

```

    );
  }

class Counter extends StatefulWidget {

    const Counter({super.key});

    @override
    State<Counter> createState() => _CounterState();
  }

class _CounterState extends State<Counter> {

    int _count = 0;

    @override
    Widget build(BuildContext context) { — ❷
      return GestureDetector(
        onTap: () {
          print('tapped!');
        },
        child: Container(
          color: Colors.red,
          width: 100,
          height: 100,
          child: Center(
            child: Text(
              '$_count',
              textDirection: TextDirection.ltr,
            ),
          ),
        ),
      );
    }
  }
}

```

Counter ウィジェットを StatefulWidget に書き換えました。StatefulWidget は build メソッドを持ちません。代わりに createState メソッドをオーバーライドし State オブジェクトを返します (❶)。build メソッドは State クラスで実装します (❷)。State クラスは状態が変化したことをフレームワークに知らせる setState() というメソッドを持っており、このメソッドを呼び出すと build メソッドが呼び出されるしくみになっています。

## Widgetの状態を変化させる

それでは`_count`の値を変化させ、それに追従して`build`メソッドが呼び出されるように修正してみます。

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

void main() {
  runApp(
    const Center(
      child: Counter(),
    ),
  );
}

class Counter extends StatefulWidget {

  const Counter({super.key});

  @override
  State<Counter> createState() => _CounterState();
}

class _CounterState extends State<Counter> {

  int _count = 0;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        print('tapped!');
        setState(() {
          _count += 1;
        });
      },
      child: Container(
        color: Colors.red,
        width: 100,
        height: 100,
        child: Center(
          child: Text(
            '$_count',
            textDirection: TextDirection.ltr,
          ),
        ),
      ),
    );
  }
}
```

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

onTapに渡すコールバックの中で、setStateの呼び出しと\_countの変更を実装しました(❶)。setStateは引数にクロージャで\_countの変更処理を渡しています。このように、状態を変更するときはsetStateの引数で行います。

このサンプルコードを実行すると、赤い四角形をタップするたびに数字が更新されカウントアップします。

## 3.4

### まとめ

StatelessWidgetとStatefulWidgetについて、それぞれの特徴をあらためておさらいします。

#### • StatelessWidgetの特徴

- 状態を持たない
- buildメソッドをオーバーライドし、1つ以上のWidgetを組み合わせてUIを構成する
- 自身で表示更新するしくみがない

#### • StatefulWidgetの特徴

- StatefulWidgetはStateを生成する
- StatelessWidgetにあったbuildメソッドはStateで実装する
- 状態を変化させるときはsetStateの引数コールバック内で行う
- setStateを呼び出すと自身の表示更新が行われる

シンプルなアプリであれば、StatelessWidgetとStatefulWidgetの組み合わせだけで開発することができます。アプリが複雑になり、ウィジェットの更新や状態の受け渡しに課題が見えてきたときは、状態管理について検討するといでしょう。第7章でその考え方や代表的な手法を解説します。



第 4 章

アプリの日本語化対応、  
アセット管理、環境変数

本章では国際化対応、アイコンなどのアセット管理のしくみ、環境変数の取り扱いについて解説します。特に製品レベルのアプリを開発する際には、はじめに整えておくべき要素であり、さしずめ「開発の土台作り」と言えます。

また、これらの要素を整えるために必要なパッケージ管理についても併せて解説します。

## 4.1

### パッケージやツールを導入する

Dart言語は標準でパッケージ管理ツールを提供しており、Flutterのプロジェクトで利用することができます。まず、混乱のないよう用語の解説を表4.1にまとめたので確認してください。

表4.1 パッケージに関する用語

用語	解説
パッケージ	Dartのプログラムライブラリ、アプリ、リソースなどを含んだディレクトリ。パッケージそのものや依存関係を記述したpubspecファイルが必ず含まれる
プラグイン	ネイティブコード(iOS向けのSwiftコードやAndroid向けのKotlinコードなど)を同梱したパッケージ
pub	パッケージ管理ツール。パッケージを入手するために使う
pub.dev	共有パッケージを閲覧、検索できるWebサイト。 <a href="https://pub.dev">https://pub.dev</a>
Flutter Favorite Program	「最初に導入を検討すべきパッケージ」を選出する活動のこと。pub.devで公開されている

pub.dev<sup>注1</sup>にはたくさんのパッケージが公開されています。サードパーティのプログラムライブラリのみならず、Flutterが公式で提供するパッケージもあります。カメラ操作、データ永続化、Firebase連携など、pub.devでパッケージを検索することで多くのユースケースに対応できるでしょう。本章でも後述のアセット管理のところで便利なパッケージを紹介します。

多くのパッケージはpubspec.yamlに所定の記述をし、コマンドを実行することで導入できます。ただし、導入に必要な手順はパッケージにより異なりますので、必ずそれぞれのドキュメントを参照してください。


注1 <https://pub.dev>

## パッケージの導入方法

pubspec.yamlにはパッケージを記述するセクションが2種類あります。以下にサンプルを示します。

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  http: ^0.13.6

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
  build_runner: ^2.3.3
```



