# State management also outside flutter

# To the moment

- InheritedWidget
- ValueNotifier
- ChangeNotifier
- Provider

- Any issues?

# To the moment

- InheritedWidget
- ValueNotifier
- ChangeNotifier
- Provider

- Any issues?
  - SOME DEPEND ON FLUTTER
  - **HOW ABOUT DART?**

pub.dev Sign in Help

# state_notifier 0.7.2+1

Published 2 months ago• dash-overflow.net Null safety

SDK DART FLUTTER

PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

197

Readme Changelog Example Installing Versions Scores

pub v0.7.2+1 Welcome to **state_notifier**~

This package is a recommended solution for managing state when using Provider or Riverpod.

Long story short, instead of extending ChangeNotifier, extend StateNotifier:

```
class City {
  City({required this.name, required this.population});
  final String name;
  final int population;
}


class CityNotifier extends StateNotifier<List<City>> {
  CityNotifier() : super(const <City>[]);
```

https://pub.dev/packages/state_notifier

197 LIKES | 130 PUB POINTS | 96% POPULARITY

Publisher

dash-overflow.net

Metadata

ValueNotifier, but outside Flutter and with some extra perks

Repository (GitHub)

View/report issues

Documentation

API reference

License

**StateNotifier has 2 main widgets:**

**StateNotifierProvider:** StateNotifierProvider is the equivalent of `ChangeNotifierProvider` but for StateNotifier. Its main task is to create StateNotifier and it will automatically dispose it when the widget is removed from the tree!

StateNotifierBuilder: StateNotifierBuilder is equivalent to `ValueListenableBuilder` from Flutter. You can rebuild the widget based on the value changed in the StateNotifier!

However, there are other widgets too available with StateNotifier. Now, let's see the code implementation of the same. We will implement the Counter Application using StateNotifier!

https://medium.com/google-developer-experts/manage-the-state-of-your-app-using-statenotifier-flutter-cf0ed89984b2

**StateNotifier has 2 main widgets:**

**StateNotifierProvider:** StateNotifierProvider is the equivalent of `ChangeNotifierProvider` but for StateNotifier. Its main task is to create StateNotifier and it will automatically dispose it when the widget is removed from the tree!

StateNotifierBuilder: StateNotifierBuilder is equivalent to `ValueListenableBuilder` from Flutter. You can rebuild the widget based on the value changed in the StateNotifier!

However, there are other widgets too available with StateNotifier. Now, let's see the code implementation of the same. We will implement the Counter Application using StateNotifier!

https://medium.com/google-developer-experts/manage-the-state-of-your-app-using-statenotifier-flutter-cf0ed89984b2

**StateNotifier has 2 main widgets:**

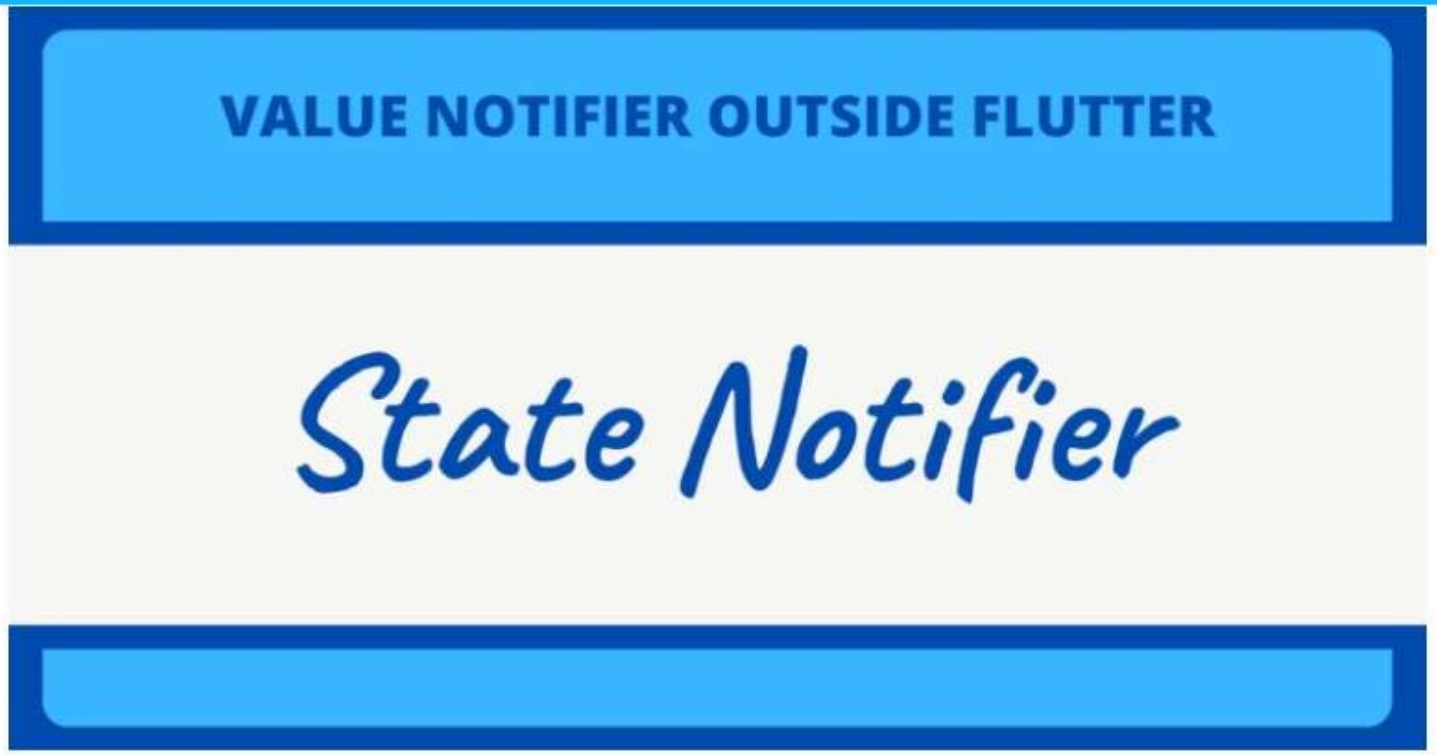
**StateNotifierProvider:** StateNotifierProvider is the equivalent of `ChangeNotifierProvider` but for StateNotifier. Its main task is to create StateNotifier and it will automatically dispose it when the widget is removed from the tree!

StateNotifierBuilder: StateNotifierBuilder is equivalent to `ValueListenableBuilder` from Flutter. You can rebuild the widget based on the value changed in the StateNotifier!

