

```

const MyApp({super.key});

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      // This is the theme of your application.
      //
      // TRY THIS: Try running your application with "flutter run". You'll see
      // the application has a purple toolbar. Then, without quitting the app,
      // try changing the seedColor in the colorScheme below to Colors.green
      // and then invoke "hot reload" (save your changes or press the "hot
      // reload" button in a Flutter-supported IDE, or press "r" if you used
      // the command line to start the app).
      //
      // Notice that the counter didn't reset back to zero; the application
      // state is not lost during the reload. To reset the state, use hot
      // restart instead.
      //
      // This works for code too, not just values: Most code changes can be
      // tested with just a hot reload.
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const MyHomePage(title: 'Flutter Demo Home Page'),
  );
}

```

①でThemeDataクラスのインスタンスを作成し、MaterialAppウィジェットに渡しています。英語のコメントにもあるように、カラーを変更して動作を確認してみましょう。

今回は②のdeepPurpleをblueに変更してください。ソースコードを保存すると、ホットリロード機能でアプリの外観が変化します。もし、変化しなければAndroid Studioのホットリロードボタンをクリックしてみましょう。

アプリの外観が紫を基調としたテーマから青に変化したことが確認できると思います。ThemeDataクラスはアプリのテーマ情報を持つクラスです。ThemeDataクラスの代表的なプロパティに、色のパラメータを持つcolorSchemeと、文字のパラメータを持つtextThemeがあります。

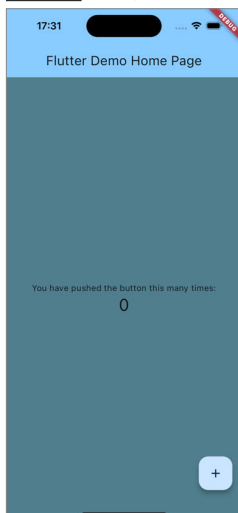
ColorSchemeクラスはマテリアルデザインのルールにのっとり、テーマの色のパラメータを計算します。さらに、計算済みの色を一部カスタマイズす

ることができます。

```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Flutter Demo',  
      // theme: ThemeData(  
      //   colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),  
      // ),  
      theme: ThemeData(  
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue)  
          .copyWith(background: Colors.blueGrey), —❶  
        useMaterial3: true,  
      ),  
      home: const MyHomePage(title: 'Flutter Demo Home Page'),  
    );  
  }  
}
```

ColorSchemeクラスのcopyWithメソッドを使い、カラーを変更したコピーを作成します。この例では背景カラーを変更しています(❶、図5.1)。

図 5.1 background カラーが変更された様子



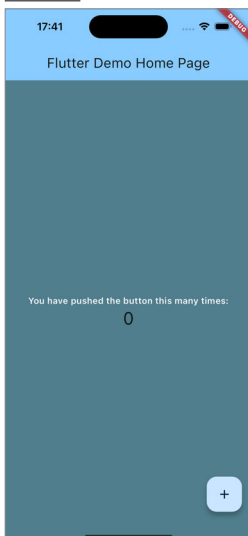
続いてtextThemeについて見てみましょう。

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue)
          .copyWith(background: Colors.blueGrey),
        textTheme: const TextTheme(
          bodyMedium: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.w600,
          ),
        ),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

ThemeData クラスへTextThemeを渡しました(❶)。ここではbodyMediumというテキストスタイルの色とフォントウェイトを変更しています(図5.2)。

図5.2 TextThemeが変更された様子



アプリのデザインを細かくカスタマイズするには、`ColorScheme`のリファレンス^{注1}と`TextTheme`のリファレンス^{注2}、それと併せてマテリアルデザインのドキュメント^{注3}を参照してください。

また最後に`useMaterial3`というコンストラクタのパラメータについても触れておきます(②)。このパラメータはMaterial Design 3(以降、M3)のテーマを利用するかどうかを指定します。M3はGoogleが提唱するマテリアルデザインの新しいバージョンで、従来のものよりも表現力が豊かでアクセシビリティが高いデザインとなっています。


M3はオプトインの形で段階的に導入されてきましたが、Flutter 3.16をもってM3がデフォルトになりました。今後は`useMaterial3`フラグは削除され、従来までのMaterial Design 2のコードは削除される予定です。

ダークモード対応

ここ数年のiOSやAndroidはユーザー設定や時間帯に応じて暗い外観に切り替わるダークモード機能を持っています(iOSはダークモード、Androidではダークテーマと言いますが、本書では両方を指してダークモードと呼ぶこととします)。MaterialApp ウィジェットやThemeDataクラスを使うことで簡単にダークモードに対応できます。サンプルをお見せしましょう。

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      darkTheme: ThemeData(
        colorSchemeSeed: Colors.deepPurple,
        brightness: Brightness.dark, ②
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



注1 <https://api.flutter.dev/flutter/material/ColorScheme-class.html>

注2 <https://api.flutter.dev/flutter/material/TextTheme-class.html>

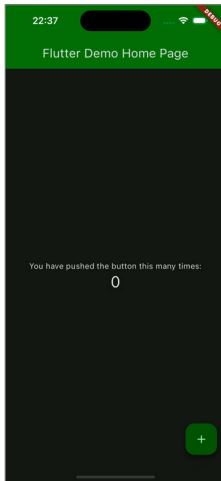
注3 <https://m3.material.io/styles/color/the-color-system/color-roles>

```
);
}
}
```

