```
class CounterText extends ConsumerWidget {
  const CounterText({super.key});
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final counter = ref.watch(counterProvider);
    return Text(
      '$counter',
      style: Theme.of(context).textTheme.headlineMedium,
    );
  }
}
```

❷

カウンタの状態を表示するウィジェットを**CounterText**ウィジェットとして切り出しました(❷)。**HomeScreen**画面の**build**メソッドではカウンタの監視は行われなくなりました(❶)。

このサンプルを実行すると、カウンタをインクリメントすると**HomeScreen**画面の**build**メソッドは呼ばれず、**CounterText**ウィジェットの**build**メソッドのみが呼ばれることが確認できます。ウィジェットの再構築範囲が小さくなり、カウンタの値を表示するウィジェットとして**CounterText**ウィジェットは再利用性のあるクラスとなりました。

Tips アプリのパフォーマンスを計測する

高速なアプリに仕上がっているかどうかを確認する際は以下の点に注意しましょう。

- **Profileモードでアプリを実行すること**
- **シミュレータなどは使用せず、実機でアプリを実行すること**

DebugビルドしたFlutterアプリはアサーションの処理が含まれています。また、ビルド方式もまったく異なるためReleaseビルドしたアプリよりも遅い可能性が高いです。ProfileモードはほぼReleaseビルドと同等のパフォーマンスを発揮し、さらに最低限のデバッグ情報を含んでいるため、パフォーマンス計測に適しています。

シミュレータやエミュレータもパフォーマンスの特性が異なるため計測には向きません。サポート対象とする端末の中でも、性能の低いものを選択し、実機計測するのが良いとされています。

10.3 まとめ

パフォーマンスを意識した実装は、ときとして保守性を下げることにつながります。本章ではアプリのパフォーマンスとプログラムの保守性、どちらも両立させるポイントに絞って紹介しました。Flutterアプリを開発する際は、本章の内容をいつも頭の片隅に置いて設計を行ってみてください。

第11章

Flutterアプリ開発に必要なネイティブの知識

Flutterでのアプリ開発において、欠かせないiOS/Androidネイティブの知識を紹介します。アプリのID設定、ビルド署名などは一度設定してしまうと繰り返し触れることはないので、知識として定着しづらい部分かもしれません。困ったときに本章が参考になれば幸いです。

11.1 ネイティブAPIのバージョンと最低サポートOSのバージョン

FlutterをiOSアプリ、Androidアプリとしてビルドする際に、ネイティブ側で指定するOSやSDKバージョンによって挙動が変わることがあります。少々ややこしいので、この部分を整理しておきます。

指定するOSとSDKのバージョンは大きく2つ(Androidは3つ)あります。

- **最低サポートOSのバージョン**
- **ビルドSDKバージョン**
- **ターゲットSDKバージョン(Androidのみ)**

最低サポートOSのバージョン

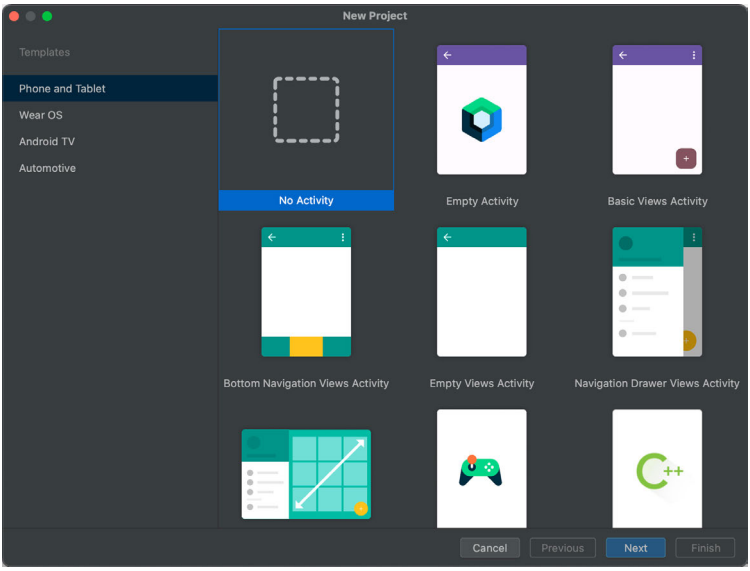
最低サポートOSのバージョンは、アプリをインストールできる最低のOSバージョンです。このバージョンを低く保つことでより多くのユーザーがアプリを利用できることになりますが、その分古いOSでの挙動についても考慮する必要があります。

iOSについてはAppleのデベロッパーサイト[注1]にてバージョン別のシェアが公開されており、この数値を参考にするとよいでしょう。

Androidについては Android Studioの新規プロジェクト作成時にバージョン別のシェアを確認することができます。アプリケーションメニューから「File」➡「New」➡「New Project」を選択します。テンプレートの選択画面は何を選んでもかまいません(**図11.1**)。ここでは「No Activity」を選択し「Next」をクリックします。

