開発の土台づくり

第4章で解説した「開発の土台づくり」の要素です。アプリの日本語化など第6章のハンズオンで実装した機能は省略します。Web APIの呼び出しに必要なアプリケーションIDを環境変数として扱います。

導入するパッケージは「8.3 アプリで使用するパッケージを導入する」で解説します。

テーマと画面遷移の方針

テーマはMaterial Design 3のテーマを使用し、すべてデフォルトのままにします。画面遷移はありません。

8.2 プロジェクトを作成する

プロジェクトを作成します。第1章の「1.1 プロジェクトの作成」で示した手順に従って、プロジェクトを作成してください。プロジェクト名は「hiragana_converter」としましょう。プロジェクト作成直後は**lib/main.dart**にテンプレートになるアプリコードが書かれていますので、まずは以下のように修正します。

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hiragana Converter',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
```

```
      ),
      home: const HomeScreen(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

テンプレートプロジェクトの不要なコードを削除したこの状態から作業を開始しましょう。

8.3 アプリで使用するパッケージを導入する

まずはじめに、アプリで使用するパッケージを一気に導入してしまいましょう。

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。

```
# httpパッケージを導入❶
$ flutter pub add http
# JSONのシリアライズ、デシリアライズを行うパッケージを導入❷
$ flutter pub add json_annotation
$ flutter pub add --dev json_serializable
# Riverpodを導入❸
$ flutter pub add flutter_riverpod riverpod_annotation
$ flutter pub add --dev riverpod_generator custom_lint riverpod_lint
# コード生成のためにbuild_runnerを導入❹
$ flutter pub add --dev build_runner
```

Web APIを呼び出すためにhttpというパッケージを導入します(❶)。このパッケージはHTTPリクエストを簡単に扱うことのできるパッケージです。

Web APIのリクエストとレスポンスはJSON形式でやりとりします。JSONを取り扱うため、json_annotationとjson_serializableパッケージを導入します

(❷)。これらのパッケージはDartのオブジェクトとMapの相互変換を行うコードを自動生成してくれるパッケージです。

そのほか、状態管理にRiverpodを採用するため❸を導入します。Riverpodとjson_serializableはコード生成を行うため、build_runnerパッケージも導入します(❹)。

riverpod_lintを設定する

第7章で紹介したriverpod_lintの設定を行います。riverpod_lintはcustom_lintパッケージを利用して実現しています。以下のように`analysis_options.yaml`へcustom_lintを有効化する記述を追加します。

```
./analysis_options.yaml
analyzer:
  plugins:
    - custom_lint
```

8.4 入力状態のウィジェットを実装する

第5章で作成したアプリはNavigation APIを使っていくつかの画面を行き来するアプリでした。本章のハンズオンは1つの画面で状態により表示を切り替えるように実装していきます。上記の2つの違いは、プログラム上どちらが良いということはなく、ユーザーにどのような体験を提供するかによります。

レイアウトを作成する

まずはテキストを入力するレイアウトを作成します。`lib`フォルダの配下に`input_form.dart`という新しいファイルを追加し、以下のコードを記述します。

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

class InputForm extends StatefulWidget {
```

```
  const InputForm({super.key});

  @override
  State<InputForm> createState() => _InputFormState();
}

class _InputFormState extends State<InputForm> {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        /* ◆ Padding
        余白を与えて子要素を配置するWidget */
        Padding( ——❶
          padding: const EdgeInsets.symmetric(horizontal: 16),
          child: TextField(                          ┐
            maxLines: 5,                             │
            decoration: const InputDecoration(       ├──❷
              hintText: '文章を入力してください',      │
            ),                                       │
          ),                                         ┘
        ),
        /* ◆ SizedBox
        サイズを指定できるWidget */
        const SizedBox(height: 20), ——❸
        ElevatedButton(                  ┐
          onPressed: () {},              │
          child: const Text(             ├──❹
            '変換',                      │
          ),                             │
        ),                               ┘
      ],
    );
  }
}
```

StatefulWidgetを継承した**InputForm**クラスを実装しました。**TextField**ウィジェット(❷)と**ElevatedButton**ウィジェット(❹)を**Column**ウィジェットで囲い、垂直にレイアウトしました。

TextFieldウィジェットはユーザーが入力可能なテキストフィールドです。**maxLines**プロパティは一度に表示する行数の指定です。今回は5を設定しましたので5行分のテキストを表示し、それ以上入力するとスクロールします。**decoration**プロパティは**TextField**ウィジェットのさまざまな装飾を指定するプロパティです。今回は未入力の場合に「文章を入力してください」と表示

するヒントテキストを指定しました。

TextFieldウィジェットが横いっぱいに広がらないように、Paddingウィジェットで余白を設けました(❶)。現時点ではPaddingウィジェットにconstantコンストラクタを使うようワーニングが表示されますが、のちほど修正します。

❸のSizedBoxウィジェットはTextFieldウィジェット(❷)とElevatedButtonウィジェット(❹)の間に余白を設けるために使用しています。

レイアウトを表示する

このままでは、アプリを実行してもInputFormウィジェットは表示されません。main.dartを以下のように修正します。

./lib/main.dart

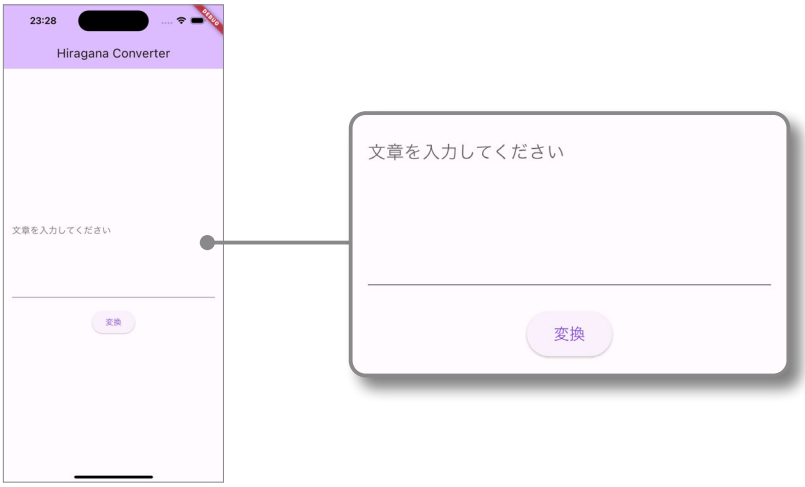
```
import 'package:flutter/material.dart';
import 'package:hiragana_converter/input_form.dart';  ❶