MaterialApp ウィジェットの **darkTheme** パラメータにダークモード用の **ThemeData** クラスを渡します(❶)。**ThemeData** クラスは **brightness** に **Brightness.dark** を指定することで、ダークモード用のテーマを自動計算してくれます(❷)。**MaterialApp** ウィジェットがシステムのダークモード設定を監視しているので、スマートフォンがダークモードに切り替わるとアプリの外観もダークモードに変化します。

それでは実際にアプリを実行し、ダークモードを切り替えてみましょう。iOS Simulator の場合は、設定アプリから「Developer」⇒「Dark Appearance」の順に選択し、スイッチでモードを切り替えます(図5.3)。

図5.3 モード切り替えてアプリの外観が変化する様子



アプリ独自のテーマ管理

マテリアルデザインにのっとったテーマについては、**MaterialApp** ウィジェットや **ThemeData** クラスを用いることで実現できることがわかりました。一方で、アプリ独自のテーマを管理する方法として **Theme Extension** があります。

```
class MyTheme extends ThemeExtension<MyTheme> {
  const MyTheme({
```

```

    required this.themeColor,
  });

  final Color? themeColor; ❶

  @override
  MyTheme copyWith({Color? themeColor}) {
    return MyTheme(
      themeColor: themeColor ?? this.themeColor, ❷
    );
  }

  @override
  MyTheme lerp(MyTheme? other, double t) {
    if (other is! MyTheme) {
      return this;
    }
    return MyTheme(
      themeColor: Color.lerp(themeColor, other.themeColor, t), ❸
    );
  }
}

```

ThemeExtensionクラスを継承したMyThemeクラスを実装しました。MyThemeクラスではthemeColorというカラーを扱うことにします(❶)。ThemeExtensionは抽象クラスで、サブクラスでは2つのメソッドを実装しなければなりません。

❷のcopyWithメソッドは任意のフィールドを変更したコピーをインスタンス化するメソッドです。

❸のlerpメソッドはテーマの変化を線形補間するメソッドです。このメソッドを実装しておくことで、テーマ変更時にアニメーション処理されるようになります。たとえば、ダークモードへの切り替えタイミングが該当します。

こうして実装したMyThemeクラスはThemeDataクラスのパラメータに渡します。

```

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(

```

```

        colorSchemeSeed: Colors.deepPurple,
        extensions: const [MyTheme(themeColor: Color(0xFF0000FF))], —①
    ),
    darkTheme: ThemeData(
        colorSchemeSeed: Colors.deepPurple,
        brightness: Brightness.dark,
        extensions: const [MyTheme(themeColor: Color(0xFFFF0000))], —②
    ),

```

ThemeDataクラスのextensionsパラメータにThemeExtensionクラスを継承したMyThemeクラスのインスタンスを渡します(①)。extensionsパラメータはList型なので、複数のTheme Extensionを設定することも可能です。ThemeDataクラスのパラメータなので、ダークモード用に別のMyThemeクラスを指定することも容易です(②)。

続いて、MyThemeクラスのテーマを適用したウィジェットを実装します。ThemedWidgetという正方形を描画するウィジェットです。

```

./lib/main.dart
class ThemedWidget extends StatelessWidget {
    const ThemedWidget({super.key});

    @override
    Widget build(BuildContext context) {
        final themeData = Theme.of(context); —①
        final myTheme = themeData.extension<MyTheme>(!); —②
        final color = myTheme.themeColor;
        return Container(width: 100, height: 100, color: color);
    }
}

```

Theme ウィジェットのofメソッドを使い、ThemeDataクラスのインスタンスを取得します(①)。さらに、ThemeDataクラスのextensionメソッドを使い、MyThemeクラスのインスタンスを取得します(②)。

Theme というウィジェットがここで初めて登場しましたが、MaterialApp ウィジェットが内部で生成しているウィジェットで、ThemeDataクラスのインスタンスを持っています。Theme ウィジェットの子孫であれば、どのウィジェットもofメソッドを使ってThemeDataクラスのインスタンスを取得することができるのです。さらにこのThemedWidgetのように、buildメソッドの中でTheme ウィジェットのofメソッドを呼び出すと、テーマが変更されたときに再描画されるしくみも備わっています(詳しくは第9章で解説します)。このしくみのおかげで、ダークモードへの切り替え時に色がアニメーションする様子を確認できます。

Theme Extensionを利用したアプリのサンプル

最後に、Theme Extensionを利用したサンプルの全体を掲載します。このサンプルではシステムのダークテーマ設定を利用せず、アプリ独自にダークテーマ設定を持つようにしました。Theme Extensionを継承したクラスでlerpメソッドを実装したことにより、テーマ変更時に色がアニメーションする様子が確認できます。

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

class MyTheme extends ThemeExtension<MyTheme> {
  const MyTheme({
    required this.themeColor,
  });

  final Color? themeColor;

  @override
  MyTheme copyWith({Color? themeColor}) {
    return MyTheme(
      themeColor: themeColor ?? this.themeColor,
    );
  }

  @override
  MyTheme lerp(MyTheme? other, double t) {
    if (other is! MyTheme) {
      return this;
    }
    return MyTheme(
      themeColor: Color.lerp(themeColor, other.themeColor, t),
    );
  }
}

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

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

  @override
  State<StatefulWidget> createState() => _MyAppState();
}
```