Android Studio 完全移行ガイド

有山 圭二　著

2016/01/04 版　　発行

はじめに

「Android Studio 完全移行ガイド」を手にとっていただき、ありがとうございます。
はじめに、本書が題材にしている「Android Studio」と「ADT」について解説します。

図 1　Android Studio（左）と ADT（右）

「Android Studio」は、Google I/O 2013 で発表された、Android アプリ開発の **IDE**（統合開発環境）です。チェコ JetBrains 社が開発している「IntelliJ IDEA[*1]」をベースに開発されています。

一方、**ADT**（Android Developer Tools）は、2007 年に Android が発表されて以来、普及してきた Eclipse ベースの IDE です。当初は Eclipse のプラグインとして配布され、中期以降は Eclipse に組み込まれて単体版が配布されていました。

Time to Migrate

2015 年 1 月、Android Studio のバージョンが 1.0 になると同時に「公式開発環境」の冠は Android Studio に移され、ADT は再びプラグインによる提供のみとなりました。

更に米 Google 社は、それから半年経つか経たないかのうちに、2015 年末で ADT の開発やサポートを打ち切ることをアナウンスしました。

- An update on Eclipse Android Developer Tools
  - http://android-developers.blogspot.jp/2015/06/an-update-on-eclipse-android-developer.html

これは「AS に移行することで発生するリスク」より「移行しないリスク」が高くなったことを意味します。今後、Android 本体のバージョンやビルドツールなどのアップデートがあれば、ADT でアプリがビルドできなくなる可能性があるからです。

これまで ADT で開発していたすべてのプロジェクトが開発を続けていくには、Android Studio に移行（Migrate）する必要があります。

とは言え、使い慣れてきた IDE を変えるのは抵抗があるものです。「移行にチャレンジしたけど挫折した」と言う人も少なくはないでしょう。また、ADT だけでなく、Ant などのビルドシステムで培ってきた資産をどのように移行すれば良いのか、不安に思う人もいることと思います。

本書ではまず、ADT のプロジェクトから Android Studio への移行について、準備から実際の手順を含めて解説します（第 1 章）。次に、Ant などのビルドシステムで実現していた処理を Android Studio(Gradle) で実現す

[*1] 「IntelliJ IDEA Community Edition」をベースに開発されています

るにはどのようにすればいいか、ユースケース毎に解説します（第 2 章）。

本書が、皆さんの Android Studio への道しるべになることを願っています。

ソフトウェアのバージョン

本書で取り上げるソフトウェアのバージョンは、次の通りです。

- ADT
  - Eclipse Mars 1. Release (4.5.1)
  - ADT plugin 23.0.7
- Android Studio 1.5.1
- Android NDK r10e

最新情報の提供

本文書に関する最新情報は、

- Android Studio 完全移行ガイド
  - PDF 版 http://keiji.github.io/farewell-adt-book/archives/farewell-adt.pdf
- GitHub https://github.com/keiji/farewell-adt-book

で、提供していきます。

本書について

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

表記関係について

本書に記載されている会社名、製品名などは、一般に各社の登録商標または商標、商品名です。会社名、製品名については、本文中では©、®、TM マークなどは表示していません。

著作権について

Creative Commons License

[*2] http://creativecommons.org/licenses/by-nc-nd/4.0/deed.ja

目次

第 1 章

Android Studio への移行

この章では、ADT のプロジェクト（ワークスペース）を Android Studio に移行する方法について解説します。

1.1 ADT プロジェクトのインポート機能について

Android Studio には、ADT のプロジェクト（ワークスペース）を自動でインポートする機能があります。

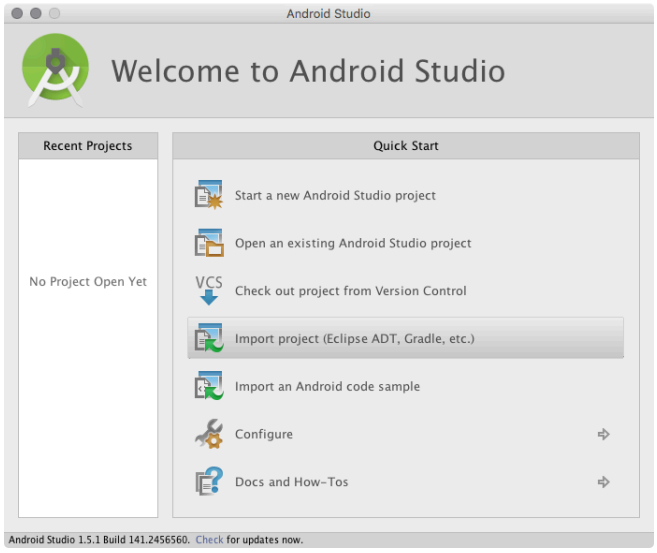


図 1.1　Import Project(Eclipse ADT, Gradle etc.)

しかし、これは ADT のワークスペースを「とりあえず使えるようにする」だけのものです。この機能を使った場合、Android Studio 本来の性能を発揮できるプロジェクト構成にはなりません。

特に、ライブラリに依存していたり、複数のプロジェクトで構成されているワークスペースは、インポートすら正常に完了しないということも起こります。

残念ながら ADT そのものが開発とサポートが終了するソフトウェアなので、今後、この機能の拡充に開発リソースが注がれることは期待できません。

ADT から Android Studio へ移行する一番確実な方法は、Android Studio のプロジェクトを新しく作成して、ADT のワークスペースから各要素を手動で移行することだと筆者は考えます。

1.2 手動での移行（準備編）

それでは、実際に ADT のワークスペースを Android Studio に移行していきましょう。

ここでは、ワークスペース「farewelladt_workspace」を例に移行作業を進めます（リスト 1.1）。

リスト 1.1: ADT のワークスペース構成

```
.
|-- FarewellAdt
|   |-- AndroidManifest.xml
|   |-- assets
|   |-- bin
|   |-- jni
|   |-- libs
|   |-- obj
|   |-- proguard-project.txt
|   |-- project.properties
|   |-- res
|   `-- src
`-- library
    |-- AndroidManifest.xml
    |-- assets
    |-- bin
    |-- libs
    |-- proguard-project.txt
    |-- project.properties
    |-- res
    `-- src
```

1.2.1 各プロジェクト設定を確認する

はじめに、現在の ADT のワークスペースを構成する各プロジェクトの設定を確認します。
確認する内容を表 1.1 に示します。

表 1.1 プロジェクトチェックシート

確認項目	確認する場所（ADT）
プロジェクト名	ADT のプロジェクト画面
パッケージ名	AndroidManifest.xml（package）
バージョンコード	AndroidManifest.xml（versionCode）
バージョン名	AndroidManifest.xml（versionName）
対応 API Level	AndroidManifest.xml（minSdkVersion）
対象 API Level	AndroidManifest.xml（targetSdkVersion）
Project Build Target	「プロジェクト設定」参照
ライブラリプロジェクト	「プロジェクト設定」参照
依存するプロジェクト	「プロジェクト設定」参照
利用ライブラリ	各プロジェクトの libs フォルダ
ユニットテスト	テストプロジェクト
NDK	各プロジェクトの jni フォルダ

