先ほどのサンプルで、FloatingActionButtonウィジェットのonPressedイベントではwatchを用いずにreadメソッドを利用していたのはそのためです。

Providerの値をフィルタする

頻繁に値が更新され、ウィジェットの再構築が不必要に発生する場合はselectメソッドで値をフィルタすることもできます。たとえば、以下のようなPointクラスを提供するProviderがあるとします。

```
class Point {
  Point(this.x, this.y);

  int x;
  int y;
}
```

ウィジェットではxの値のみが必要な場合、selectメソッドを利用してxの値のみを取得することができます。

```
final x = ref.watch(pointProvider.select((point) => point.x));
```

このようにselectメソッドを利用すると、yの値が変化しても値は通知されません。ウィジェットの再構築を抑制したい場合などに便利です。

Providerのライフサイクル

コード生成を利用した場合、Providerは購読されなくなると自動的に破棄されます。たとえば、あるダイアログの状態を管理するProviderは、ダイアログが閉じられると破棄され、再度開かれた際には状態がリセットされていることになります。

ただ、時にはアプリの起動中は状態を保持したいケースや、複数の画面をまたいで状態を共有したいケースもあるでしょう。そのような場合はProviderを自動で破棄させないようにすることも可能です。

```
@Riverpod(keepAlive: true)
class CounterNotifier extends _$CounterNotifier {
  @override
  int build() => 0;

  void increment() {
    state = state + 1;
```

```
  }
}
```

@riverpodアノテーションに代えて、@Riverpodにします。大文字から始まるアノテーションを利用し、keepAliveプロパティにtrueを設定します。このようにすることで、Providerは自動的に破棄されなくなります。

Providerを任意のタイミングで再構築したい場合は、refreshメソッドを利用します。

```
ref.refresh(counterNotifierProvider);
```

Providerにパラメータを渡す

Providerにパラメータを渡す方法を解説します。関数ベースのProviderの場合は、第二引数以降にパラメータを記述します。

関数ベースのProviderは以下のようになります。

```
@riverpod
String greet(GreetRef ref, String str) {
  return 'Hello $str';
}
```

Providerにアクセスする際は以下のようにパラメータを渡します。

```
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final greet = ref.watch(greetProvider('Flutter'));
```

クラスベースのProviderの場合は、buildメソッドの引数にパラメータを記述します。

```
@riverpod
class CounterNotifier extends _$CounterNotifier {
  @override
  int build(int num) {
    return num;
  }

  void increment() async {
    state = state + 1;
  }
}
```