However, there are other widgets too available with StateNotifier. Now, let's see the code implementation of the same. We will implement the Counter Application using StateNotifier!

https://medium.com/google-developer-experts/manage-the-state-of-your-app-using-statenotifier-flutter-cf0ed89984b2

HOME CLEANER FLUTTER CONTRIBUTIONS TALKS ABOUT




# StateNotifier, ValueNotifier outside Flutter

Another way to manage state

Marcos Sevilla · Jan 25, 2021 · 7 min read

https://marcossevilla.dev/state-notifier

HOME CLEANER FLUTTER CONTRIBUTIONS TALKS ABOUT

VALUE NOTIFIER OUTSIDE FLUTTER

State Notifier

# Hello StateNotifier

`StateNotifier` is another type of class that serves as a logic component and is a "reimplementation" of `ValueNotifier`, with the difference that it doesn't depend on Flutter.

**Flutter**

Another way to manage state

Marcos Sevilla · Jan 25, 2021 · 7 min read

https://marcossevilla.dev/state-notifier

HOME CLEANER FLUTTER CONTRIBUTIONS TALKS ABOUT

VALUE NOTIFIER OUTSIDE FLUTTER

State Notifier

# Hello StateNotifier

`StateNotifier` is another type of class that serves as a logic component and is a "reimplementation" of `ValueNotifier`, with the difference that it doesn't depend on Flutter.

**Flutter**

Another way to manage state

Marcos Sevilla · Jan 25, 2021 · 7 min read

https://marcossevilla.dev/state-notifier

HOME CLEANER FLUTTER CONTRIBUTIONS TALKS ABOUT

VALUE NOTIFIER OUTSIDE FLUTTER

State Notifier

# Hello StateNotifier

`StateNotifier` is another type of class that serves as a logic component and is a "reimplementation" of `ValueNotifier`, with the difference that it doesn't depend on Flutter.

**Flutter**

Another way to manage state

Marcos Sevilla · Jan 25, 2021 · 7 min read

https://marcossevilla.dev/state-notifier

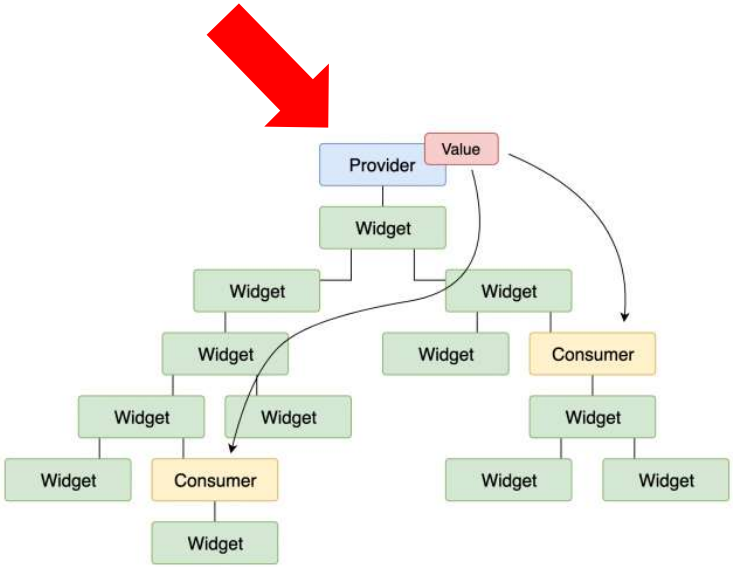
```
abstract class UserState {}

class UserNotLogged extends UserState {}

class UserLogged extends UserState {
  UserLogged({required this.user});
  final User user;
}
```

```
class UserStateNotifier extends StateNotifier<UserState> {
  // we create our initial state by passing it to super, just li
  UserStateNotifier() : super(UserNotLogged());

  // we change state assigning a new one to the state property.
  void logIn(User user) => state = UserLogged(user: user);
}
```

```
abstract class UserState {}

class UserNotLogged extends UserState {}

class UserLogged extends UserState {
  UserLogged({required this.user});
  final User user;
}
```

```
class UserStateNotifier extends StateNotifier<UserState> {
  // we create our initial state by passing it to super, just li
  UserStateNotifier() : super(UserNotLogged());

  // we change state assigning a new one to the state property.
  void logIn(User user) => state = UserLogged(user: user);
}
```

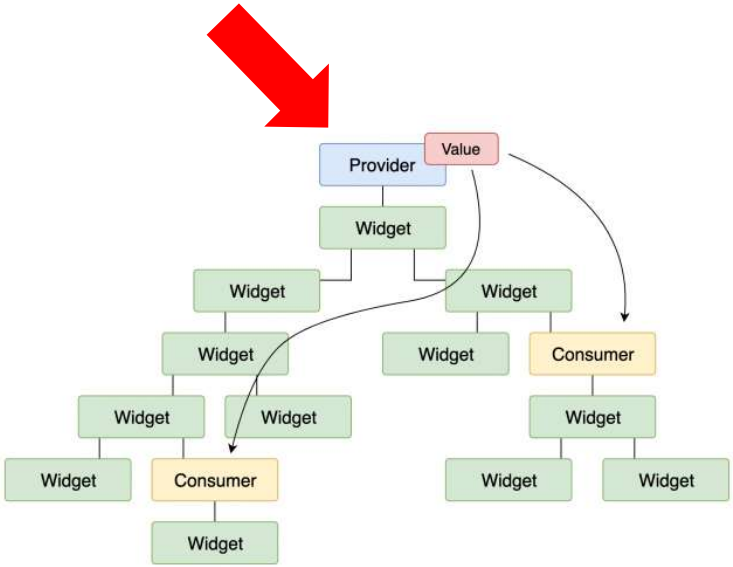
```
void main() {
  runApp(
    StateNotifierProvider<UserStateNotifier, UserState>(
      create: (_) => UserStateNotifier(),
      child: MyApp(),
    ),
  );
}
```



```
void main() {
  runApp(
    StateNotifierProvider<UserStateNotifier, UserState>(
      create: (_) => UserStateNotifier(),
      child: MyApp(),
    ),
  );
}
```

```
class UserStateNotifier extends StateNotifier<UserState> {
  // we create our initial state by passing it to super, just li
  UserStateNotifier() : super(UserNotLogged());

  // we change state assigning a new on
  void logIn(User user) => state = User
}
```

