app/build.gradleに記述します。androidエントリのcompileSdkVersionが該当します。

Androidはgradleというビルドツールを使用しており、Androidプロジェクトを構成するためにAndroid Gradle Pluginというプラグインを使用しています。gradleのバージョン、Android Gradle Pluginのバージョン、compileSdkVersionのそれぞれは互いに依存関係があります。compileSdkVersionのみを変更するとビルドできないこともあります。

また、compileSdkVersionを変更する際はAndroidの公式ドキュメント[注2]を参照し、動作の変更点をチェックしましょう。

ターゲットSDKバージョン

ターゲットSDKバージョンはAndroidのみに存在する概念です。アプリを動作させたいSDKバージョンを指定します。SDKのバージョンによって見た目や挙動が変わることがありますので、どのバージョンで動作させる想定なのかを明示する設定になります。

この値はandroid/app/build.gradleに記述します。androidエントリ、defaultConfig内のtargetSdkVersionが該当します。必要な部分だけを抜粋すると以下のようになります。

```
./android/app/build.gradle
android {
    defaultConfig {
        targetSdkVersion 30
    }
}
```

Google Play Storeでは、このtargetSdkVersionを毎年新しいバージョンに引き上げることを必須要件として開発者に課しています。targetSdkVersionを更新しなければ、アプリをアップデートできなくなったり、新しいOSのAndroidからはストアでアプリを検索できなくなったりします。

注2 https://developer.android.com/about/versions?hl=ja

11.2 アプリの設定変更

ホーム画面に表示されるアプリの名前などアプリ設定に関わる内容を紹介します。

アプリ名

ホーム画面に表示されるアプリ名はiOS/Androidネイティブ部分の設定になります。

iOSのアプリ名を変更する

ios/Runner/Info.plistのCFBundleDisplayNameを変更します。

./ios/Runner/Info.plist

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDisplayName</key>
    <string>アプリ名</string>
    <!-- 省略 -->
</dict>
</plist>
```

Androidのアプリ名を変更する

android/app/src/main/AndroidManifest.xmlを編集します。マニフェストファイルと呼ばれ、アプリの情報をAndroidのビルドツールやOSに提供するためのファイルです。アプリ名はapplicationタグのandroid:labelに設定します。

./android/app/src/main/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example"><!-- packageの値は環境によって変わります -->
    <application
        android:label="アプリ名"
        android:name="${applicationName}"
        android:icon="@mipmap/ic_launcher">
        <!-- 省略 -->
    </application>
```

```
</manifest>
```

アプリアイコン

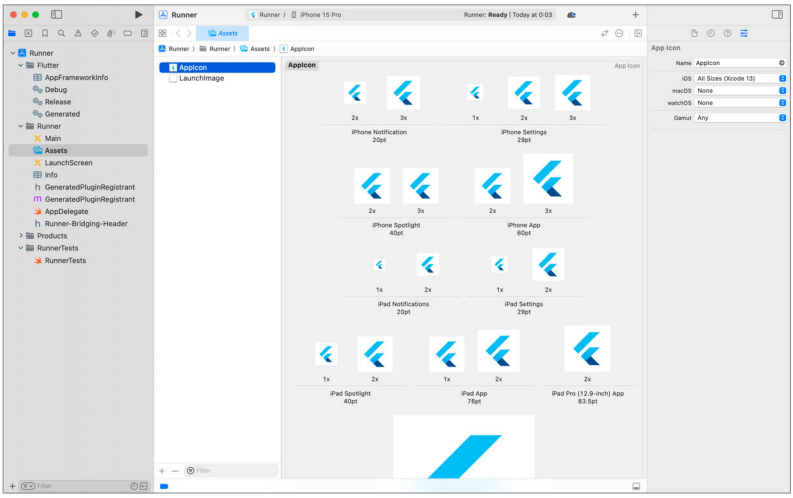
アプリアイコンの設定方法について解説します。

iOSのアプリアイコンを変更する

iOSのアプリアイコンはXcodeのアセットカタログから設定します。Xcodeで`ios/Runner.xcworkspace`を開きます。左側のナビゲーターから「Runner」の配下にある「Assets」を選択します。

