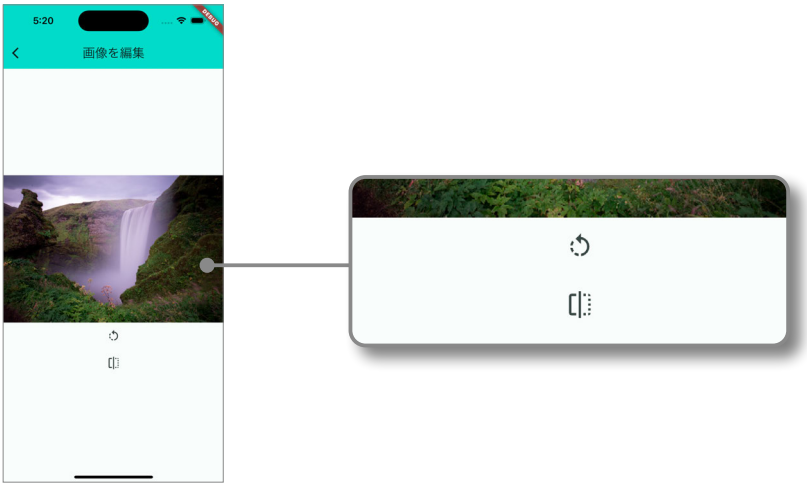
図6.10 画像編集画面



アイコンを追加する —— flutter_genによるアセット管理

先ほどIconButtonウィジェットにFlutterフレームワーク組み込みのアイコンを設定しましたが、アプリ独自のアイコンに置き換えてみましょう。

アセットを扱うためにflutter_genを利用します。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。

```
# build_runnerパッケージとflutter_gen_runnerパッケージを導入
$ flutter pub add --dev build_runner flutter_gen_runner
# flutter_svgパッケージを導入
$ flutter pub add flutter_svg
```

flutter_genを使用するためにflutter_gen_runnerとbuild_runnerパッケージを導入しました。SVGファイルを扱うためにflutter_svgパッケージも導入しました。

flutter_genでSVGファイルを有効にするため、pubspec.yamlに以下の設定を追加します。

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

続いてプロジェクトに独自のアイコンをアセットとして追加します。今回

はiconmonstrというサイトからアイコンを利用させてもらいます。

・https://iconmonstr.com/

イメージを検索して、SVGファイルをダウンロードします。今回は「Refresh - Lined」と「Layout Vertical - Lined」というアイコンをダウンロードし、それぞれrotate_icon.svgとflip_icon.svgという名前で保存します。

ダウンロードしたファイルを以下のようにassetsフォルダに配置します。

```
~/project_root
└── assets
    ├── flip_icon.svg
    └── rotate_icon.svg
```

pubspec.yamlにて、assetsキーを追加します。

./pubspec.yaml
```
# 省略

flutter:
  # 省略
  assets:
    - assets/
```

アセットの追加が完了したら、以下のコマンドを実行してアイコンを扱うコードを生成します。

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

それでは追加したアイコンをコードに反映させましょう。

./lib/edit_snap_screen.dart
```
import 'package:edit_snap/gen/assets.gen.dart'; ——❶
// 省略
class _ImageEditScreenState extends State<ImageEditScreen> {

  @override
  Widget build(BuildContext context) {
    final l10n = L10n.of(context);
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(l10n.imageEditScreenTitle),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
```

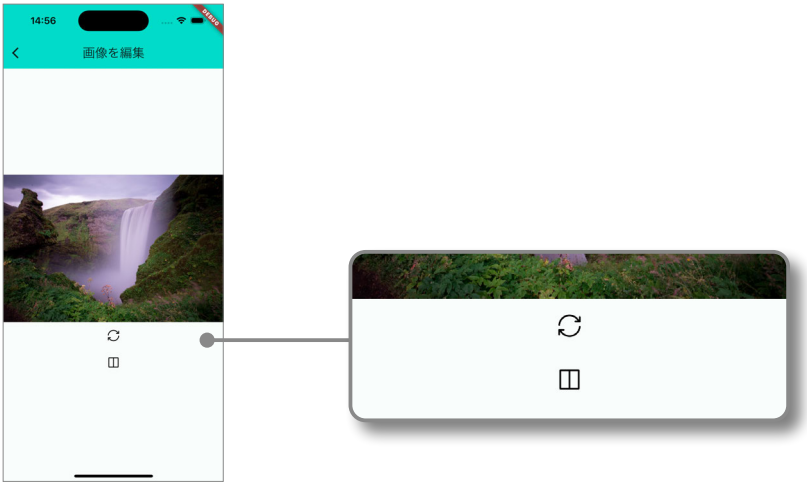
```
          children: [
            Image.memory(widget.imageBitmap),
            IconButton(
              onPressed: () {},
              icon: Assets.rotateIcon.svg(
                width: 24,
                height: 24,
              ),
            ),
            IconButton(
              onPressed: () {},
              icon: Assets.flipIcon.svg(
                width: 24,
                height: 24,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

❷

❸

flutter_genが生成したコードをインポートし(❶)、IconButtonウィジェットのiconプロパティにflutter_genで生成したコードからアイコンを設定しました(❷、❸)。アプリを実行してアイコンが変更されていることを確認しましょう(**図6.11**)。

図6.11 **アイコンを変更した画像編集画面**



画像を編集する処理を実装する

画像を回転、反転させる処理を実装します。

./lib/edit_snap_screen.dart