プロジェクト設定

いくつかの設定は、ADT の画面上で確認する必要があります。

プロジェクト一覧で右クリックして［Properties］をクリックすると、Project Properties が表示されます。左側メニューから［Android］を選択すると図 1.2 の画面になります。

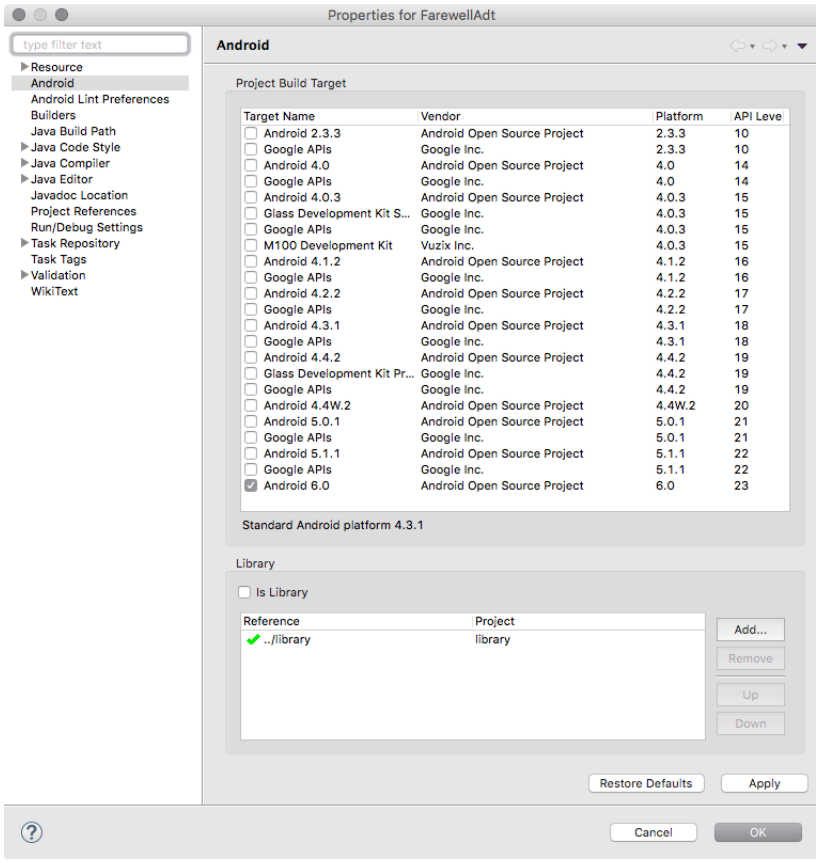


図 1.2　Project Build Target と Library

確認が必要な情報は次の通りです。

- ［Project Build Target］で選択されている Android のバージョン（API Level）
- ［Library］ → ［is library］のチェックの有無
- ［Reference］に登録されているプロジェクト名

例となる「farewelladt_workspace」には 2 つのプロジェクト「FarewellAdt」と「library」があります。それぞれについて値を確認していきます。

表 1.2　FarewellAdt プロジェクト

確認項目	値
プロジェクト名	FarewellAdt
パッケージ名	io.keiji.farewelladt
バージョンコード	1
バージョン名	1.0
対応 API Level	15
対象 API Level	23
Project Build Target	Android 6.0 - API Level 23
ライブラリプロジェクト	false
依存するプロジェクト	library
利用ライブラリ	android-support-v4.jar
ユニットテスト	なし
NDK	あり

表 1.3　library プロジェクト

確認項目	値
プロジェクト名	library
パッケージ名	io.keiji.library
バージョンコード	1
バージョン名	1.0
対応 API Level	8
対象 API Level	21
Project Build Target	Android 6.0 - API Level 23
ライブラリプロジェクト	true
依存するプロジェクト	なし
利用ライブラリ	android-support-v4.jar
ユニットテスト	なし
NDK	なし

1.3　手動での移行（実践編）

情報が揃ったら、いよいよ移行作業を始めます。移行作業は次のステップで進めていきます。

1. 新規（Android Studio）プロジェクトの作成
2. モジュールの作成
3. ライブラリの移行
4. リソースの移行
5. アセットの移行
6. ソースコードの移行
7. テストの移行
8. AndroidManifest.xml の移行
9. NDK（JNI）の移行

移行の途中でエラーが出ても慌てないでください。エラーの修正は一通り移行作業が終わった後の方が負担が少ないので、まずは最後までやりきるように心がけましょう。

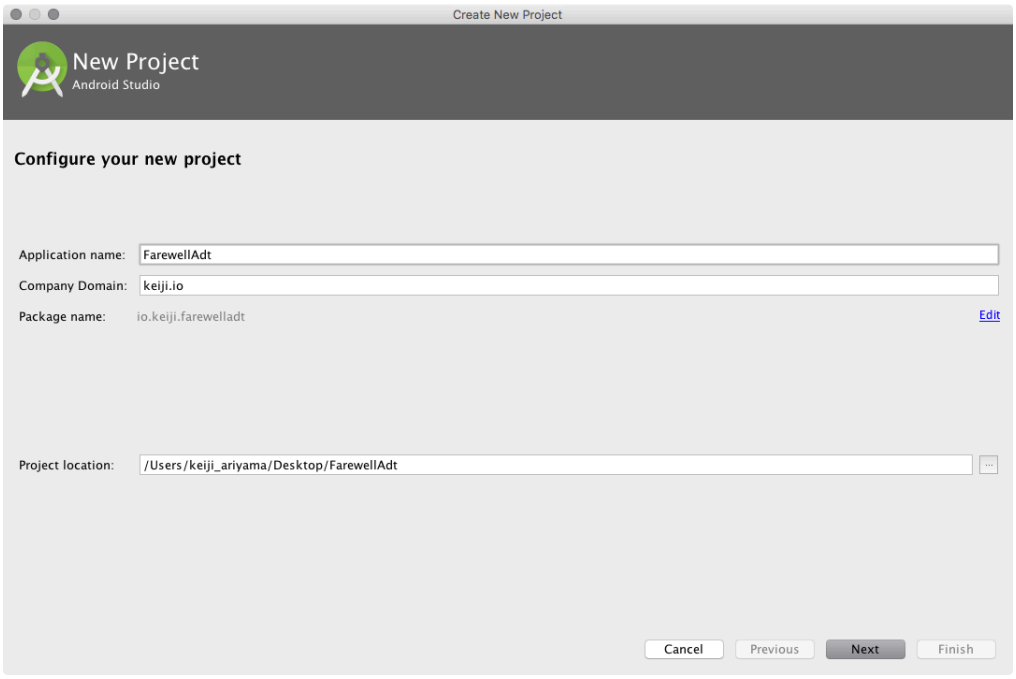
1.3.1　新規プロジェクトの作成

移行先（Android Studio）でプロジェクトを作成します。まずはアプリの起点となる「FarewellAdt」から作成しましょう。

初期の画面から［Start a new Android Studio project］をクリックすると、プロジェクトウィザードが起動します。

Company Domain と Package name

［Application name］と［Company Domain］をそれぞれ入力します。これらを組み合わせたものがデフォルトの［Package name］になります。

