```
# imageパッケージとimage_pickerパッケージを導入
$ flutter pub add image image_picker
```

iOSネイティブの設定を行う

続いてiOSネイティブの設定を行います。画像ライブラリにアクセスするiOSアプリは、その用途を伝える説明文を記述する必要があります。**ios/Runner/Info.plist**を開き、**NSPhotoLibraryUsageDescription**キーの下に説明文を追加します。

./ios/Runner/Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
  <!-- 省略 -->
    <key>NSPhotoLibraryUsageDescription</key>
    <string>編集する画像を選択します。</string>
</dict>
</plist>
```

画像を取得する処理を実装する

それでは画像を取得する処理を実装します。

./lib/image_select_screen.dart

```
import 'package:flutter/foundation.dart'; ——❶
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:image_picker/image_picker.dart'; ——❷
import 'package:image/image.dart' as image_lib; ——❸

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

  @override
  State<ImageSelectScreen> createState() => _ImageSelectScreenState();
}

class _ImageSelectScreenState extends State<ImageSelectScreen> {

  /* ◆ ImagePicker
  image_pickerパッケージが提供するクラス
  画像ライブラリやカメラにアクセスする機能を持つ */
  final ImagePicker _picker = ImagePicker(); ——❹
```

```
  /* ◆ Uint8List
  8bit 符号なし整数のリスト */
  Uint8List? _imageBitmap; ——❺

  Future<void> _selectImage() async { ——❻
    /* ◆ XFile
    ファイルの抽象化クラス */
    // 画像を選択する
    final XFile? imageFile = await _picker.pickImage(source: ImageSource.gallery); —❼

    // ファイルオブジェクトから画像データを取得する
    final imageBitmap = await imageFile?.readAsBytes(); ——❽
    assert(imageBitmap != null);
    if (imageBitmap == null) return;

    // 画像データをデコードする
    final image = image_lib.decodeImage(imageBitmap); ——❾
    assert(image != null);
    if (image == null) return;

    /* ◆ Image
    画像データとメタデータを内包したクラス */
    final image_lib.Image resizedImage;
    if (image.width > image.height) {
      // 横長の画像なら横幅を500pxにリサイズする
      resizedImage = image_lib.copyResize(image, width: 500);
    } else {
      // 縦長の画像なら縦幅を500pxにリサイズする
      resizedImage = image_lib.copyResize(image, height: 500);
    }                                                          ——❿

    // 画像をエンコードして状態を更新する
    setState(() {
      _imageBitmap = image_lib.encodeBmp(resizedImage);        —⓫
    });
  }

  @override
  Widget build(BuildContext context) {
// 省略
```

まず、必要なパッケージ群をインポートします。flutter/foundation.dartは、画像データとしてUint8List型を使用するためにインポートしました(❶)。image_picker/image_picker.dartは画像ライブラリへのアクセスするパッケージ、image/image.dartは画像データを扱うパッケージを使用するためにインポートしました(❷、❸)。imageパッケージのImageクラスは、画像を表示

するImageウィジェットと名前が競合します。そのためasキーワードに続けてimage_libという別名を付けています(❸)。こうするとimage_lib.Imageと記述するとimageパッケージのImageクラスを参照できます。

_ImageSelectScreenStateクラスでは、画像を取得するためにパッケージimage_pickerが提供するImagePickerクラスをインスタンス化しました(❹)。このクラスは画像ライブラリやカメラへアクセスする機能を提供します。実際の画像選択処理は_selectImageメソッドで実装しました(❻)。

ImagePickerクラスにて画像を取得すると、画像データはXFileというファイルを抽象化したクラスで返されます(❼)。XFileクラスから画像のバイト列を取得し(❽)、imageパッケージのdecodeImageメソッドで画像データをデコードします(❾)。iOS Simulatorは初期状態で画像ライブラリにいくつか写真が登録されていますが、これらの画像はサイズが大きいため、そのままでは表示に時間がかかる場合があります。今回は❿でリサイズして扱いやすくしています。

最後に画像データをバイト列に戻し、状態を更新します(⓫)。

画像取得処理をWidgetに組み込む

前項で実装した画像取得処理をWidgetと組み合わせてみましょう。

./lib/image_select_screen.dart

```
// 省略

class _ImageSelectScreenState extends State<ImageSelectScreen> {

  final ImagePicker _picker = ImagePicker();
  Uint8List? _imageBitmap;

  Future<void> _selectImage() async {
    // 省略
  }

  @override
  Widget build(BuildContext context) {
    final l10n = L10n.of(context);
    final imageBitmap = _imageBitmap; ——❶
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(l10n.imageSelectScreenTitle),
```