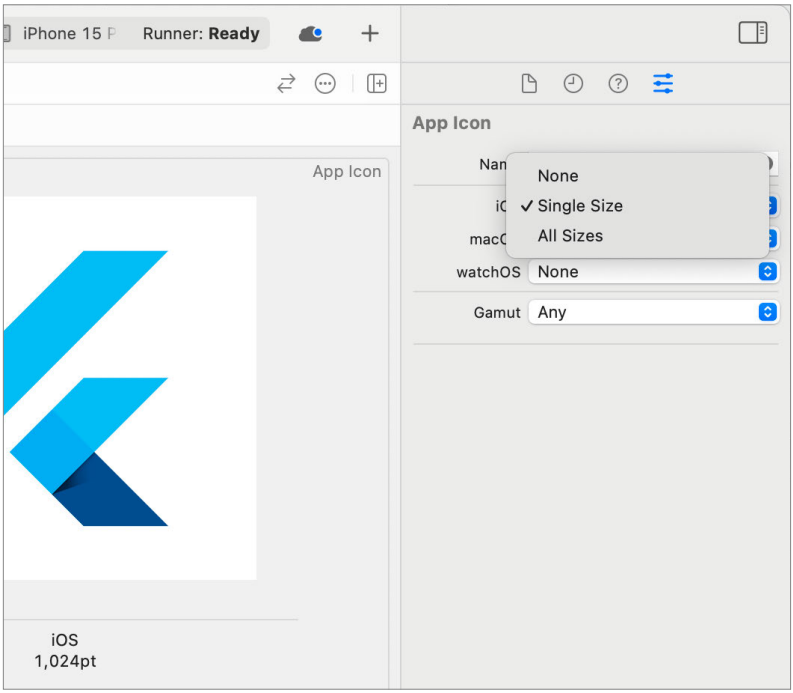
「AppIcon」がアプリアイコンの設定箇所です(**図11.4**)。「AppIcon」をアプリアイコンのアセットとして扱うことはBuild Settingsで設定されています。

図11.4 **AppIconを選択した様子**



デフォルトではさまざまなサイズのアイコン画像を要求されます。2xや3xなどのバリエーションは、Flutterのアセットと同じくディスプレイの解像度によって使い分けられます。大きな画像を1つ指定して自動的にリサイズさせることも可能で、右側のインスペクタで「Single Size」を選択します(**図11.5**)。この場合、細かな線が消えてしまうこともあるので注意してください。

図11.5 アイコンのインスペクタ



Androidのアプリアイコンを変更する

Androidのアイコンはシンプルな画像のほか、バックグラウンドとフォアグラウンドの2つのレイヤで構成されるアダプティブアイコンと呼ばれるものがあります。アダプティブアイコンはOSバージョン8.0以降で導入され、Androidのモデルによってアイコンの表示が変わります。円形、角丸四角形、操作によってアイコンだけが動いて見えるアニメーションが加わるなどさまざまです。

アダプティブアイコンの作成は必須ではありませんが、設定されているとアプリのブランディングに役立ちます。サイズや余白のレギュレーションが細かく定められているので、アダプティブアイコンを作成する場合は公式のアダプティブアイコンの資料[注3]を参照してください。

アイコンの指定はマニフェストファイルで行います。`application`タグの`android:icon`に設定します。

注3 https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

```
./android/app/src/main/AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example"><!-- packageの値は環境によって変わります -->
    <application
        android:label="アプリ名"
        android:name="${applicationName}"
        android:icon="@mipmap/ic_launcher">
        <!-- 省略 -->
    </application>
```

デフォルト値の`@mipmap/ic_launcher`は`android/app/src/main/res`配下にある`mipmap`から始まるフォルダの`ic_launcher.png`という名前の画像を指定しています。`mipmap`から始まるフォルダは、`mipmap-mdpi`や`mipmap-hdpi`などのようにサイズごとに分かれていて、こちらもディスプレイの解像度によって使い分けられます。論理解像度に対して物理解像度の倍率で表現すると**表11.1**のようになります。

表11.1 **フォルダ名と倍率の対応表**

フォルダ名	倍率
mipmap-mdpi	1.0
mipmap-hdpi	1.5
mipmap-xhdpi	2.0
mipmap-xxhdpi	3.0
mipmap-xxxhdpi	4.0

アプリアイコンを手軽に生成するパッケージ

iOSとAndroidそれぞれにアプリアイコンファイルを作成し、設定するのは手間がかかります。そこで、アプリアイコンを手軽に生成するパッケージを紹介します。flutter_launcher_iconsというパッケージです。

flutter_launcher_iconsの導入は`pub`コマンドで行います。アプリの実行に必要なコードではないので、`--dev`オプションを付けてインストールします。

```
$ flutter pub add --dev flutter_launcher_icons
```

アプリアイコンの設定情報は`pubspec.yaml`または任意のYAMLファイルに記述します。以下はその一例です。

```
./pubspec.yaml
# 省略
flutter_launcher_icons:
```

```
image_path: "icon.png" # アイコン画像のパス
ios: true # iOSのアイコンを生成し、デフォルトのものと置き換えるか
android: true # Androidのアイコンを生成し、デフォルトのものと置き換えるか
```

pubspec.yamlに設定を記述したら、以下のコマンドを実行します。

```
$ flutter pub run flutter_launcher_icons
```

もし、任意のYAMLファイルに設定を記述した場合は、-fオプションでファイルを指定します。

```
$ flutter pub run flutter_launcher_icons -f path/to/config.yaml
```