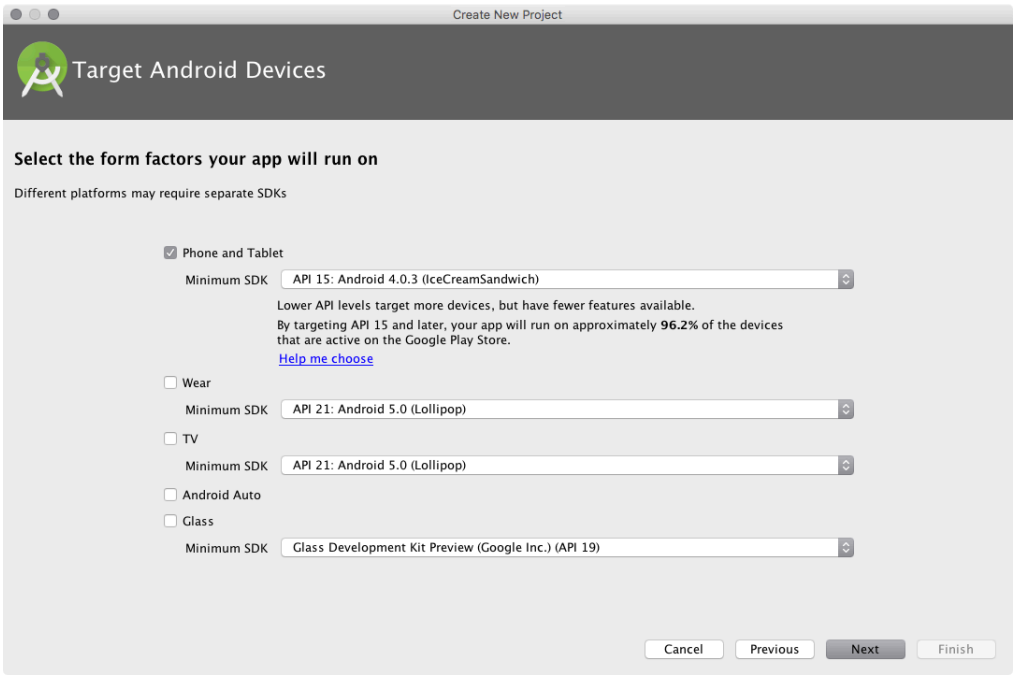


この際［Company Domain］は自動的に逆順になります。ADT のプロジェクトウィザードのように逆順で入力してしまうと、間違った［Package name］が設定されてしまうので注意が必要です。

なお、［Edit］をクリックすると、［Package name］を直接書き換えることができます。

対応バージョン

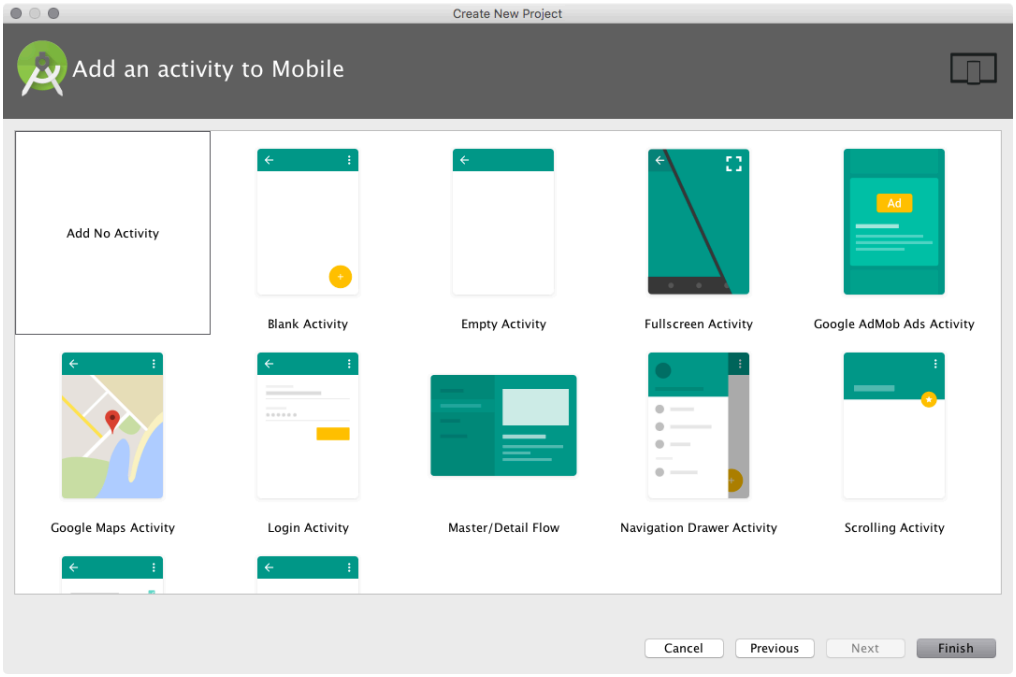
対応バージョンを選択します。ここで選択した API Level が、`minSdkVersion` になります。後から書き換えることもできますが、今回は先ほど確認した値「Android 4.0.3 (Ice Cream Sandwich)」を選択します。



最後に、追加する Activity を選択する画面が表示されますが、今回は ADT からの移行なので Activity は必要ありません。

［Add no Activity］を選択して［Finish］をクリックすると、Android Studio はプロジェクトの生成を開始し

ます。



Android Studio では、プロジェクトの生成時にはインターネット接続が必要となる場合があります。プロジェクトの生成（ビルド）時に必要なソフトウェアをインターネットを通じてダウンロードするためです。

> 管理の単位の違い
>
> ADT と Android Studio では「プロジェクト」という単語の意味が異なるので注意が必要です。
>
> ADT（Eclipse）における最上位の単位は「ワークスペース」と呼ばれ、ワークスペースの下には複数の「プロジェクト」が配置されます。一方、Android Studio（IntelliJ IDEA）は「プロジェクト」が最上位の単位であり、その下に「モジュール」が配置される構造になっています。