```
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            if (imageBitmap != null) Image.memory(imageBitmap), ——❷
            ElevatedButton( //  「画像を選ぶ」ボタン
              onPressed: () => _selectImage(), ——❸
              child: Text(l10n.imageSelect),
            ),
            if (imageBitmap != null) ——❹
              ElevatedButton( //  「画像を編集する」ボタン
                onPressed: () {
                },
                child: Text(l10n.imageEdit),
              ),
          ],
        ),
      ),
    );
  }
```

buildメソッドでは、画像のバイト列を変数imageBitmapに格納しました(❶)。クラス変数の_imageBitmapはnull許容型です。一時変数に置き、if文でnullチェックをすることでタイププロモーションが働き、非null許与型のように扱えるようになります(❷)。

imageBitmapがnullでない、すなわち画像が選択されたあとであれば画像を表示し(❸)、「画像を編集する」ボタンを表示します(❹)。

動作を確認してみましょう。画像を選択すると**図6.9**のように画像が表示されます。

図6.9 画像選択後の画像選択画面

6.7 画像編集画面を作成する

画像を回転、反転させる編集画面を作成します。

メッセージを追加する

まず編集画面で使用する文字列をarbファイルに追加します。

./lib/l10n/app_ja.arb

```
{
  // 省略
  // （一つ上の行の末尾にカンマを追加してください）
  "imageEditScreenTitle": "画像を編集"
}
```

arbファイルに追加したらコードジェネレータを実行します。

```
$ flutter gen-l10n
```

レイアウトを作成する

次にコードを記述するファイルを作成します。ファイル名はedit_snap_screen.dartとしましょう。画像編集画面もStatefulWidgetを継承したクラスで実装します。

./lib/edit_snap_screen.dart

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class ImageEditScreen extends StatefulWidget {
  const ImageEditScreen({super.key, required this.imageBitmap});

  final Uint8List imageBitmap; ——❶

  @override
  State<ImageEditScreen> createState() => _ImageEditScreenState();
}

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), ——❷
            /* ◆ IconButton
            アイコンを表示するボタン */
            IconButton(
              onPressed: () {},
              icon: const Icon(Icons.rotate_left), // フレームワーク組み込みのアイコンを設定
            ),
            IconButton(
              onPressed: () {},
              icon: const Icon(Icons.flip), // フレームワーク組み込みのアイコンを設定
            ),
```

❸

```
        ],
      ),
    ),
  );
 }
}
```

表示する画像はウィジェットのコンストラクタで受け取るようにしました(❶)。レイアウトは上部にAppBarウィジェットがあり、画像を表示するImageウィジェット(❷)とIconButtonウィジェット(❸)が垂直方向に並びます。IconButtonウィジェットにはFlutterフレームワーク組み込みのアイコンを設定しました。

画像編集画面への遷移を実装する

続いて、画像編集画面に遷移する処理を実装します。

./lib/image_select_screen.dart

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:edit_snap/edit_snap_screen.dart'; ──❶

// 省略

class _ImageSelectScreenState extends State<ImageSelectScreen> {

  // 省略

  @override
  Widget build(BuildContext context) {
    final l10n = L10n.of(context);
    final imageBitmap = _imageBitmap;
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(l10n.imageSelectScreenTitle),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            if (imageBitmap != null) Image.memory(imageBitmap),
            ElevatedButton( // 「画像を選ぶ」ボタン
```

```
              onPressed: () => _selectImage(),
              child: Text(l10n.imageSelect),
            ),
            if (imageBitmap != null)
              ElevatedButton( // 「画像を編集する」ボタン
                onPressed: () {
                  Navigator.of(context).push(                    ┐
                    MaterialPageRoute(                           │
                      builder: (context) => ImageEditScreen(     │
                        imageBitmap: imageBitmap,                ├ ❷
                      ),                                         │
                    ),                                           │
                  );                                             ┘
                },
                child: Text(l10n.imageEdit),
              ),
          ],
        ),
      ),
    );
  }
}
```

ImageEditScreen画面を参照するためimage_select_screen.dartをインポートしました(❶)。「画像を編集する」ボタンのonPressedコールバックで画像編集画面に遷移する処理を実装しました(❷)。こちらもNavigator 1.0のAPIを採用し、MaterialPageRouteクラスを使用した画面遷移です。

これで画像編集画面に遷移する処理が実装できました。アプリを実行し、画像編集画面へ遷移させてみましょう。**図6.10**のように表示されます。