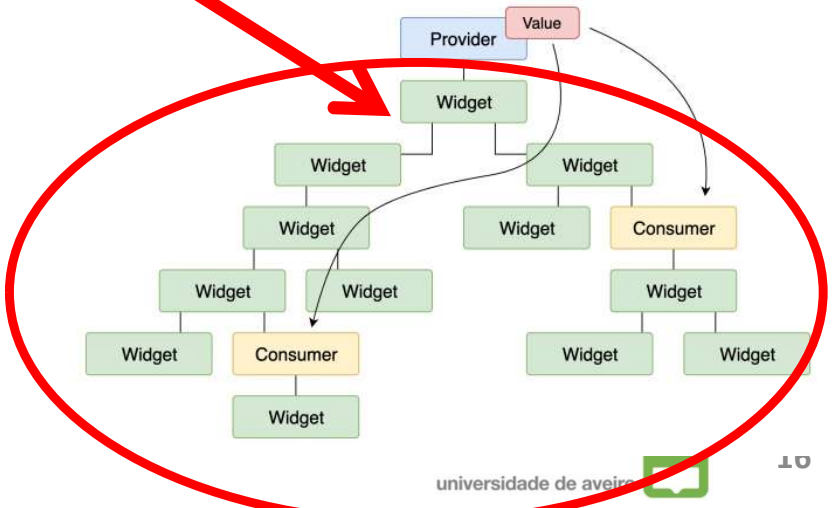
```
abstract class UserState {}

class UserNotLogged extends UserState {}

class UserLogged extends UserState {
  UserLogged({required this.user});
  final User user;
}
```

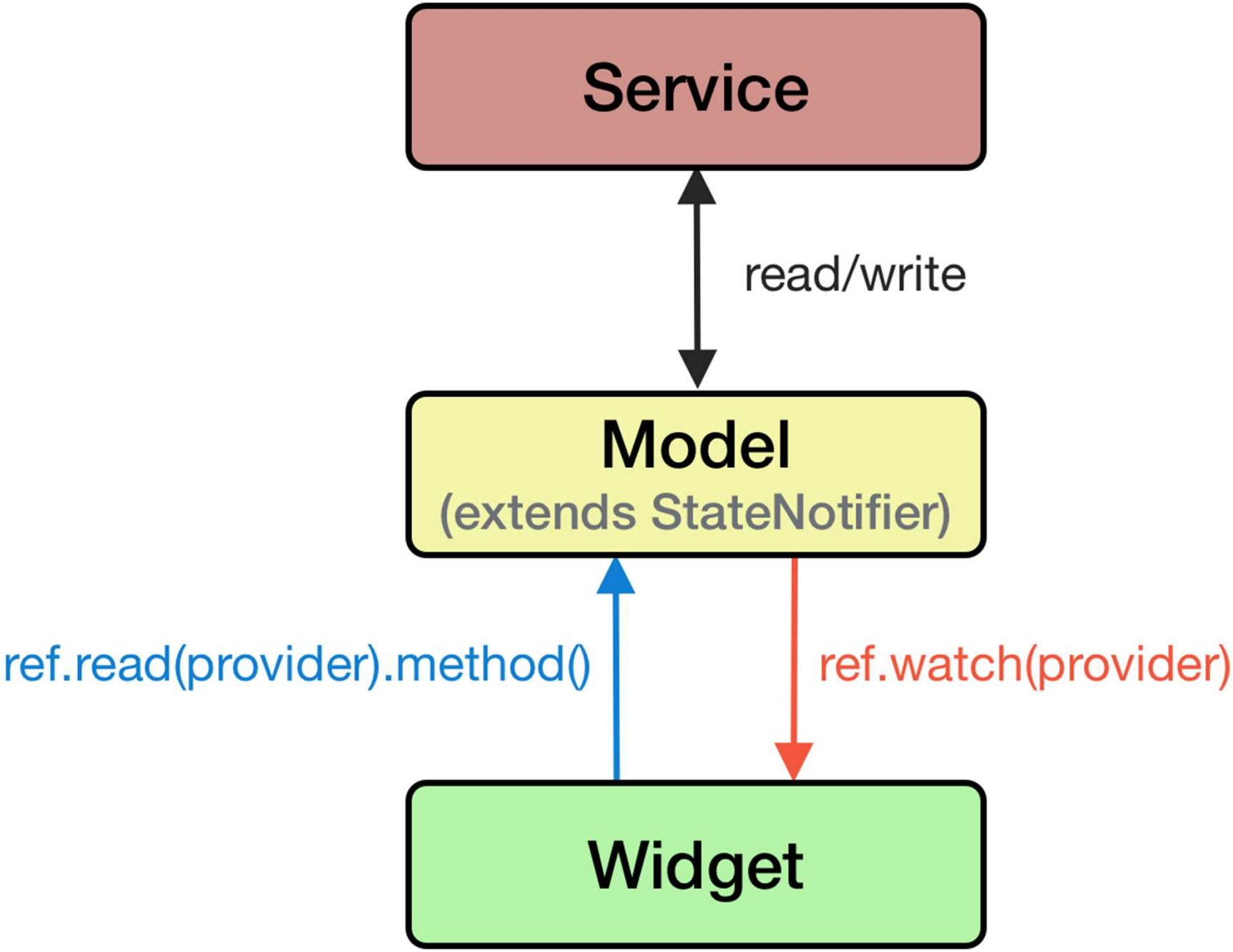


```
void main() {
  runApp(
    StateNotifierProvider<UserStateNotifier, UserState>(
      create: (_) => UserStateNotifier(),
      child: MyApp(),
    ),
  );
}
```

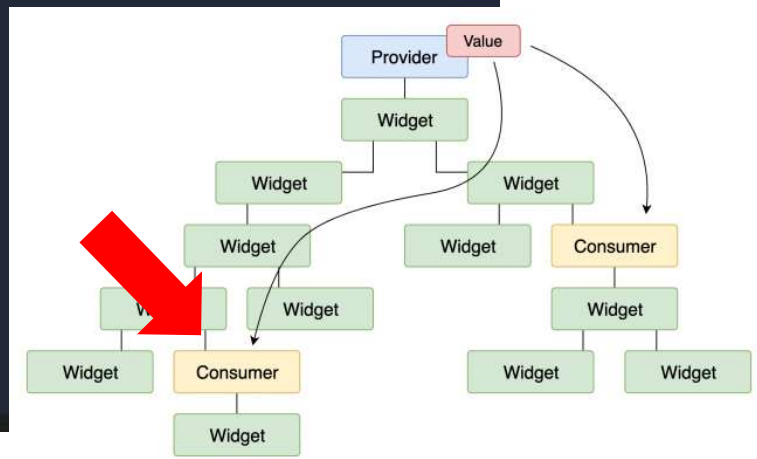
```
class UserStateNotifier extends StateNotifier<UserState> {
  // we create our initial state by passing it to super, just li
  UserStateNotifier() : super(UserNotLogged());

  // we change state assigning a new on
  void logIn(User user) => state = User
}
```

```
abstract class UserState {}

class UserNotLogged extends UserState {}

class UserLogged extends UserState {
  UserLogged({required this.user});
  final User user;
}
```