Project View への切り替え

Android Studio は、プロジェクトの作成直後には標準で「Android View」を表示します。しかし「Android View」は、実際のプロジェクト構造が見えず、移行作業には適しません。

作業を進めるにあたってまず、左上のメニューから「Project View」に切り替えてください。

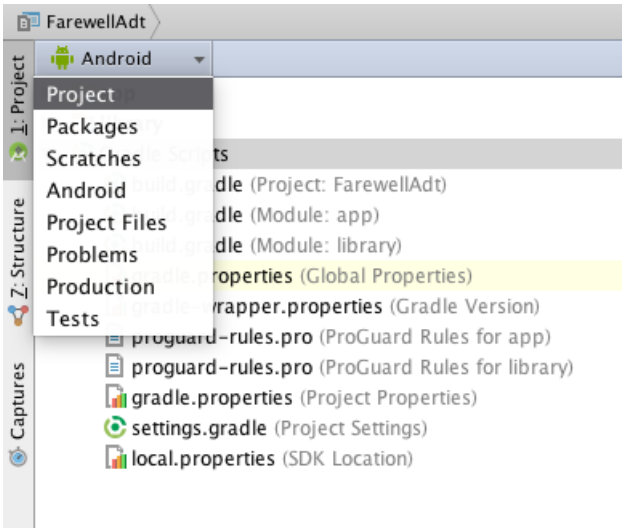


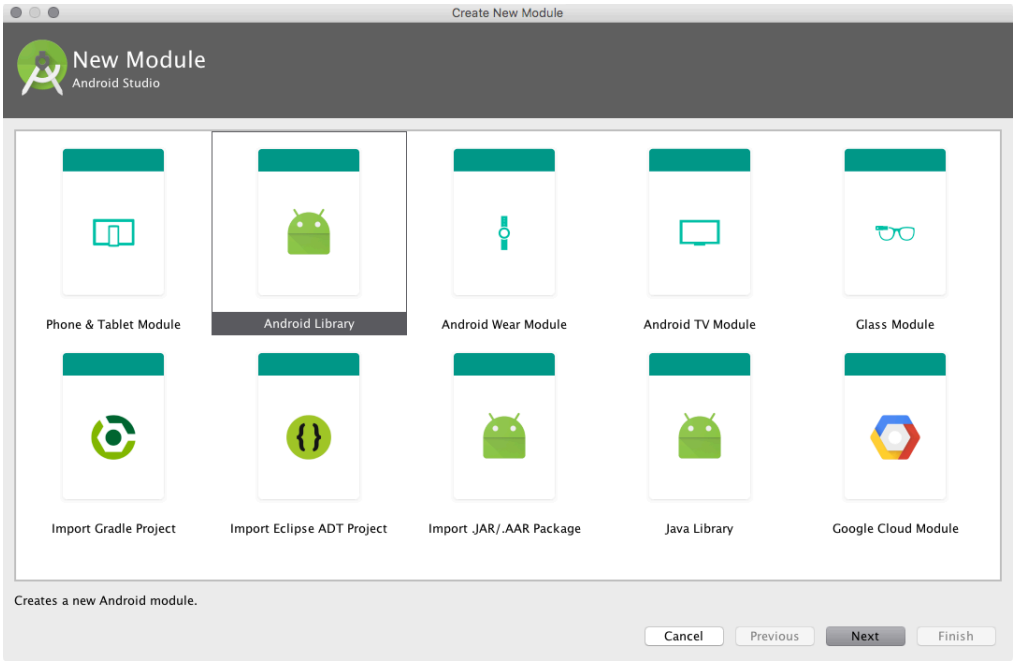
図 1.3　［Project］を選択する。［Project Files］でないことに注意

1.3.2　モジュールの作成

次に、モジュールを作成します。ADT では「library」として扱っていたプロジェクトを「モジュール」にします。

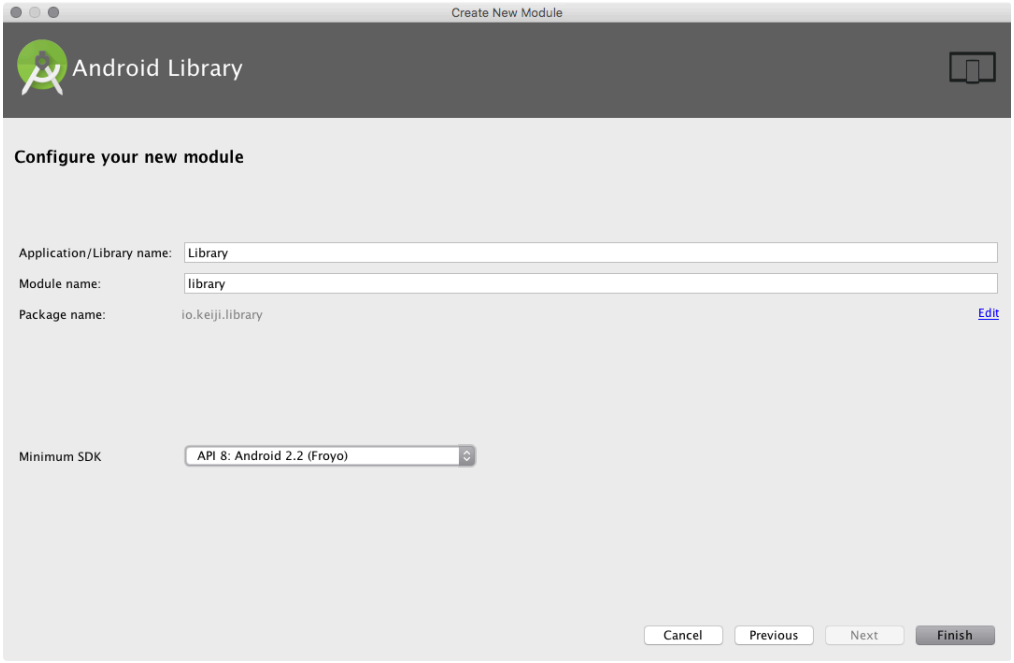
［File］メニューから［New］→［New Module］をクリックすると、モジュール作成ウィザードが起動します。

まず、モジュールの種類を選択します。ADT の「library」プロジェクトの場合は、［Android Library］を選択します（ADT のプロジェクトがアプリケーションなら、［Phone and Tablet］を選択します）。



［Application/Library name］を入力[*1]し、次に対応バージョンを選択します。ここで選択した API Level が、モジュールの `minSdkVersion` になります。ADT の「library」プロジェクトの場合は、「Android 2.2 (Froyo)」を選択します。

[*1] ［Application/Library name］は、先頭が大文字で入力することが推奨されていますが、ADT のプロジェクトに完全に合わせるなら小文字でも問題ありません



