パッケージを導入するために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_svgパッケージを導入
$ flutter pub add flutter_svg
```

flutter_genはflutter_svgと組み合わせて利用することを想定して、オプションを用意しています。pubspec.yamlのトップレベルにflutter_genセクションを追加します。

./pubspec.yaml
```
flutter_gen:
  integrations:
    flutter_svg: true
```

続いてプロジェクトにSVG形式のアセットを追加します。assetsフォルダに**図4.15**のSVG画像を配置します。

図4.15 **SVG形式のアイコン**

```
~/project_root
└── assets
    └── check.svg
```

アセットを追加したら、コードを生成するコマンドを実行します。

```
$ flutter packages pub run build_runner build
```

SVG画像を描画するコードを書いてみましょう。

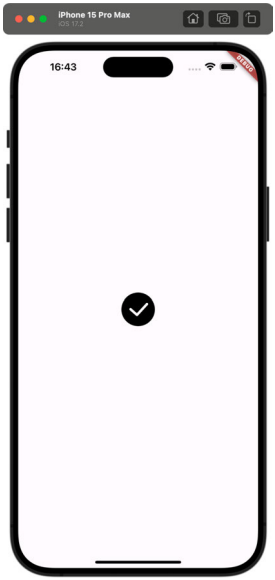
./lib/main.dart
```
import 'package:flutter/material.dart';
import 'gen/assets.gen.dart';

void main() {
  runApp(
    const MaterialApp(
      home: HomeScreen(),
    ),
```

```
  );
}

class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Assets.check.svg(
          width: 72,
          height: 72,
        ),
      ),
    );
  }
}
```

❶

SVG画像を描画するコードに書き換えました(❶)。これで端末の解像度を気にすることなく、画像アセットを扱うことができます。このコードを実行すると、**図4.16**のようにSVG画像が表示されます。

図4.16 **SVG画像を描画した様子**



その他のアセット

flutter_genは画像アセットだけでなく、フォントやJSONファイルなどのアセットにも対応しています。また、オプションで組み合わせて利用できるパッケージもflutter_svg以外にいくつか用意されています。アプリで扱うアセットの種類が増えた場合は、pub.dev[注3]を参照しサポートされているか確認してみましょう。

また、既知の問題があるようで、ソースコードを自動生成する際にエラーが発生した際も同様にpub.devを参照してください。

4.4 dart-define-from-file —— 環境変数を扱う

アプリを設計する際に、コードと設定を分離することは重要です。たとえば開発環境と本番環境でAPIのエンドポイントが異なる場合、環境を切り替えるためにコードを書き換えるのは良い運用とは言えません。ログレベルなども同様です。こういった設定情報は環境変数として扱うことで、コードと分離することができます。

環境変数をJSON形式で記述する

Flutterの`dart-define-from-file`というしくみを利用することで、環境変数をコードから参照できます。例として、プロジェクトルートに`define/env.json`というファイルを作成し、以下の内容を記述します。

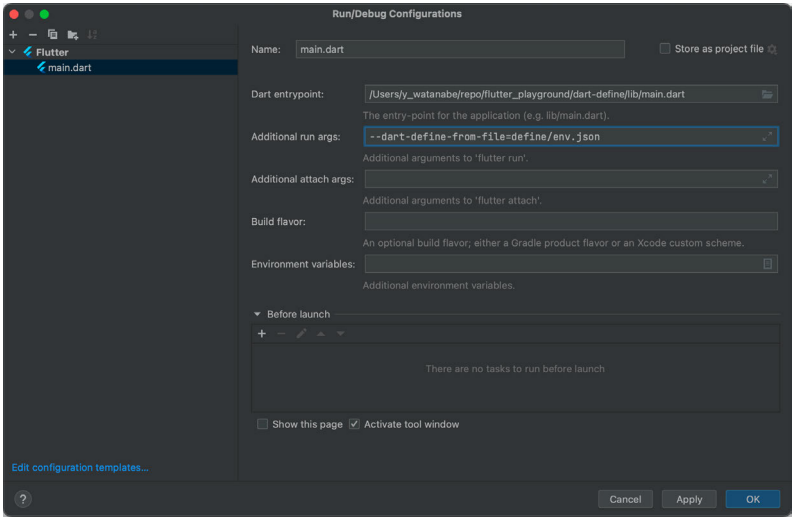
```
./define/env.json
{
  "apiEndpoint": "https://example.com/api",
  "logLevel": 1,
  "enableDebugMenu": true
}
```

このファイルのパスをFlutterのコマンドへ渡すことで、コードから参照できるようになります。Android Studioで実行する場合は、「Run」➡「Edit Configurations」を選択し、「Run/Debug Configrations」ウィンドウを開きます

注3　https://pub.dev/packages/flutter_gen

（**図4.17**）。「Additional run args」に `--dart-define-from-file=` に続いて、JSONファイルのパスを記述します。今回の場合は `--dart-define-from-file=define/env.json` となります。

図4.17 **Run/Debug Configrations ウィンドウ**



環境変数をコードから参照する

Dartのコードから環境変数を参照するには、以下のように記述します。