Service
read/write
Model
(extends StateNotifier)
ref.read(provider).method()
ref.watch(provider)
Widget

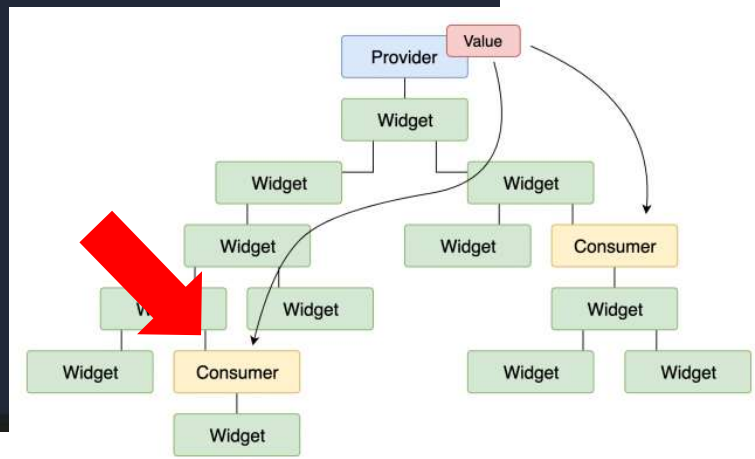
```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userState = context.watch<UserStateNotifier>();
    return Scaffold(
      body: Center(
        child: Text('${userState.runtimeType}'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          context.read<UserStateNotifier>().logIn(User());
        },
      ),
    );
  }
}
```



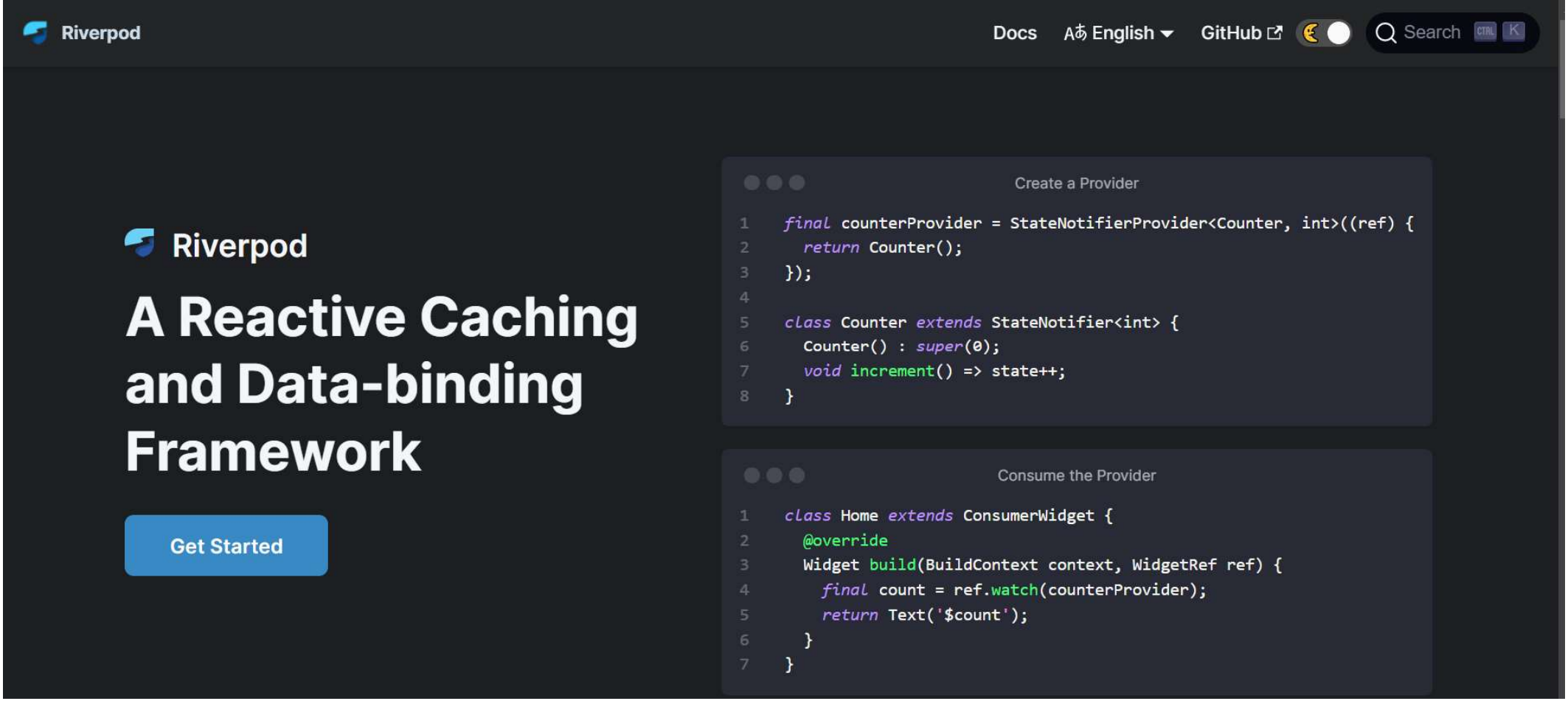
```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userState = context.watch<UserStateNotifier>();
    return Scaffold(
      body: Center(
        child: Text('${userState.runtimeType}'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          context.read<UserStateNotifier>().logIn(User());
        },
      ),
    );
  }
}
```

Look for notifier
~dependency injection
Get reference

Look for notifier
~dependency injection
READ



riverpod




https://riverpod.dev/

A Reactive Caching and Data-binding Framework

Define the “provider”
~ Singleton

# Declare shared state from anywhere

No need to jump between your `main.dart` and your UI files anymore.
Place the code of your shared state where it belongs, be it in a separate package or right next to the Widget that needs it, without losing testability.

```
// A shared state that can be accessed by multiple
// objects at the same time
final countProvider = StateProvider((ref) => 0);

// Comsumes the shared state and rebuild when it changes
class Title extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final count = ref.watch(countProvider);
    return Text('$count');
  }
}
```

Riverpod provides a StateNotifier implementation among other...