［Finish］をクリックすると、Android Studio はモジュール「library」を生成します。

参照（dependencies）の追加

ここまでを終えて、Android Studio のプロジェクトには「FarewellAdt」と「library」の 2 つのモジュールがあります。

しかし、この段階では「FarewellAdt」は「library」にあるコードやリソースを参照できません。モジュール「FarewellAdt」から、モジュール「library」へ参照を設定する必要があります。

参照は、`build.gradle` に記述します。モジュール「FarewellAdt」の `app/build.gradle` を開いて、リスト 1.2 のように `dependencies` に `compile project(':library')` を追記します。

リスト 1.2: library への参照（dependencies）を設定

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

android {
    ...
}

dependencies {
    ...
    compile project(':library')
}
```

> **Gradle**
>
> Android Studio は、ビルドシステムに「Gradle」を採用しています。「build.gradle」は Gradle のビルド設定をまとめたファイル（ビルドファイル）です。

図: Gradlephant

ADT では、Eclipse とプラグインがビルドを実行して **APK**（Application PacKage）を生成しています。また、**CI**（Continuous Integration）を実施しようとすれば、`android` コマンドを使って Ant のビルドファイル生成するのが一般的です。

しかし、ADT と Ant はあくまで別のビルド環境です。Ant のビルド設定を複雑にしていくうちに、ADT ではビルドできないプロジェクト構造になる（分断されてしまう）ことも珍しくありません。それでは、統合開発環境としての ADT の機能がフルに発揮できなくなります。

一方、「Gradle」を使ってビルドする Android Studio は、ADT と Ant のような分断が起こりません。

Gradle によるビルドは、ADT から Android Studio に移行する最も大きなメリットと言えるでしょう。

1.3.3 ライブラリの移行

続いて、ライブラリを移行します。

ADT のプロジェクト「FarewellAdt」と「library」は、どちらもライブラリ `android-support-v4.jar` を `libs` に配置しています。

Android Studio のモジュール「FarewellAdt」と「library」の `build.gradle` を開くと、どちらも `dependencies` に `appcompat-v7` が設定されています（リスト 1.3）。

`appcompat-v7` は、`android-support-v4` を含みます[*2]。つまり、このケースではライブラリの移行は必要ないということになります。

リスト 1.3: appcompat-v7 への参照を設定している

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
    compile 'com.android.support:appcompat-v7:23.1.1'
    ...
}
```

リポジトリからライブラリを追加

Android Studio（Gradle）では基本的に、ライブラリの追加にあたって jar ファイルを `libs` に配置する必要がありません。使いたいライブラリの名前とバージョンを dependencies に記載すると、Gradle は、Maven

[*2] Gradle で appcompat-v7 を設定すると、自動的に android-support-v4 の機能も取り込まれます

Cnetral[*3]や jCenter[*4]などのリポジトリからバイナリを自動的にダウンロードしてビルドを実行します。

例えば、ButterKnife を使う場合、ADT では ButterKnife のバイナリ（jar）ファイルをダウンロードして libs にコピーします。一方、Android Studio（Gradle）では、dependencies に ButterKnife の情報を記載するだけで設定が完了します（リスト 1.4）。

もちろん、リポジトリに登録されていないライブラリについては、これまで通り jar ファイルを libs に配置できます。

リスト 1.4: ButterKnife を dependencies に追加

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
    compile 'com.jakewharton:butterknife:7.0.1'
}
```

1.3.4 リソースの移行

ADT のプロジェクトのリソースは、res ディレクトリにあります。Android Studio の各モジュールの src/main/res ディレクトリにコピーします。

> drawable と mipmap
>
> ADT では、アプリのアイコンは標準で res/drawable-*に配置されます。
>
> 一方、Android Studio では、アプリのアイコンは標準で src/main/res/mipmap-*に配置されます。
>
> mipmap は、3D テクスチャとしての描画に最適化された画像群の名称ですが、実際にはこれまで res/drawable-*に置いていた画像と違いはありません。PNG 形式で、各解像度別の画像サイズも同じです。
>
> しかし、src/main/res/mipmap-*に配置したファイルは R.mipmap.*や@mipmap/*を通じてアクセスすることに注意してください。ソースコードやリソースから drawable として参照していた画像を mipmap に置く場合、それぞれの参照を変更しなくてはなりません。
>
> ADT から画像リソースを移行する場合、アイコン画像は src/main/res/mipmap-*に移動する。また src/main/AndroidManifest.xml を開いて、アイコンの参照先を@drawable/ic_launcher から @mipmap/ic_launcher に変更するなどして調整してください。

1.3.5 アセットの移行

ADT のプロジェクトのアセットは、assets ディレクトリにあります。Android Studio の各モジュールの src/main/assets ディレクトリにコピーします。

1.3.6 Java ソースコードの移行

ADT のプロジェクトの Java ソースコードは、src ディレクトリにあります。Android Studio の各モジュールの src/main/java ディレクトリにコピーします。

[*3] Maven Central: http://search.maven.org/

[*4] jCenter: https://bintray.com/bintray/jcenter

この際、ADT からパッケージを選択してコピー&ペーストすると、パッケージ構造を正確にコピーできない場合があります。エクスプローラー（OS X の場合は Finder）などを使ってディレクトリ（パッケージ）ごとコピーしてください。

1.3.7 テストコードの移行

ADT にテストコードがある場合、Android Studio の各モジュールの `src/AndroidTest/java` ディレクトリにコピーします（Android Studio は、テストコードをモジュール毎に管理します）。

この際、ADT からパッケージを選択してコピー&ペーストすると、パッケージ構造を正確にコピーできない場合があります。エクスプローラー（OS X の場合は Finder）などを使ってディレクトリ（パッケージ）ごとコピーしてください。

build.gradle の設定

最後に、テストを実行するために、`build.gradle` に `testInstrumentationRunner` を記述します。

また、`dependencies` に `com.android.support.test:support-annotations` と `com.android.support.test:runner` を追加します。

リスト 1.5: testInstrumentationRunner と dependencies を記述する

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

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23

        versionCode 1
        versionName "1.0"

        // 実際には一行で記述
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunn\
er"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
        }
    }
}
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile project(':library')

    androidTestCompile 'com.android.support:support-annotations:23.1.1'
    androidTestCompile 'com.android.support.test:runner:0.3'
}
```

1.3.8 AndroidManifest.xml の移行

ADT のプロジェクトの AndroidManifest.xml は、プロジェクトの直下にあります。Android Studio の各モジュールの src/main/ディレクトリにある AndroidManifest.xml に内容をコピーします。

build.gradle へ項目を移動する

Android Studio では、ADT で AndroidManifest.xml に設定していた値のいくつかは、build.gradle に設定するように変更されています。

build.gradle へ移動する項目を表 1.4 に示します。

表 1.4　build.gradle へ移動する項目

作業項目
compileSdkVersion
minSdkVersion
targetSdkVersion
versionCode
versionName

これらの値を AndroidManifest.xml（リスト 1.6）から build.gradle に移動します。

リスト 1.6:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="io.keiji.farewelladt"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="23" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