dependenciesセクションはアプリのコードが依存するパッケージを記述します(①)。dev\_dependenciesは開発フェーズでのみ利用するパッケージを記述します。たとえばテストに関わるパッケージや、コード生成ツールなどです(②)。

依存するパッケージを追加するには、YAML (YAML Ain't Markup Language) ファイルを直接編集するか、コマンドを実行します。たとえば、httpというパッケージを導入する場合は、以下のようにコマンドを実行するとdependenciesセクションにhttpパッケージが追加されます。

```
$ flutter pub add http
```

dev\_dependenciesセクションに追加する際は--devオプションを付与します。たとえば、build\_runnerというパッケージを導入する場合は、以下のようコマンドを実行します。

```
$ flutter pub add --dev build_runner
```

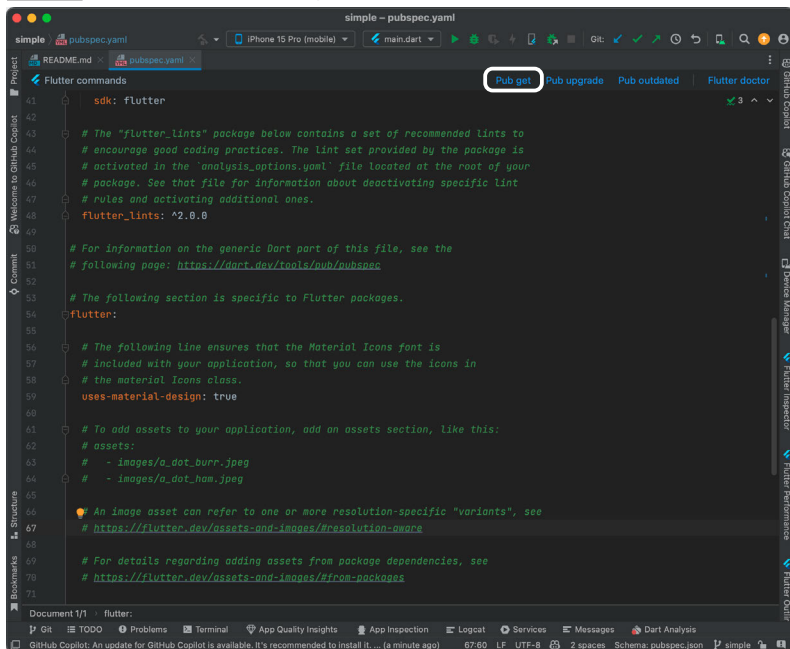
こうしてpubspec.yamlにパッケージを追加したら、コマンドを実行してパッケージを導入します。

```
$ flutter pub get
```

またはAndroid Studioにはpub getコマンドを実行するためのボタンが用意されています。pubspec.yamlを開いている状態で「Pub get」ボタンを押すとパ

パッケージを導入できます(図4.1)。

図4.1 Android Studioの「Pub get」ボタン



## パッケージバージョンの指定方法

パッケージのバージョン指定には次のような記述方法があります。

# 2.1.0以上、互換性のある限り最新のバージョンを利用する

shared\_preferences: ^2.1.0 —①

# 2.1.0以上 3.0.0未満のバージョンを利用する

shared\_preferences: '>=2.1.0 <3.0.0' —②

# 2.1.0以下のバージョンを利用する

shared\_preferences: '<=2.1.0' —③

# 2.0.0より新しいバージョンを利用する

shared\_preferences: '>2.0.0' —④

# バージョンを2.1.1に固定する

shared\_preferences: 2.1.1 —⑤