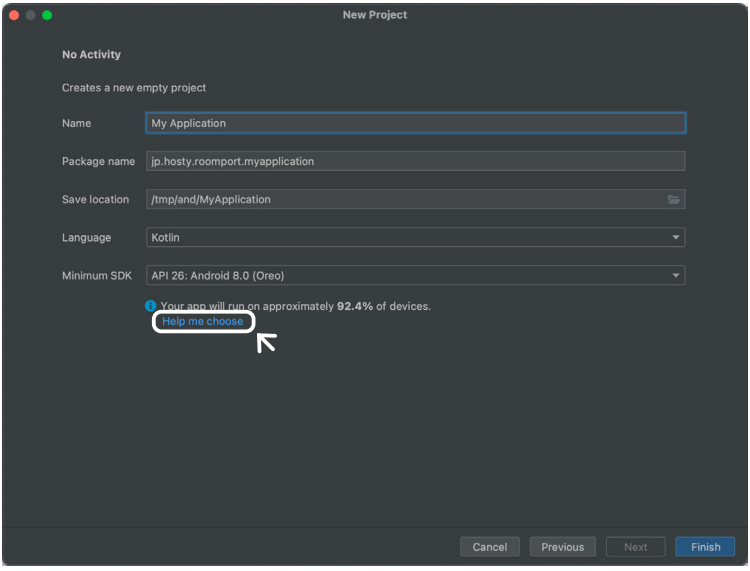
注1 https://developer.apple.com/support/app-store/

図11.1　**テンプレート選択画面**



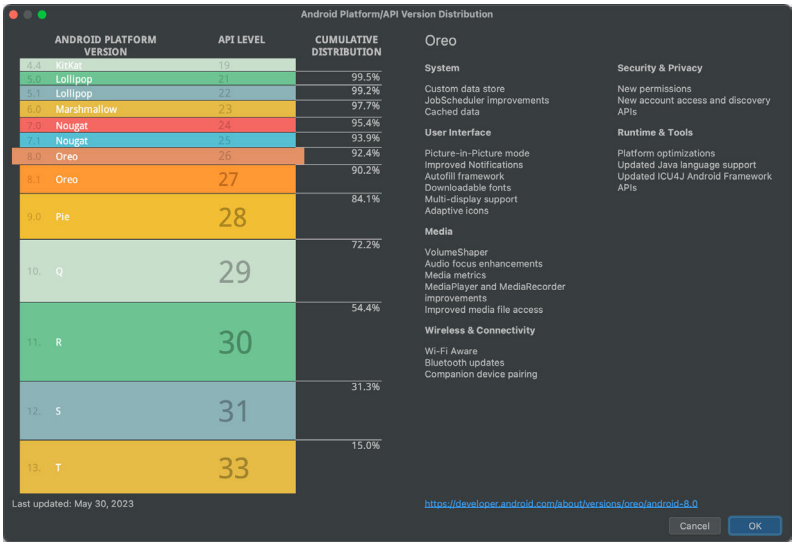
プロジェクトの設定画面にて、「Help me choose」をクリックします（**図11.2**）。