次に、versionCode と versionName の値を build.gralde に移行します（リスト 1.7）。

リスト 1.7: versionCode と versionName の値を設定

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

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile project(':library')
}
```

最後に、移行した各値を Android Studio 側の `AndroidManifest.xml` から削除すれば移行は完了です。

1.3.9 NDK の移行

NDK（Native Development Kit）を使っている場合は、ネイティブのコードを移行するのに加え、NDK を扱えるように設定する必要があります。

NDK のパスを設定

Android Studio に NDK のパスを設定します。プロジェクトで右クリックをして［Open Module Settings］をクリックすると設定画面が開きます。

［SDK Locations］の［Android NDK Location］に、NDK のパスを設定します。

図 1.4　Android NDK Location

ネイティブのソースコード

ADT のプロジェクトのネイティブのソースコードは、`jni` ディレクトリにあります。Android Studio の各モジュールの `src/main/jni` ディレクトリにコピーします。

build.gradle の設定

続いて、`build.gradle` を NDK 用に設定します。

まず、プロジェクトのトップレベルにある `build.gradle` を変更します（リスト 1.8）。本稿執筆時点で、NDK ビルドに対応しているのは実験（experimental）バージョンの Gradle だけです。

リスト 1.8: プロジェクトのトップにある build.gradle

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle-experimental:0.4.0"
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

gradle-experimental のバージョン

本稿執筆時点（2015 年 12 月）で gradle-experimental の最新バージョンは 0.6.0-alpha2 です。

0.6.0-alpha2 に設定した場合、gradle/wrapper/gradle-wrapper.properties で gradle-2.9-all.zip を指定する必要があります。

次に、プロジェクト「FarewellAdt」と「library」それぞれに適用する plugin を変更します。

リスト 1.9 は、com.android.application を com.android.model.application に変更しています。

com.android.model.application は通常のプラグインと **DSL**（Domain-Specific Language）が違うので注意してください。

android は model の下に位置します。compileSdkVersion などの値も空白ではなく=で指定し、defaultConfig には.with が必要です。また、API Level の指定は minSdkVersion.apiLevel のようになります。

リスト 1.9: NDK の DSL 用に記述を調整

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

model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.2"

        defaultConfig.with {
            applicationId = "io.keiji.farewelladt"
            minSdkVersion.apiLevel = 15
            targetSdkVersion.apiLevel = 23
            versionCode = 1
            versionName = "1.0"
        }
    }

    android.buildTypes {
        release {
            minifyEnabled = false
            proguardFiles.add(file("proguard-rules.pro"))
        }
    }

    android.ndk {
        moduleName = "hello-jni"
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.android.support:appcompat-v7:23.1.1'
}
```

プロジェクト「library」のプラグインも com.android.model.library に変更します（リスト 1.10）。

リスト 1.10:

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

```
model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.2"

        defaultConfig.with {
            minSdkVersion.apiLevel = 15
            targetSdkVersion.apiLevel = 23
        }
    }

    android.buildTypes {
        release {
            minifyEnabled = false
            proguardFiles.add(file("proguard-rules.pro"))
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.1.1'

}
```

注意

ライブラリプロジェクトに JNI を設定する際、versionCode と versionName を削除するのを忘れないようにしてください。

ライブラリの場合、defaultConfig.with の中に versionCode や versionName があると、org.gradle.api.internal.ExtensibleDynamicObject が原因でビルドに失敗します。

その他、NDK に関する最新の情報、詳細な設定方法については次のサイトを参照してください。

- 参考
  - http://tools.android.com/tech-docs/new-build-system/gradle-experimental

第 2 章

Gradle Cookbook

この章では、Ant などのビルドシステムで実現していたことを Gradle でどのように実現するか、ユースケース毎に解説します。

2.1 ビルドによって含めるソースコードやリソースを入れ替えたい

「Product Flavors」を使えば、生成する APK ファイルに含めるソースコード（クラスファイル）やリソースを、ビルドのタスクに応じて入れ替えることができます。

ここでは無料の試用版（trial）と、有料版（commercial）を分ける場合を考えます。

2.1.1 Product Flavor を設定する

まず app/src の下に 2 つのディレクトリ「trial」と「commercial」を作成します。これらが各 flavor の起点となります。

commercial と trial を追加した構成

```
.
|-- app.iml
|-- build
|-- build.gradle
|-- libs
|-- proguard-rules.pro
‘-- src
    |-- androidTest
    |-- commercial
    |-- main
    |   |-- AndroidManifest.xml
    |   |-- java.io.keiji.farewelladt
    |   |   ‘-- MainActivity.java
    |   ‘-- res
    |-- test
    ‘-- trial
```

次に、app/build.gradle を開いて、productFlavors の情報を記述します（リスト 2.1）。

リスト 2.1: productFlavors を追加

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

android {

    productFlavors {
```

```
        trial
        commercial
    }
}
```

2.1.2 クラスを入れ替える

それぞれの flavor に、切り替えたいクラスのソースコード（例. `FooBar.java`）を配置します。

FooBar.java をそれぞれの flavor に配置する

