple:

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

  }
```

Vous obtiendrez cette sortie.

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

Lire Configurez votre build avec Gradle en ligne: https://riptutorial.com/fr/android-gradle/topic/2161/configurez-votre-build-avec-gradle

Chapitre 8: Déclarer les dépendances

Examples

Comment ajouter des dépendances

L'exemple ci-dessous décrit comment déclarer trois types de dépendances directes dans le fichier `build.gradle` l'application / du module:

```
   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'])
    }
```

Comment ajouter un référentiel

Pour télécharger des dépendances, déclarez le référentiel afin que Gradle puisse les trouver. Pour ce faire, ajoutez un `repositories { ... }` à la `build.gradle` de l'application / module dans le fichier de niveau supérieur.

```
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
  }
}
```

Dépendances des modules

Dans une version `gradle build` plusieurs projets, vous pouvez avoir une dépendance avec un

autre module de votre génération.

Exemple:

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

La ligne du `compile project(':mylibrary')` déclare un module de bibliothèque Android local nommé "mylibrary" comme une dépendance et requiert que le système de compilation compile et inclue le module local lors de la création de votre application.

Dépendances binaires locales

Vous pouvez avoir une dépendance avec un seul fichier JAR ou plusieurs fichiers JAR.

Avec un seul fichier jar, vous pouvez ajouter:

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

Il est possible d'ajouter un répertoire de jars à compiler.

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

La ligne compile `fileTree(dir: 'libs', include: ['*.jar']`) indique au système de compilation d'inclure tous les fichiers JAR dans le répertoire `app/libs/` dans le classpath de compilation et dans le package final de votre application.

Si vous avez des modules qui nécessitent des dépendances binaires locales, copiez les fichiers JAR correspondant à ces dépendances dans `<moduleName>/libs` intérieur de votre projet.

Si vous devez ajouter un **fichier aar,** vous pouvez lire plus de détails ici.

Dépendances binaires distantes

Vous pouvez ajouter des dépendances à distance dans Gradle en utilisant cette structure:

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

ou cette syntaxe alternative:

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

Par exemple:

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

La ligne de `compile 'com.android.support:appcompat-v7:24.1.0 '` déclare une dépendance à la version 24.1.0 de la bibliothèque de support Android.

Déclarer les dépendances pour les configurations

Des dépendances peuvent être ajoutées pour une configuration spécifique comme test / androidTest

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

Sinon, créez votre propre configuration

```
configurations {
    myconfig
}
```

Et puis téléchargez la dépendance pour cette configuration

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

Déclarez les dépendances pour les saveurs

Des dépendances peuvent être ajoutées pour des variantes de produits spécifiques de la même manière que les configurations de construction .

```
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'
}
```

Déclarez les dépendances pour les types de construction

Des dépendances peuvent être ajoutées pour des types de build spécifiques:

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

        debug {
            //....
        }
    }
}

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

Lire Déclarer les dépendances en ligne: https://riptutorial.com/fr/android-gradle/topic/3289/declarer-les-dependances

Chapitre 9: Gradle - Informations sur les balises

Examples

Gradle - Informations sur les balises

Gradle: Il est utilisé pour créer des logiciels pour n'importe quel logiciel, c'est un langage spécifique à un domaine utilisé pour configurer et exécuter tous les plugins, bibliothèques téléchargées à partir de référentiels.

Utilisez les plugins:

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

Le plugin est une propriété sous forme de valeur clé. Dans les instructions ci-dessus, le plugin indique la clé et la chaîne du côté droit en une seule couche devient sa valeur.

Gradle est DSL (langage spécifique au domaine):

Il contient différents `blocks:Tags`

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

Les référentiels et les dépendances permettent de configurer les exigences pour le code de l'application. Le bloc Android est utilisé pour ajouter un code ou une information spécifique à Android dans l'application. Nous générons également nos balises personnalisées et définissons notre propre code, bibliothèque et information personnalisés.

En utilisant la `"task" tag :`

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

Fichiers Gradle pour n'importe quelle application

`Build.gradle` - Ce fichier fonctionne pour tous les projets. `Settings.gradle` - définir tous les sous répertoires ou projets sont inclus dans l'application.

`Build.gradle` contient ci-dessous:

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

`mevenCentral()` signifie que toutes les dépendances sont téléchargées à partir de `mevenCentral()` Nous pouvons également utiliser `jcenter()` ou toute autre source. Le bloc de dépendances contient toutes les **dépendances de compilation** à télécharger depuis les `repositories` .

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

Ci- `meven` bibliothèque `meven` : syntaxe:

`org.codehous.groovy` -> identifiant du groupe

`groovy-all` -> ID de fait de commande, c'est un nom utilisé pour identifier la bibliothèque.

`2.3.2'` -> version

`Settings.gradle` - il contient une balise pour tous les sous-projets ajoutés au projet.

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

Lire Gradle - Informations sur les balises en ligne: https://riptutorial.com/fr/android-gradle/topic/9439/gradle---informations-sur-les-balises

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec android-gradle	Community, Daniele Segato, Gabriele Mariotti
2	Code rétractable et ressources	Gabriele Mariotti
3	Comment inclure des fichiers AAR dans un projet sous Android	Gabriele Mariotti, JBirdVegas
4	Configurer les paramètres de signature	DArkO, Gabriele Mariotti
5	Configurer les saveurs du produit	David Medenjak, Gabriele Mariotti, piotrek1543, Tarek El-Mallah
6	Configurer les types de construction	Gabriele Mariotti
7	Configurez votre build avec Gradle	Gabriele Mariotti
8	Déclarer les dépendances	4444, cricket_007, Gabriele Mariotti, jitinsharma
9	Gradle - Informations sur les balises	Chetan Joshi