以上の操作を行うことで、iOSとAndroidそれぞれにアイコンが設定されます。flutter_launcher_iconsの詳細な設定項目などはpub.dev[注4]をご覧ください。

スプラッシュ画面

スプラッシュ画面とは、アプリ起動時に一瞬表示される画面のことです。

iOSとAndroidで異なるスプラッシュ画面の位置付け

スプラッシュ画面はiOSとAndroidで位置付けが異なります。

iOSはアプリがすばやく起動することが重視されています。アプリの最初の画面と似たスプラッシュ画面を表示することで、すばやく起動したように感じさせることが推奨されています。表示時間をコントロールすることはできません。表現やブランディングの機会ではなく、文字を含めることは避けるようにガイドラインで定められています。

一方、Androidのスプラッシュ画面はアプリアイコンを中央に表示する標準レイアウトが提供され、ブランディングを意識して、アニメーションや色をカスタマイズ可能となっています。表示時間をコントロールすることも可能です。

iOSのスプラッシュ画面

iOSのスプラッシュ画面はStoryboardというXML形式のファイルで作成します。使用するStoryboardはios/Runner/Info.plistのUILaunchStoryboardNameに設定します。

注4 https://pub.dev/packages/flutter_launcher_icons

./ios/Runner/Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>UILaunchStoryboardName</key>
        <string>LaunchScreen</string>
    </dict>
</plist>
```

デフォルトではLaunchScreenとなっており、ios/Runner/Base.lproj/LaunchScreen.storyboardを指しています。

このStoryboardを編集することでスプラッシュ画面をカスタマイズできます。StoryboardはXcodeで編集します。さまざまなデバイスサイズに対応させるにはAuto Layoutという制約でレイアウトを計算するしくみが不可欠です。本書ではAuto Layoutの解説は割愛しますが、筆者の印象ではAuto Layoutの習得難易度はやや高めです。

Androidのスプラッシュ画面

AndroidではOSバージョン12を境に標準のスプラッシュ画像が導入されました。アプリを実行するAndroidがOSバージョン12以上であれば、アプリアイコンが中央に表示されるスプラッシュ画面が自動的に表示されます。OSバージョン12未満の場合は、androidx.core:core-splashscreenという下位互換ライブラリを使用することで同様のスプラッシュ画面を実現できます。

Androidのスプラッシュ画面は、中央のアイコン、背景色、終了アニメーションなどをカスタマイズできます。本書で詳細な実現方法は割愛します。詳しくはAndroidの公式ドキュメント[注5]を参照してください。

スプラッシュ画面を手軽に実現するパッケージ

iOSとAndroidでスプラッシュ画面の位置付けが異なること、また実現方法もまったくバラバラであることを解説しました。それぞれのプラットフォームに合わせてスプラッシュ画面を実装するのは骨の折れる作業です。Flutterのパッケージで iOSとAndroidのスプラッシュ画面を自動生成するflutter_native_splashというパッケージがありますので、強いこだわりがなければこ

注5　https://developer.android.com/about/versions/12/features/splash-screen?hl=ja

のパッケージを使用するのが手軽でよいと筆者は考えています。

このパッケージはiOSであればスプラッシュ画面に相当するStoryboardを自動生成します。Androidの場合はOSバージョン12以上であれば標準のスプラッシュ画面を使用し、OSバージョン12未満は独自にスプラッシュ画面を生成します。

プロジェクトのディレクトリで、ターミナルから以下のコマンドを実行してください。

```
# flutter_native_splashパッケージを導入
$ flutter pub add flutter_native_splash
```

スプラッシュ画像に関する設定は**pubspec.yaml**または任意のYAMLファイルに記述します。以下はその一例です。

./pubspec.yaml
```
# 省略
flutter_native_splash:
  color: "#00FFFF" # スプラッシュ画面の背景色
  image: icon.png # スプラッシュ画面の中央に表示する画像

  android_12: # Android 12以上の標準スプラッシュ画面の設定
    color: "#FF00FF"

  android_gravity: left # Androidのアイコンの画像の位置
  ios_content_mode: left # iOSのアイコンの画像の位置
```

pubspec.yamlに設定を記述したら、以下のコマンドを実行します。

```
$ dart run flutter_native_splash:create
```

もし、任意のYAMLファイルに設定を記述した場合は、**--path**オプションでファイルを指定します。

```
$ dart run flutter_native_splash:create --path=path/to/config.yaml
```

iOSに関してはStoryboardを自動生成するので、アイコン画像や背景のほか、アイコンの位置などもカスタム可能となっています。一方、Androidに関してはOSバージョン12以上であれば標準のスプラッシュ機能を使用するので、設定項目が分かれています。上記の例ですと、Android12未満の場合は背景色は**#00FFFF**が採用され、アイコンの位置は左寄せになります。Android12以上の場合は背景色は**#FF00FF**が採用され、アイコンの位置は変更不可のため中央になります。