# Safely read providers

Reading a provider will never result in a bad state. If you can write the code needed to read a provider, you will obtain a valid value.

This even applies to asynchronously loaded values. As opposed to with provider, Riverpod allows cleanly handling loading/error cases.

```
final configurationsProvider = FutureProvider<Configuration>((ref) async {
  final uri = Uri.parse('configs.json');
  final rawJson = await File.fromUri(uri).readAsString();

  return Configuration.fromJson(json.
});

class Example extends ConsumerWidget {
  @override
  Widget build(BuildContext conte  WidgetRef ref) {
    final configs = ref.watch(configurationsProvider);

    // Use Riverpod's built-in support
    // for error/loading states using "when":
    return configs.when(
      loading: () => const CircularProgressIndicator(),
      error: (err, stack) => Text('Error $err'),
      data: (configs) => Text('data: ${configs.host}'),
    );
  }
}
```

Get reference to provider

Riverpod provides a StateNotifier implementation among other...

Riverpod

Docs | English | GitHub | Search

- Getting started
- Concepts
- Guides
  - Testing
- Migration
  - ^0.13.0 to ^0.14.0
  - ^0.14.0 to ^1.0.0
- Official examples
- Third party examples
- Api references
  - All Providers
  - riverpod
  - flutter_riverpod
  - hooks_riverpod

# Getting started

Before diving into the inner mechanisms of Riverpod, let's start with the basics: Installing Riverpod, then writing a "Hello world".



## What package to install

Before anything, you need to be aware that Riverpod is spread across multiple packages, with slightly differen
The variant of Riverpod that you will want to install depends on the app you are making.

Riverpod provides several implementations

You can refer to the following table to help you decide which package to use:

| app type | package name | description |
|---|---|---|
| Flutter only | flutter_riverpod | A basic way of using Riverpod with flutter. |
| Flutter + flutter_hooks | hooks_riverpod | A way to use both flutter_hooks and Riverpod together. |
| Dart only (No Flutter) | riverpod | A version of Riverpod with all the classes related to Flutter stripped out. |

## Installing the package

Once you know what package you want to install, proceed to add the following to your `pubspec.yaml`:

Flutter + flutter_hooks | Flutter | Dart only

pubspec.yaml

```
environment:
  sdk: ">=2.12.0 <3.0.0"
  flutter: ">=2.0.0"

dependencies:
  flutter_hooks: 0.18.0
  hooks_riverpod: ^2.0.0-dev.4
```

https://riverpod.dev/docs/getting_started

Riverpod

Docs | English | GitHub | Search

- Getting started
- Concepts
- Guides
  - Testing
- Migration
  - ^0.13.0 to ^0.14.0
  - ^0.14.0 to ^1.0.0
- Official examples
- Third party examp
- Api references
  - All Providers
  - riverpod
  - flutter_riverpod
  - hooks_riverpod

# Getting started

Before diving into the inner mechanisms of Riverpod, let's start with the basics: Installing Riverpod, then writing a "Hello world".



## What package to install

| app type | package name | description |
|---|---|---|
| Flutter only | flutter_riverpod | A basic way of using Riverpod with flutter. |
| Flutter + flutter_hooks | hooks_riverpod | A way to use both flutter_hooks and Riverpod together. |
| Dart only (No Flutter) | riverpod | A version of Riverpod with all the classes related to Flutter stripped out. |

Riverpod provides several implementations

```
  sdk: ">=2.12.0 <3.0.0"
  flutter: ">=2.0.0"

dependencies:
  hooks_riverpod: ^2.0.0-dev.4
```

https://riverpod.dev/docs/getting_started

Riverpod

Docs A English GitHub

Dart only

lib/main.dart

```dart
import 'package:riverpod/riverpod.dart';

// We create a "provider", which will store a value (here "Hello
// By using a provider, this allows us to mock/override the value
final helloWorldProvider = Provider((_) => 'Hello world');

void main() {
  // This object is where the state of our providers will be stor
  final container = ProviderContainer();

  // Thanks to "container", we can read our provider.
  final value = container.read(helloWorldProvider);

  print(value); // Hello world
}
```

Which you can then execute with `dart lib/main.dart`.
This will print "Hello world" in the console.

Riverpod provides Dart implementation

Riverpod

Docs A あ English GitHub

Dart only

lib/main.dart

```dart
import 'package:riverpod/riverpod.dart';

// We create a "provider", which wil tore a value (here "Hello
// By using a provider, this allows us mock/override the value
final helloWorldProvider = Provider((_) => 'Hello world');

void main() {
  // This object where the state of our providers will be stor
  final container = ProviderContainer();

  // Thanks to "container", can read our provider.
  final value = container.read(helloWorldProvider);

  print(value); // Hello world
}
```

Declare provider

Manager all notifiers
~riverpod singleton

Container manages
the notifiers

Which you can then execute with `dart lib/main.dart`.
This will print "Hello world" in the console.

Riverpod provides
Dart implementation

lib/main.dart

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

// We create a "provider", which will store a value (here "Hello
// By using a provider, this allows us to mock/override the value
final helloWorldProvider = Provider((_) => 'Hello world');

void main() {
  runApp(
    // For widgets to be able to read providers, we need to wrap
    // application in a "ProviderScope" widget.
    // This is where the state of our providers will be stored.
    ProviderScope(
      child: MyApp(),
    ),
  );
}
```

Declare provider

Top of the tree

Riverpod provides Flutter implementation

Riverpod

Docs A English GitHub

Flutter

lib/main.dart

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

// We create a "provider", which wi
// By using a provider, this allows
final helloWorldProvider = Provider

void main() {
  runApp(
    // For widgets to be able to re
    // application in a "ProviderSc
    // This is where the state of o
    ProviderScope(
      child: MyApp(),
    ),
  );
}
```

```
// Extend ConsumerWidget instead of StatelessWidget, whic
class MyApp extends ConsumerWidget {
  @override
  Widget build(BuildCon t context, WidgetRef ref) {
    final String value = ref.watch(helloWorldProvider);

    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Example')),
        body: Center(
          child: Text(value),
        ),
      ),
    );
  }
}
```

Watch the pr

- Note not the Provider refe

Riverpod provides Flutter implementation

Which you can then execute with `flutter run`.