// 省略

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: const Text('Hiragana Converter'),
      ),
      body: const InputForm(),
    );
  }
}
```

❷

先ほど実装したInputFormウィジェットを参照するため、input_form.dartをインポートしました(❶)。HomeScreen画面ではScaffoldウィジェットを返すように修正しました(❷)。bodyには先ほど作成したInputFormウィジェットを指定しています。

これでアプリを実行すると、**図8.5**のようにInputFormウィジェットが表示されます。

図8.5 InputForm Widgetの表示（図8.2再掲）



入力値のバリデーションを行う

TextFieldウィジェットに与えられた文字列が空でないことをバリデーションします。こういったケースではFormウィジェットとFormFieldウィジェットを組み合わせて使うと便利です。

./lib/input_form.dart

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

class InputForm extends StatefulWidget {
  const InputForm({super.key});

  @override
  State<InputForm> createState() => _InputFormState();
}

class _InputFormState extends State<InputForm> {

  final _formKey = GlobalKey<FormState>(); ——❶

  @override
  Widget build(BuildContext context) {
    /* ◆ Form
    TextFormFieldやFormFieldを
    グループ化して管理するWidget */
    return Form( ——❷
```

```
      key: _formKey,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16),
            /* ◆ TextFormField
            テキスト入力フォームを実現するWidget */
            child: TextFormField( ──❸
              maxLines: 5,
              decoration: const InputDecoration(
                hintText: '文章を入力してください',
              ),
              validator: (value) {                    ┐
                if (value == null || value.isEmpty) {  │
                  return '文章が入力されていません';      ├─❹
                }                                       │
                return null;                            │
              },                                        ┘
            ),
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              final formState = _formKey.currentState!;
              formState.validate(); ──❺
            },
            child: const Text(
              '変換',
            ),
          ),
        ],
      ),
    );
  }
}
```

_InputFormStateクラスのbuildメソッドで返すウィジェット全体をFormウィジェットで包み(❷)、Formウィジェットのkeyプロパティには(❶)で用意したGlobalKeyを渡しています。また、TextFieldウィジェットはFormFieldウィジェットのサブクラスであるTextFormFieldウィジェットに置き換えました(❸)。TextFormFieldウィジェットはconstantコンストラクタを持ちません。よって、先ほどまでPaddingウィジェットのconstantコンストラクタを呼び出す旨のワーニングは解消されます。

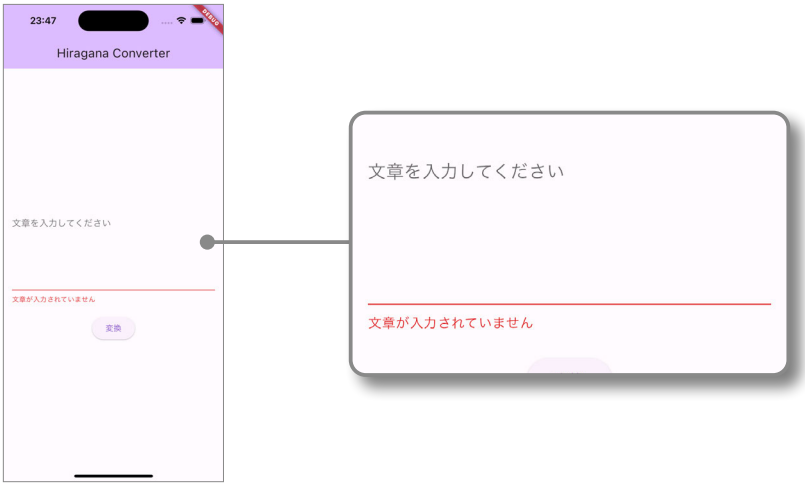
Formウィジェットに関する操作はFormウィジェットのStateを経由して行

います（Formウィジェットは StatefulWidget です）。ElevatedButtonウィジェットのonPressedコールバックで、GlobalKeyからFormウィジェットのStateを取得しvalidateメソッドを呼び出しています（❺）。validateメソッドが呼び出されると、Formウィジェットの子孫にあるFormFieldウィジェットでバリデーションが行われます。ちょうど、TextFieldウィジェットをTextFormFieldウィジェットに書き換えたところでした。

TextFormFieldウィジェットはvalidatorコールバックで文字の空チェックを行っています（❹）。

以上のようにFormウィジェット、FormFieldウィジェットを用いてバリデーション機能を実装できます（**図8.6**）。

図8.6　**バリデーションが機能している様子**



8.5 入力文字を取得する

バリデーションを通過した入力文字をWeb APIのパラメータとして扱えるように取得します。方法はいくつかありますが、今回はTextEditingControllerを利用します。

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