```
.
|-- app.iml
|-- build
|-- build.gradle
|-- libs
|-- proguard-rules.pro
‘-- src
    |-- androidTest
    |-- commercial
    |   ‘-- java.io.keiji.farewelladt
    |       ‘-- FooBar.java
    |-- main
    |   |-- AndroidManifest.xml
    |   |-- java.io.keiji.farewelladt
    |   |   ‘-- MainActivity.java
    |   ‘-- res
    |-- test
    ‘-- trial
        ‘-- java.io.keiji.farewelladt
            ‘-- FooBar.java
```

この状態でビルド（`assemble`）すると、それぞれの flavor に配置した `FooBar.java` が組み込まれた APK が作成されます。

> クラスの重複（main と flavor）
>
> 各 flavor の下に置いたクラスを、重複して「main」に置かないように注意してください。「main」とクラスが重複した場合、エラーが起きてビルドが完了しません。
>
> また、flavor の下に置いたクラスで「main」から参照するメソッド、フィールドは、すべて共通にしておく必要があります。

「Gradle」タブで表示されるタスク一覧から `assemble` を実行すると、すべての flavor のビルドを一度に実行できます。`assembleTrial` や `assembleCommercial` のように、flavor を個別にビルドすることもできます。

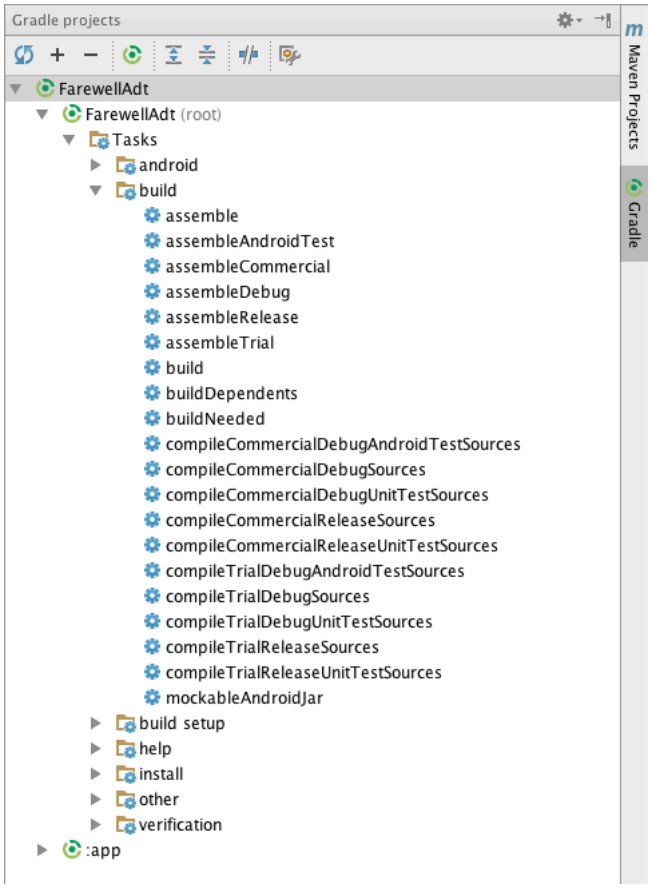


図 2.1　タスク一覧

エミュレーターや実機で実行する flavor を選びたい場合、Android Studio の「Build Variants」タブ（図 2.2）で切り替えることができます。

「`Build Variants`」とは、「Product Flavors」と、デバッグ版・リリース版を切り替える「Build Types」の 2 つの要素を組み合わせたものを言います。

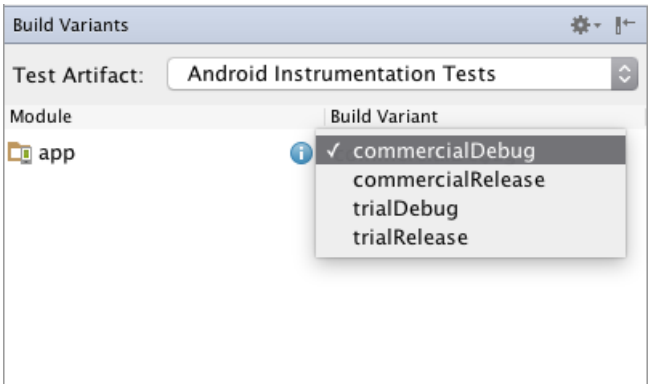


図 2.2　Build Variants - Product Flavor と Build Type の組み合わせ

ビルドは、コマンドラインからも実行できます。エクスプローラー（OS X の場合は Finder）からプロジェクトのトップにディレクトリを移動して、`./gradlew [タスク名]` を実行します。利用できるタスク一覧を表示するには、`./gradlew tasks` を実行します。

コマンドライン版の Gradle は gradle wrapper を使う

```
$ ./gradlew tasks
Starting a new Gradle Daemon for this build (subsequent builds will be faster).
Parallel execution is an incubating feature.
:tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Android tasks
-------------
androidDependencies - Displays the Android dependencies of the project.
signingReport - Displays the signing info for each variant.
sourceSets - Prints out all the source sets defined in this project.

Build tasks
-----------
assemble - Assembles all variants of all applications and secondary packages.
assembleAndroidTest - Assembles all the Test applications.
assembleCommercial - Assembles all Commercial builds.
assembleDebug - Assembles all Debug builds.
assembleRelease - Assembles all Release builds.
assembleTrial - Assembles all Trial builds.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
clean - Deletes the build directory.

[省略]

BUILD SUCCESSFUL

Total time: 8.705 secs
```

2.1.3 リソースを入れ替える

それぞれの flavor のディレクトリに、切り替えたいリソース（例. `ic_launcher.png`）を配置します。

次の例では、commercial に画像リソース `ic_launcher.png` を配置しています。

commercial の flavor に違うアイコンを配置する

```
.
|-- app.iml
|-- build.gradle
|-- libs
|-- proguard-rules.pro
`-- src
    |-- androidTest
    |-- commercial
    |   `-- res
    |       `-- mipmap-xxxhdpi
    |           `-- ic_launcher.png
    |-- main
    |   |-- AndroidManifest.xml
    |   |-- java
    |   `-- res
    |       |-- drawable
    |       |-- layout
    |       |-- mipmap-xxxhdpi
    |       |   `-- ic_launcher.png
```

```
    |       |-- values
    |       `-- values-w820dp
    |-- test
    `-- trial
```

この状態でビルドすると、flavor「commercial」には設定したアイコンが組み込まれた APK が作成されます。

> リソースの重複（main と flavor）
> リソースの場合は「main」のリソースと重複しても、ビルドエラーにはなりません。flavor にリソースがあれば flavor のものが優先され、なかった場合は「main」のリソースが APK に組み込まれます。

2.2 ビルドによってアプリケーションの情報を変えたい

アプリに設定するさまざまな情報（applicationId, minSdkVersion, versionCode, versionName）を、ビルドに応じて変更できます。

2.2.1 applicationIdSuffix と versionNameSuffix

リスト 2.2 は、`applicationIdSuffix` と `versionNameSuffix` で情報を変更している例です。

`applicationIdSuffix` は `applicationId` の末尾に、`versionNameSuffix` は `versionName` の末尾に、それぞれ接尾辞（Suffix）を付加します。

基本となる `applicationId` や `versionName` は、`defaultConfig` で設定した値です。

リスト 2.2: debug ビルドで applicationId の末尾に.debug を付加する

```
android {
    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }

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

`debug` については、`applicationId` が `io.keiji.farewelladt.debug`、`versionName` が `1.0 debug` に設定されます。

2.2.2 ApplicationId の変更

ビルドするアプリの applicationId を変更できます。

リスト 2.3 の例では、ビルドするアプリの applicationId をまったく違うものに変更しています。

リスト 2.3: ApplicationId の変更

```
android {
    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }

    productFlavors {
        trial {
        }
        commercial {
            applicationId 'jp.foo.helloandroidstudio.commercial'
        }
    }
}
```

2.2.3 versionCode および versionName の変更

ビルドするアプリの versionCode および versionName を変更できます。

リスト 2.4 では、ビルドするアプリの versionCode と versionName を変更しています。

リスト 2.4: trial と commercial で異なる versionCode と versionName を設定している

```
android {
    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }

    productFlavors {
        trial {
            versionCode 30
        }
        commercial {
            versionName 'Commercial Version'
            versionCode 10001
        }
    }
}
```