This will render "Hello world" on your device.

## Flutter + flutter_hooks

lib/main.dart

Copy

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

// We create a "provider", which will store a value (here "Hello
// By using a provider, this allows us to mock/override the value
final helloWorldProvider = Provider((_) => 'Hello world');

void main() {
  runApp(
    // For widgets to be able to read providers, we need to wrap
    // application in a "ProviderScope" widget.
    // This is where the state of our providers will be stored.
    ProviderScope(
      child: MyApp(),
    ),
  );
}
```

Declare provider

Top of the tree

Riverpod provides
Flutter+hooks implementation

Riverpod Docs Aあ English

## Flutter + flutter_hooks

lib/main.dart

Copy

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

// We create a "provider", which will store a value (here "Hello
// By using a provider, th
final helloWorldProvider =

void main() {
  runApp(
    // For widgets to be a
    // application in a "P
    // This is where the s
    ProviderScope(
      child: MyApp(),
    ),
  );
}
```

```
// Note: MyApp is a HookConsumerWidget, from hooks_riverpod.
class MyApp extends HookConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final String value = ref.watch(helloWorldProvider);

    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Example')),
        body: Center(
          child: Text(value),
        ),
      ),
    );
```

Riverpod provides
Flutter+hooks implementation

# Flutter State Management with Riverpod: The Essential Guide

#dart #flutter #state-management #app-architecture #riverpod

UPDATED NOV 18, 2021 | 28 MIN READ

Riverpod is a popular Flutter state management library that shares many of the advantages of Provider and brings many additional benefits.

According to the official documentation:

https://codewithandrea.com/articles/flutter-state-management-riverpod/

```
final valueProvider = Provider<int>((ref) {
  return 36;
});
```

Define provider

```
void main() {
  runApp(ProviderScope(
    child: MyApp(),
  ));
}
```

Provider scope on top of the tree

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          'Some text here 👍',
          style: Theme.of(context).textTheme.headline4,
        ),
      ),
    );
  }
}
```

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Cente
        child: Te
          'Some t
          style:
        ),
      ),
    );
  }
}
```

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        // 1. Add a Consumer
        child: Consumer(
          // 2. specify the builder and obtain a WidgetRef
          builder: (_, WidgetRef ref, __) {
            // 3. use ref.watch() to get the value of the provider
            final value = ref.watch(valueProvider);
            return Text(
              'Value: $value',
              style: Theme.of(context).textTheme.headline4,
            );
          },
        ),
      ),
    );
  }
}
```

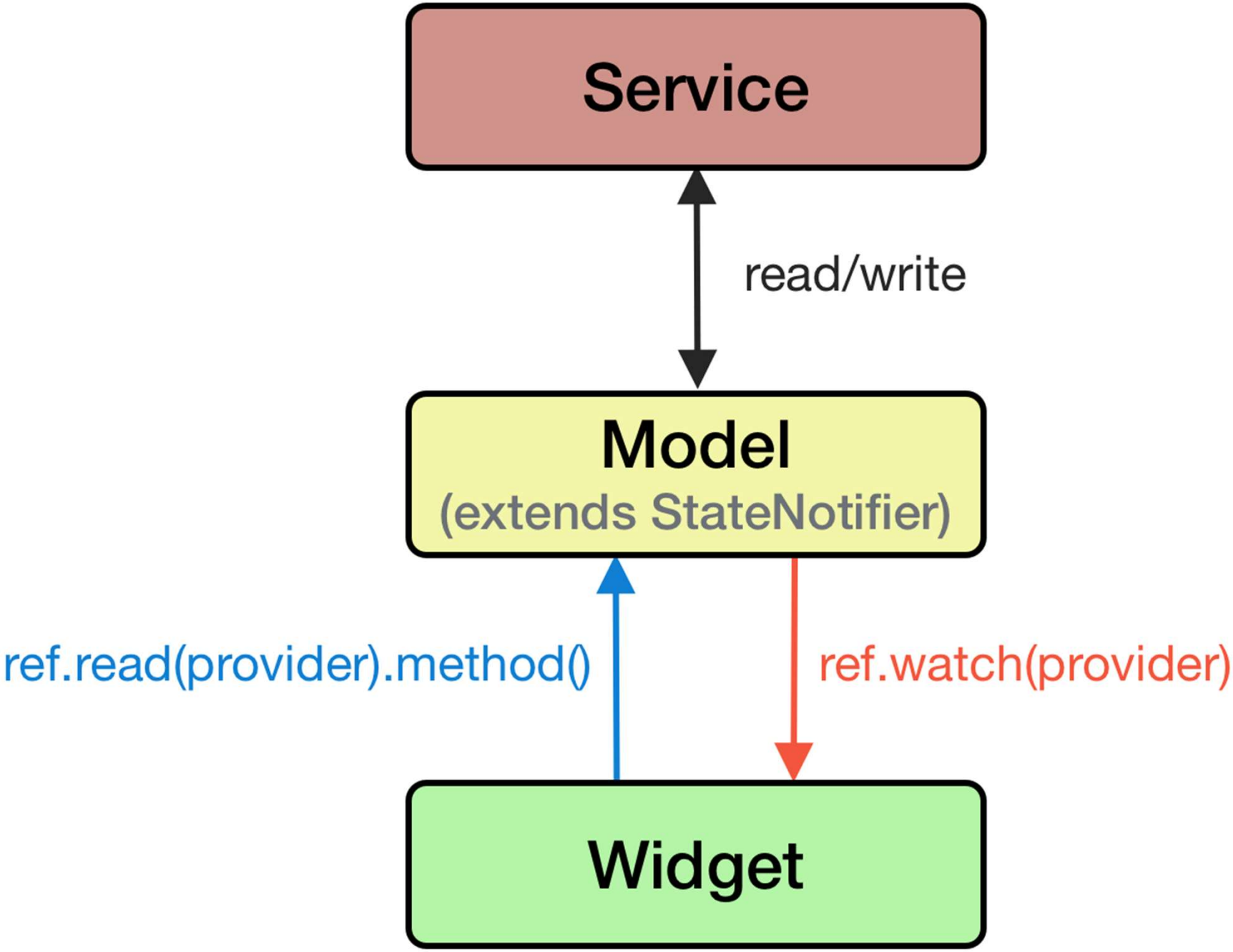
The consumer

Watch provider

```
final valueProvider = Provider<int>((ref) {
  return 36;
});
```

Define provider