こちらもProviderにアクセスする際は以下のようにパラメータを渡します。

```
class MyHomePage extends ConsumerWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final counterNotifier = counterNotifierProvider(3); ——❶
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '${ref.watch(counterNotifier)}', ——❷
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          ref.read(counterNotifier.notifier).increment(); ——❸
        },
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

CounterNotifierの例では、値を監視するためにProviderにアクセスする❷と、カウンタをインクリメントするためにProviderにアクセスする❸がありますので、❶でプロバイダに初期値を渡し変数に置きました。

7.5 まとめ

本章ではFlutterの状態管理の概念と、Riverpodの使い方を解説しました。

Riverpodの役割を理解するために、冒頭でFlutterにおける状態管理について解説しました。Riverpodは強力なパッケージですが、初学者には概要がつかみづらいので(筆者自身がそうでした)主要なクラスの役割と関係性、関連パッケージを先に紹介し、Riverpodの全体像をつかんでもらうように解説を進めました。

Riverpodは活発に開発が行われており、本章で紹介できなかった機能もあります。より理解を深めるうえでは、村松龍之介さんの電子書籍『Flutter x Riverpodでアプリ開発！実践入門』[注11]がお勧めです。Riverpodの更新に追従しながら、新しい情報を、日本語で詳しく解説された貴重な書籍です。

次章では、Riverpodを状態管理に利用したアプリを実装していきます。

注11 https://zenn.dev/riscait/books/flutter-riverpod-practical-introduction

第8章

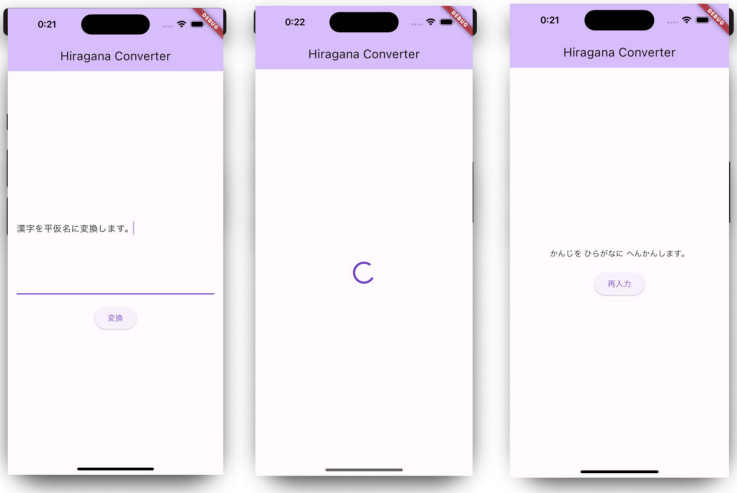
実践ハンズオン②

ひらがな変換アプリを開発

本章では再びハンズオン形式でアプリの実装に挑戦します。入力したテキストをひらがなに変換するアプリです。第7章で学んだRiverpodを採用し、状態管理を行います。なお、本章でもfvmコマンドを省略してflutterコマンドを記載しています。ご自身の環境、コマンドを実行するディレクトリにあわせて読み替えてください。

図8.1が完成イメージです。

図8.1　**アプリの完成イメージ**



8.1 開発するアプリの概要

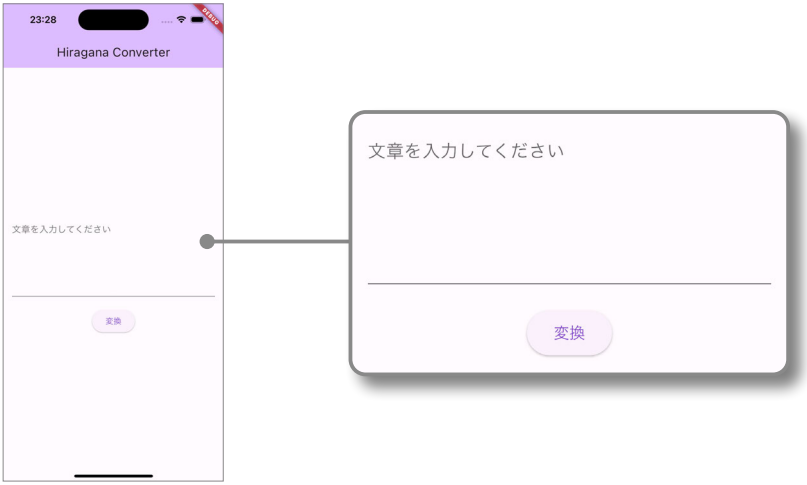
このハンズオンで実装するアプリの概要を説明します。公開されたWeb APIを利用し、入力したテキストをひらがなに変換します。文字入力とバリデーション、Web APIのリクエストにJSONの取り扱いと実用的な機能を盛り込んでいます。

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ます。入力状態で変換ボタンをタップし(図8.1の左)、APIリクエストを実行するとインジケータを表示します(図8.1の中央)。APIのレスポンスが返ると結果を表示します(図8.1の右)。再入力ボタンをタップすると、再び入力状態に戻ります。

入力状態

テキストを入力する状態です（**図8.2**）。テキストはバリデーションチェックを行い、空文字の場合はメッセージを表示します。「変換」ボタンをタップすると、ひらがな変換のリクエストを行います。

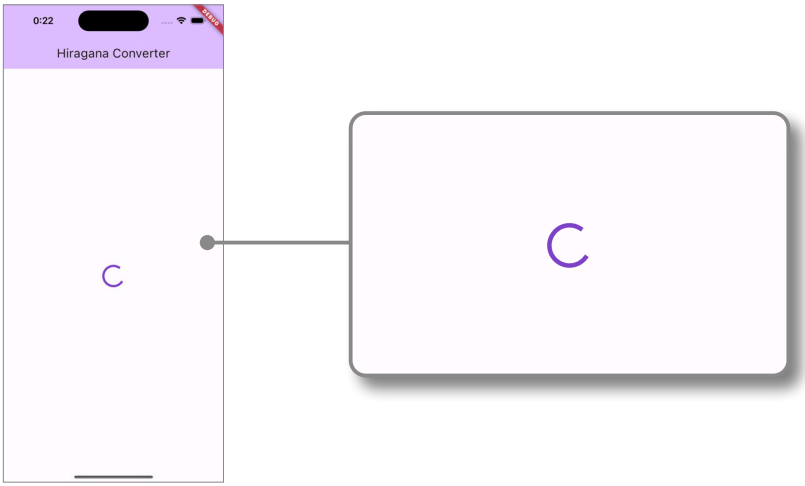
図8.2　**入力状態**



レスポンス待ち状態

Web APIのレスポンス待ちの状態です（**図8.3**）。インジケータを表示することで、ユーザーにリクエスト中であることを伝えます。

図8.3 **レスポンス待ちの状態**



変換完了状態

APIリクエストが完了し、変換結果のある状態です(**図8.4**)。「再入力」ボタンをタップすると、再び入力状態に戻ります。

図8.4 **変換完了状態**