2.2.4 AndroidManifest.xml への反映

applicationIdSuffix を付加したときや applicationId そのものを変えた場合、デバッグ版とリリース版など複数のアプリを一台の端末にインストールできますが、`AndroidManifest.xml` に記述する項目については変更されません。

すると、applicationId が違っていても、ContentProvider の authorities が重複していると、アプリがインストールができないという問題が発生します。

図 2.3　INSTALL_FAILED_CONFLICTING_PROVIDER

AndroidManifest.xml に、`build.gradle` で変更した applicationId を反映するには、`Manifest Merger` を使います（リスト 2.5）。

リスト 2.5: provider の authorities を applicationId で置き換える

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    package="io.keiji.farewelladt"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

        <provider
            android:name=".provider.SomeContentProvider"
            android:authorities="${applicationId}"/>
    </application>

</manifest>
```

`{applicationId}`は、ビルド時にアプリの Biuld Variants に設定されている applicationId で置き換えられます。

そのほか、build.gradle と AndroidManifest.xml との連携については、次の URL を参照してください。

- Android Tools Project Site - Manifest Merger

– http://tools.android.com/tech-docs/new-build-system/user-guide/manifest-merger

2.3 外部コマンドの実行結果をビルドに反映したい

ビルドの過程で外部コマンドを実行できます。

リスト 2.6 は、Git で管理しているプロジェクトの（Git の）ハッシュ値を取得する例です。gitSha メソッドの中で、git コマンドを実行しています。

buildTypes の debug 内でメソッドを実行することで、versionName の末尾にハッシュ値を追加しています（versionNameSuffix）。

リスト 2.6: gitSha メソッド

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

def gitSha() {
    return 'git rev-parse --short HEAD'.execute().text.trim()
}

android {
    buildTypes {
        debug {
            applicationIdSuffix '.debug'
            versionNameSuffix ' ' + gitSha()
        }
    }
}
```

2.4 新しい Build Type を追加したい

最初に定義されている debug と release 以外にも Buid Type を追加できます。

また、initWith を使うと、すでにある Build Type 設定を引き継いで新しい Build Type を追加することができます。

リスト 2.7 は、debug の設定を引き継いで、新しく openbeta を作成する例です。

リスト 2.7: Build Type の追加

```
    buildTypes {
        debug {
            applicationIdSuffix '.debug'
            versionNameSuffix ' ' + gitSha()
        }
        openbeta.initWith(buildTypes.debug)
        openbeta {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
        }
    }
```

openbeta をビルドすると、debug の versionNameSuffix と applicationIdSuffix の設定を引き継いだ上で、さらに minifyEnabled を有効になった APK が生成されます。

2.5 ビルドによって定数の内容を変えたい

アプリのビルド時に自動で生成される BuildConfig には、標準でいくつかの定数が宣言されています。

build.gradle を設定すると定数を追加したり、さらに Build Variants ごとに変更したりできます（リスト 2.8）。

リスト 2.8: buildConfigField による定数の宣言

```
android {

    defaultConfig {
        applicationId "io.keiji.farewelladt"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"

        buildConfigField "String", "API_URL", "\"https://test.keiji.io/\""
    }

    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
            buildConfigField "String", "API_URL","\"https://blog.keiji.io/\""
        }
    }
}
```

この状態でビルドをすると、リスト 2.9 のように定数を追加した BuildConfig.java を生成します。また、リリースビルドでは release の中で宣言した値で BuildConfig.java を生成します。

リスト 2.9: 定数 API_URL が追加されている

```
/**
 * Automatically generated file. DO NOT MODIFY
 */
package io.keiji.farewelladt;

public final class BuildConfig {
  public static final boolean DEBUG = Boolean.parseBoolean("true");
  public static final String APPLICATION_ID = "io.keiji.farewelladt.debug";
  public static final String BUILD_TYPE = "debug";
  public static final String FLAVOR = "trial";
  public static final int VERSION_CODE = 30;
  public static final String VERSION_NAME = "1.0";
  // Fields from default config.
  public static final String API_URL = "https://test.keiji.io/";
}
```

2.6 Lint のエラーでリリースビルドを中止させたくない

> 注意
> Lint の指摘するエラーには対応すべき項目が数多くあります。この設定は慎重に行ってください。

リリースビルド（assembleRelease）を実行すると Lint がコードをチェックして、エラー項目があるとビルドを中止します。

lintOptions を設定することで、Lint でエラーを指摘してもリリースビルドを中止せず、APK を生成できます（リスト 2.10）。

リスト 2.10: Lint のエラーで中止しないようにする

```
android {

    lintOptions {
        abortOnError false
    }
}
```

2.7 署名のキーストアに関する情報をバージョン管理に含めたくない

アプリの署名に使うキーストアの情報を build.gradle に記述することはセキュリティ上、避けたいところです。

キーストアの情報をバージョン管理から切り離すには、まず、プロジェクトのトップに新しくファイル foo-bar.properties を作成します。

次に、作成した foo-bar.properties を .gitignore に加えて、Git の管理から外した上で、リスト 2.11 のようにプロパティを記述します。

リスト 2.11: foo-bar.properties

```
storeFile=[キーストアのパス（フルパス）]
storePassword=[キーストアのパスワード]
keyAlias=[キーの名前]
keyPassword=[キーのパスワード]
```

記述したら、次は build.gradle を書き換えます（リスト 2.12）。

signingConfigs で foo-bar.properties があれば Properties として読み込み、署名に使うキーストアとして設定します。

リスト 2.12: プロパティファイルがあれば読み込む

```
android {

    signingConfigs {
        release {
```

```
            File propFile = rootProject.file("foo-bar.properties")
            if (propFile.exists()) {
                Properties props = new Properties()
                props.load(new FileInputStream(propFile))

                storeFile file(props.storeFile)
                storePassword props.storePassword
                keyAlias props.keyAlias
                keyPassword props.keyPassword
            }
        }
    }

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

            signingConfig signingConfigs.release
        }
    }
}
```

紹介［改訂版］Android Studio ではじめる簡単 Android アプリ開発

図: Android Studio ではじめる 簡単 Android アプリ開発

本書は，新しい Android アプリケーション開発用ソフトウェア“Android Studio”を使った入門書です。セットアップ方法からエミュレータや実機での実行手順を説明し，初版で好評だった「天気予報」「シューティングゲーム」「迷路ゲーム」をさらに工夫して，実際に動かせるプログラムを改良しながら作っていきます。なお，「Android Studio 1.5」をベースに解説しています。

[技術評論社 書籍紹介[*1]より]

目次

[*1] http://gihyo.jp/book/2016/978-4-7741-7859-2

Android Studio 完全移行ガイド

2016 年 01 月 04 日 公開版発行

著　者　有山 圭二