```
// 1. Widget class now extends [ConsumerWidget]
class MyHomePage extends ConsumerWidget {
  @override
  // 2. build() method has an extra [WidgetRef] argument
  Widget build(BuildContext context, WidgetRef ref) {
    // 3. use ref.watch() to get the value of the provider
    final value = ref.watch(valueProvider);
    return Scaffold(
      body: Center(
        child: Text(
          'Value: $value',
          style: Theme.of(context).textTheme.headline4,
        ),
      ),
    );
  }
}
```


Service
read/write
Model
(extends StateNotifier)
ref.read(provider).method()
ref.watch(provider)
Widget

```
final counterStateProvider = StateProvider<int>((ref) {
  return 0;
});
```

```
class MyHomePage extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    // 1. watch the counterStateProvider
    final counter = ref.watch(counterStateProvider);
    return Scaffold(
      body: Center(
        child: Text(
          // 2. use the counter value
          'Value: $counter',
          style: Theme.of(context).textTheme.headline4,
        ),
      ),
    );
  }
}
```

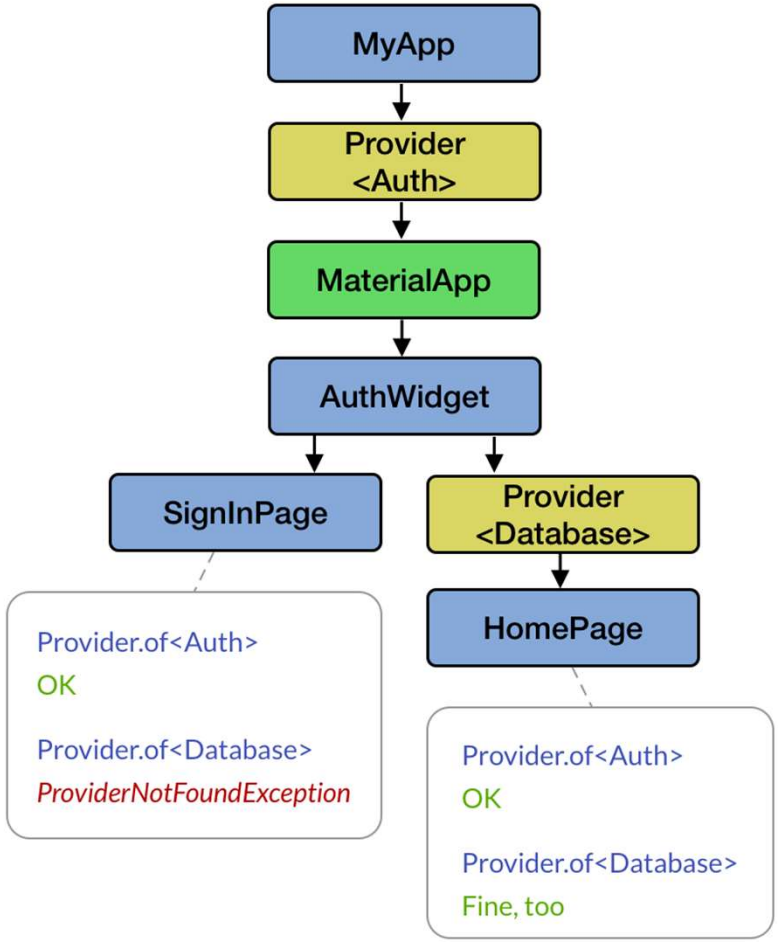
```
final futureProvider = FutureProvider<int>((ref) {
  return Future.value(36);
});

final streamProvider = StreamProvider<int>((ref) {
  return Stream.fromIterable([36, 72]);
});
```

# And more…

```
final clockProvider = StateNotifierProvider<Clock, DateTime>((ref) {
  return Clock();
});
```

# Why use Riverpod for Flutter state management?

- **Provider Drawback #1: Combining Providers is very verbose**
- **Provider Drawback #2: Getting Providers by type and runtime exceptions**




# Riverpod: Rewriting Provider

Understanding Riverpod and not die trying

Marcos Sevilla · Feb 6, 2022 · 11 min read

TABLE OF CONTENTS

Cause

https://marcossevilla.dev/riverpod

# RIVERPOD

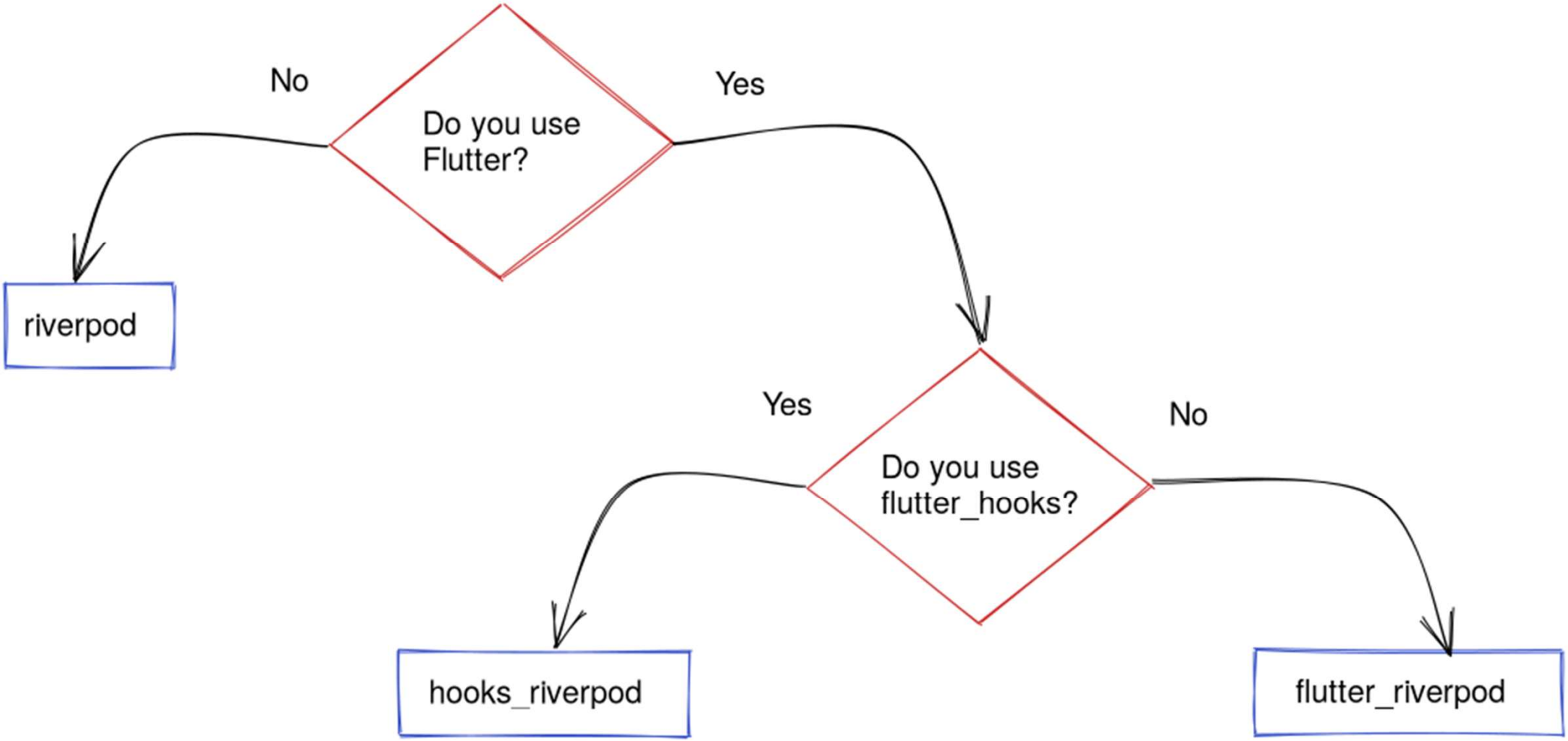



TABLE OF CONTENTS

Cause

https://marcossevilla.dev/riverpod

# Flutter_hooks

Not focus

Sign up

rrousselGit / flutter_hooks Public

React hooks for Flutter. Hooks are a new kind of object that manages a Widget life-cycles. They are used to increase code sharing between widgets and as a complete replacement for StatefulWidget.

MIT License

2.2k stars 124 forks

Star Notifications

Code Issues 19 Pull requests 1 Discussions ...

https://github.com/rrousselGit/flutter_hooks
https://pub.dev/packages/flutter_hooks

# Hooks and HookWidget

- Hook
  - **Delegate state / function**
  - code-sharing between widgets by removing duplicates.
- HookWidget
  - Hooks still works with widget
- Predefined & custom

