

```

void _incrementCounter() {
  setState(() {
    _counter++;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Home Screen'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const ColoredText(
            text: 'You have pushed the button\nthis many times:',
            color: Colors.blueGrey,
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ),
  );
}

class ColoredText extends StatelessWidget {
  const ColoredText({super.key, required this.text, required this.color});

  final String text;
  final Color color;

  @override
  Widget build(BuildContext context) {
    print('ColoredText build');
  }
}

```

①

②

③

```

return ColoredBox(
  color: color,
  child: Text(
    text,
    style: Theme.of(context).textTheme.headlineSmall,
  ),
);
}
}

```

`ColoredBox` ウィジェットを構築していた部分を、`ColoredText` ウィジェットとして切り出しました(❷)。`ColoredText` ウィジェットは `constant` コンストラクタを実装しています。また、`ColoredText` ウィジェットを生成している箇所は、`const` 修飾子を付与しています(❶)。これで、`ColoredText` ウィジェットは祖先の再構築の影響を受けなくなります。

このサンプルを実行すると、`FloatingActionButton` ウィジェットをタップしても `ColoredText` ウィジェットは再構築されず、❸で出力しているログはアプリ起動時の一度しか出力されません。

また、`ColoredText` ウィジェットは `constant` コンストラクタを実装しているため、状態の変化しないイミュータブルなクラスとして実装され副作用を持ちません。プログラムの堅牢性が高まります。

一方、以下のような実装方法も考えられますが、お勧めしません。ウィジェットをクラスではなく、ヘルパメソッドで実装する方法です。

```

class _MyHomeScreenState extends State<HomeScreen> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Screen'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [

```

```

        _colordText(
          text: 'You have pushed the button\nthis many times:',
          color: Colors.blueGrey,
        ),
        Text(
          '$_counter',
          style: Theme.of(context).textTheme.headlineMedium,
        ),
      ],
    ),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: const Icon(Icons.add),
  ),
);
}

Widget _colordText({required String text, required Color color}) {
  return ColoredBox(
    color: color,
    child: Text(
      text,
      style: Theme.of(context).textTheme.headlineSmall,
    ),
  );
}
}

```

この例では先ほどの `ColorText` ウィジェットを関数として実装しました (❶、❷)。同じ UI は実現可能ですが、ウィジェットの再構築を抑える効果はありません。

状態を末端のウィジェットに移す

状態を末端のウィジェットに移すことで、ウィジェットが再構築される範囲を小さくすることができます。タップすると数字がカウントアップするボタンを例に考えてみましょう。

```

import 'package:flutter/material.dart';

void main() {
  runApp(const MaterialApp(

```

```

    home: HomeScreen(),
  ));
}

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


  @override
  State createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  int _counter = 0;

  void _increment() {
    setState(() {
      _counter += 1;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('count = $_counter'),
          onPressed: () => _increment(),
        ),
      ),
    );
  }
}

```



画面中央のボタンがカウント数を表示しており、タップするとカウントアップします。このサンプルでは、❶のボタンをタップするたびに **Scaffold** や **AppBar** ウィジェットも含めて再構築されます。実際に表示更新を行うのは **Text** ウィジェットのみです。

そこで、このアプリの状態である **_counter** を持つウィジェットを末端に移動させてみましょう。

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

```
void main() {
  runApp(const MaterialApp(
    home: HomeScreen(),
  ));
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Screen'),
      ),
      body: const Center(
        child: CountButton(),
      ),
    );
  }
}

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

  @override
  State createState() => _CountButtonState();
}

class _CountButtonState extends State<CountButton> {
  int _counter = 0;

  void _increment() {
    setState(() {
      _counter += 1;
    });
  }

  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      child: Text('count = $_counter'),
      onPressed: () => _increment(),
    );
  }
}
```

タップするとカウントアップするボタンを `CountButton` ウィジェットとして切り出しました。アプリの状態である `_counter` は `CountButton` ウィジェットが持つようになりました。ボタンをタップして再構築されるのは `CountButton` ウィジェットのみになり不必要な再構築が行われなくなりました。

また、関心事を分けることにもつながり、`HomeScreen` 画面のコードからカウントアップのロジックを分離することができました。保守性が高まり、`CountButton` ウィジェットは再利用性も確保されています。

Riverpodの状態監視は末端のウィジェットで行う

Riverpodの状態監視を末端のウィジェットで行うことで、ウィジェットの再構築範囲を小さくすることができます。`StatefulWidget`の状態を末端に移動することと同じ考え方です。FlutterのテンプレートプロジェクトをRiverpodで書き換えた例を見てみましょう。

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

part 'main.g.dart';

@riverpod
class Counter extends _$Counter {
  @override
  int build() => 0;

  void increment() => state++;
}

void main() {
  runApp(
    const ProviderScope(
      child: MaterialApp(
        home: HomeScreen(),
      ),
    ),
  );
}

class HomeScreen extends ConsumerWidget {
  const HomeScreen({super.key});
  @override
```

```

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

```

カウンタをRiverpodのクラスベースのProviderで実装しました❶。カウンタの状態は❷の箇所です。監視、FloatingActionButtonウィジェットのonPressedコールバックでは❸のようにカウンタをインクリメントしています。

この例では、カウンタをインクリメントすると、HomeScreen画面のbuildメソッドが呼ばれウィジェットが再構築されます。実際に更新が必要なのはカウンタの状態を表示するTextウィジェットのみですので、別のウィジェットクラスに切り出してしまうでしょう。

```

import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:riverpod_annotation/riverpod_annotation.dart';

part 'main.g.dart';

@riverpod

```

```

class Counter extends _$Counter {
  @override
  int build() => 0;

  void increment() => state++;
}

void main() {
  runApp(
    const ProviderScope(
      child: MaterialApp(
        home: HomeScreen(),
      ),
    ),
  );
}

class HomeScreen extends ConsumerWidget {
  const HomeScreen({super.key});
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home Screen'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: const [
            Text(
              'You have pushed the button this many times:',
            ),
            CounterText(), —❶
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          ref.read(counterProvider.notifier).increment();
        },
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}

```