```
// 省略
import 'package:image/image.dart' as image_lib; ——❶

// 省略

class _ImageEditScreenState extends State<ImageEditScreen> {

  late Uint8List _imageBitmap; ——❷

  @override
  void initState() {
    super.initState();
    _imageBitmap = widget.imageBitmap;
  }                                          ——❸

  void _rotateImage() { ——❹
    // 画像データをデコードする
    final image = image_lib.decodeImage(_imageBitmap);
    if (image == null) return;
    // 画像を時計回りに90° 回転する
    final rotateImage = image_lib.copyRotate(image, angle: 90);
    // 画像をエンコードして状態を更新する
    setState(() {
      _imageBitmap = image_lib.encodeBmp(rotateImage);
    });
  }

  void _flipImage() { ——❺
    // 画像データをデコードする
    final image = image_lib.decodeImage(_imageBitmap);
    if (image == null) return;
    // 画像を水平方向に反転する
    final flipImage = image_lib.copyFlip(
      image,
      direction: image_lib.FlipDirection.horizontal,
    );
    // 画像をエンコードして状態を更新する
    setState(() {
      _imageBitmap = image_lib.encodeBmp(flipImage);
    });
  }
```

```
  @override
  Widget build(BuildContext context) {
    final l10n = L10n.of(context);
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(l10n.imageEditScreenTitle),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.memory(_imageBitmap), ——❻
            IconButton(
              onPressed: () => _rotateImage(), ——❼
              icon: Assets.rotateIcon.svg(
                width: 24,
                height: 24,
              ),
            ),
            IconButton(
              onPressed: () => _flipImage(), ——❽
              icon: Assets.flipIcon.svg(
                width: 24,
                height: 24,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

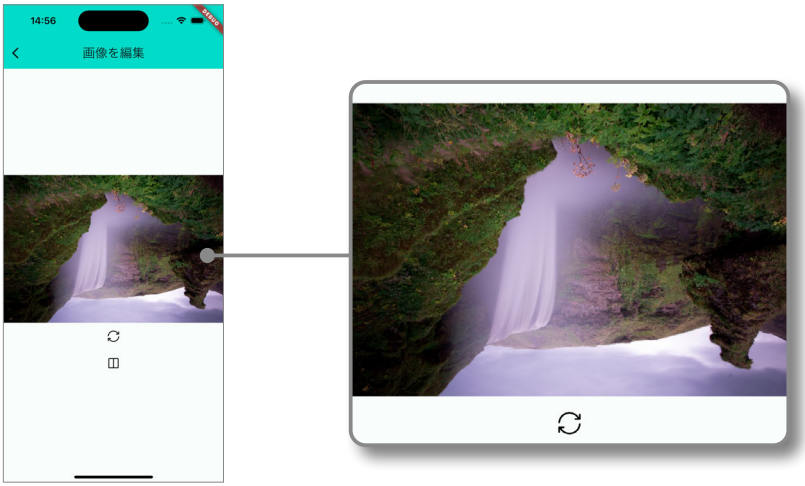
画像の加工処理を実装するためにimageパッケージをインポートしました(❶)。ここでもImageウィジェットと名前が競合するため、asキーワードに続けてimage_libという別名を付けています。

ウィジェットのStateクラスである_ImageEditScreenStateに、画像のバイト列を格納するためのクラス変数_imageBitmapを定義しました(❷)。_imageBitmapの初期化はinitStateメソッドにてImageEditScreen画面のフィールドimageBitmapから画像のバイト列を取得することで行います(❸)。initStateメソッドはStatefulWidgetのライフサイクルメソッドの一つで、ウィジェットが生成されたときに一度だけ呼び出されるメソッドなのでStateクラスの初期化処理に適しています。

画像を回転させる処理(❹)と画像を反転させる処理(❺)はそれぞれimageパッケージを利用して実装しました。加工済みの画像データを`_imageBitmap`に格納し、`setState`メソッドを呼び出すことで画面を再描画します(❻)。それぞれのメソッドはボタンがタップされたときに呼び出すようにしました(❼、❽)。

これで画像を編集する処理が実装できました。アプリを実行し、画像を回転、反転させてみましょう(**図6.12**)。

図6.12　**画像を編集した様子**



6.8 まとめ

画像ライブラリから取得した画像を回転、反転させて編集するアプリを作成しました。画像の取得にはimage_pickerパッケージを、画像の加工にはimageパッケージを利用しました。

第4章で学んだアプリ開発の土台づくりを踏まえ、アプリをしっかりと日本語化し、アプリ内のメッセージはarbファイルに集約しました。アイコンをアセットとしてプロジェクトに追加し、flutter_genを利用して安全にアイコンを扱うコードを生成しました。

第5章で学んだ内容として、簡易的にテーマをアレンジすることにも挑戦

しました。画面遷移はシンプルな要件のため、Navigator 1.0のAPIで十分に対応できました。

Tips WidgetとStateのライフサイクルについて

_ImageEditScreenStateクラスでは、わざわざImageEditScreenウィジェットのフィールドimageBitmapから画像のバイト列を取得していました（❸）。この_imageBitmapの初期化処理は、以下のように_ImageEditScreenStateクラスのコンストラクタで受け取る実装も考えられます。

./lib/edit_snap_screen.dart
```
class _ImageEditScreenState extends State {

  _ImageEditScreenState(this._imageBitmap);

  final Uint8List _imageBitmap;
// 省略
```

しかし、このようにStateクラスのコンストラクタでデータを受け取ることは推奨されません。理由はウィジェットよりもStateのライフサイクルが長いためです。ウィジェットが異なるパラメータで再生成されたときに、Stateは再生成されずにそのまま使い回されることがあります。そのとき、ウィジェットとStateの間で状態の不一致が発生する可能性があります。詳しくは第9章をご覧ください。

wassyoi：t--sekiguchi@hotmail.co.jp