```
const endpoint = String.fromEnvironment('apiEndpoint');
const logLevel = int.fromEnvironment('logLevel');
const enableDebugMenu = bool.fromEnvironment('enableDebugMenu');
```

`String`型、`int`型、`bool`型のそれぞれに対応した`fromEnvironment`メソッドを呼び出します。引数には環境変数のキーを指定します。

このとき必ず`const`変数に代入するか、呼び出し側に`const`キーワードを付与する必要があるので注意してください。これを忘れると環境変数が取得できず、デフォルト値が返されます。キーが間違っている場合も同様です。デフォルト値は`fromEnvironment`メソッドの第二引数で指定するか、未指定の場合は**表4.3**の値が返されます。

表4.3 **fromEnvironment() のデフォルト値**

型	デフォルト値
String	空文字
int	0
bool	false

4.5 まとめ

アプリのローカライズ、アセットの管理、環境変数の扱い方を紹介しました。

Flutterは、たとえ日本語だけをサポートするアプリであっても、しっかりと対応しなければ意図せず英語が表示されてしまうことがあります。ローカライズ対応は少々手間ですが、はじめに整えておくことでメッセージの管理にも役立ちます。

アセットはパス文字列にタイプミスの懸念があるのでflutter_genを利用して安全に扱うことが望ましいです。また、解像度別の画像を用意するのは手間がかかりますので、SVG形式のファイルを使うのがお勧めです。

コードと設定を分離する手法として、環境変数を扱う方法を紹介しました。

本章で紹介した内容は、製品レベルのアプリを開発、保守していくうえで重要な要素です。もちろん要件によっては不要な要素もあるでしょうが、採用するか否かをはじめに検討しておくことで、後々の開発がスムーズに進むことでしょう。

wassyoi：t--sekiguchi@hotmail.co.jp

第5章 テーマとルーティング

本章ではFlutterフレームワークの機能を紹介します。その中でも「テーマ」「画面遷移」この2点に的を絞って解説します。筆者の経験上、この2つの要素はおおよそどのようなアプリを開発する場合にも知識として必要になり、後工程での方針変更は手間がかかることがあります。

前章は、はじめに整えておくべき要素を紹介しました。本章では、はじめに設計しておくとよい要素として、フレームワークの2つの要素を紹介します。

なお、本章では解説に重きを置くため、関連したコードの断片を掲載していますので、省略されている部分がある点に留意してください。完全なサンプルコードの場合は、その旨を明記しています。手もとで動作確認する際は、`./lib/main.dart`を書き換えてください。

5.1 テーマ —— アプリ全体のヴィジュアルを管理

アプリ全体を通した色やフォントを定義し、適用する方法を解説します。アプリのUIで一貫した世界観を演出したり、視覚的にわかりやすいことは重要です。

本節では「アプリ全体を通した色やフォントの定義」をテーマと呼ぶことにし、テーマに関する2つの機能を解説します。1つ目はアプリのテーマを自動的に計算し適用する機能、2つ目はアプリ独自のテーマを管理し適用する機能です。

フレームワークによるテーマの自動計算機能

まずはテーマの自動計算機能を確認しましょう。Flutterのテンプレートプロジェクトで動作を確認します。第1章で解説した方法で新たにプロジェクトを作成し、アプリを実行します。

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
```