図11.2　**新規プロジェクトの設定画面**



すると、AndroidのOSバージョン別のシェアが表示されます(**図11.3**)。

図11.3 **Androidのバージョン別シェア画面(2023年7月時点)**



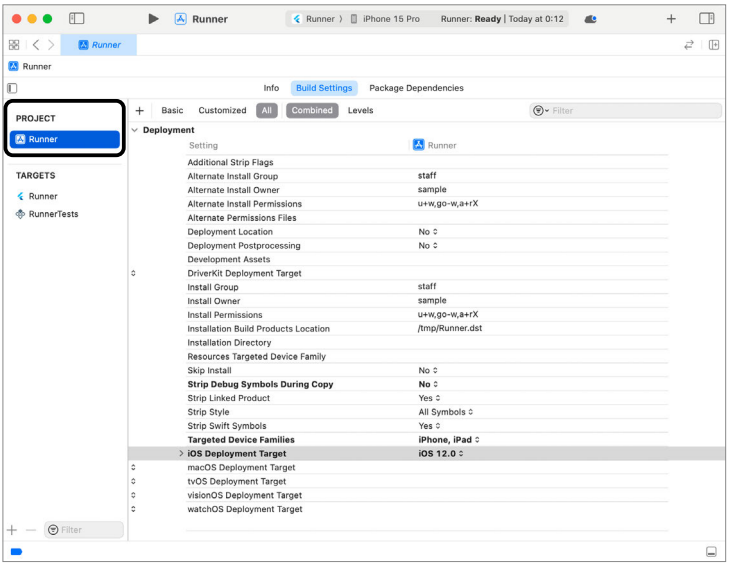
iOSの最低サポートOSバージョンを設定する

iOSアプリの最低サポートOSバージョンを設定する際はXcodeで行うのがよいでしょう。`ios/Runner.xcworkspace`をXcodeで開きます。左側のナビゲーターから「Runner」を選択し、「Runner」プロジェクトを選択します。「Build Settings」タブを選択し、「iOS Deployment Target」を変更します。

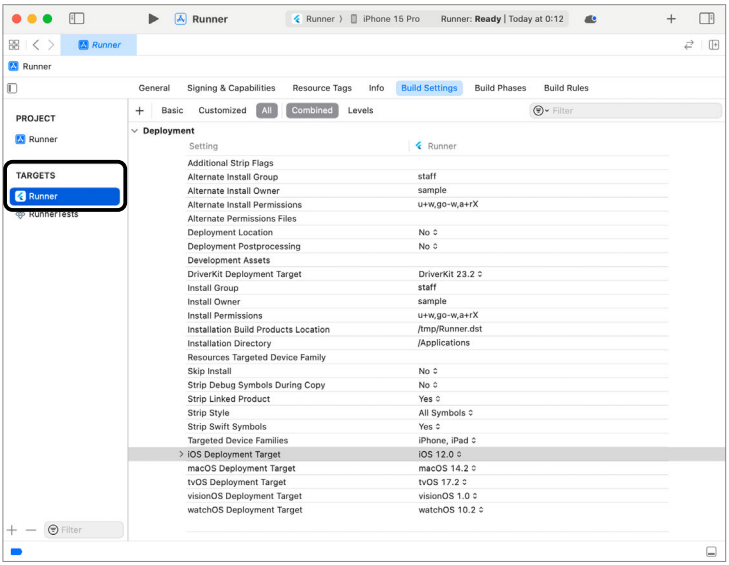
> **Tips XcodeのBuild Settings**
>
> XcodeプロジェクトのBuild Settingsは階層構造になっています。具体的にはプロジェクト全体の設定と、アプリ側の設定があり継承関係にあります。プロジェクト全体設定を変更しても、アプリ側でオーバーライドされていると、変更は反映されません。**図A**の「PROJECT」と「TARGETS」がそれぞれプロジェクト全体の設定とアプリ側の設定になります。

図A　プロジェクトの設定とアプリの設定

●プロジェクト全体の設定



●アプリの設定



Build Settingsのところで太字になっているものが明示的に指定されているものになります。図Aでは、プロジェクト全体のiOS Deployment Target（図A

の上）は「iOS 12.0」が太字になっており明示的に指定されています。
一方、アプリ側のiOS Deployment Target（図Aの下）は太字になっておらず、プロジェクト全体の設定を継承しているということになります。

Androidの最低サポートOSバージョンを設定する

Androidアプリの最低サポートOSバージョンはアプリのビルド構成ファイル`android/app/build.gradle`に記述します。`android`フォルダにも`android/build.gradle`が存在しますが、こちらはプロジェクト全体の構成ファイルなので間違えないように注意してください。`android`エントリ、`defaultConfig`内の`minSdkVersion`が最低サポートOSバージョンになります。

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 {
        minSdkVersion 26
    }
}
```

ビルドSDKバージョン

ビルドSDKバージョンはその名のとおりビルド時に使用するSDKのバージョンで、このバージョンを上げることで新しいネイティブAPIを利用することができます。ただし、このバージョンを上げると古いOSで挙動が変わることがありますので注意が必要です。この挙動の変化は特にAndroidに多く見られます。

iOSのビルドSDKバージョンの設定

iOSには実はビルドSDKバージョンという考え方はありません。Xcodeのバージョンによって使用できるAPIが変わります。Xcodeのバージョン変更により、Flutterアプリの挙動が大きく変わることはまれですが、バージョンを上げる際は慎重に動作確認しましょう。

Androidのビルド SDKバージョンの設定

Androidのビルド SDKバージョンはアプリのビルド構成ファイル`android/`