```
class Example extends HookWidget {
  const Example({Key key, required this.duration})
      : super(key: key);

  final Duration duration;

  @override
  Widget build(BuildContext context) {
    final controller = useAnimationController(dura
    return Container();
  }
}
```

# Rationale: needs a common "resource"

If this idea is still unclear, a naive implementation of hooks could look as follows:

```
class HookElement extends Element {
  List<HookState> _hooks;
  int _hookIndex;

  T use<T>(Hook<T> hook) => _hooks[_hookIndex++].build(this);

  @override
  performRebuild() {
    _hookIndex = 0;
    super.performRebuild();
  }
}
```

# StatefullWidget

```
class _ExampleState extends State<Example> with SingleTickerProviderStateMixin {
  AnimationController? _controller;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(vsync: this, duration: widget.duration);
  }

  @override
  void didUpdateWidget(Example oldWidget) {
    super.didUpdateWidget(oldWidget);
    if (widget.duration != oldWidget.duration) {
      _controller!.duration = widget.duration;
    }
  }

  @override
  void dispose() {
    _controller!.dispose();
    super.dispose();
  }

  @override
```

# StatefullWidget vs Hook

```
class _ExampleState extends State<Example> with SingleTickerProviderStateMixin {
  AnimationController? _controller;

  @override
  void initState() {
    super.init
    _controlle
  }

  @override
  void didUpda
    super.didU
    if (widget
      _control
    }
  }

  @override
  void dispose
    _controlle
    super.disp
  }

  @override
```

```
class Example extends HookWidget {
  const Example({Key key, required this.duration})
      : super(key: key);

  final Duration duration;

  @override
  Widget build(BuildContext context) {
    final controller = useAnimationController(duration: duration
    return Container();
  }
}
```

Chidume Nnamdi Follow

I'm a software engineer with over six years of experience. I've worked with different stacks, including WAMP, MERN, and MEAN. My language of choice is JavaScript; frameworks are Angular and Node.js.

# How to use Flutter Hooks

August 23, 2021 · 6 min read

Hooks, meet Flutter. Inspired by React Hooks and Dan Abramov's piece, *Making Flutter.*

https://blog.logrocket.com/how-to-use-flutter-hooks/

# Counter example)

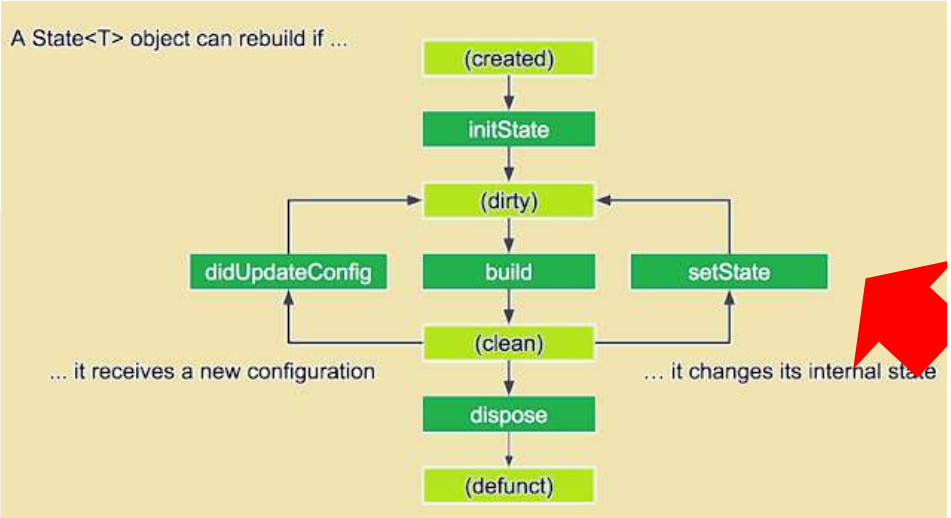
```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

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

StatefulWidget

Hook

```
class MyHomePage extends HookWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    final _counter = useState(0);
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
```

# Counter example




```
@override
Widget build(BuildContext context) {
  final _counter = useState(0);
  return Scaffold(
    appBar: AppBar(
      title: Text(title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Wi
          Text(
            'You have
          ),
          Text(
```

```
floatingActionButton: FloatingActionButton(
  onPressed: () => _counter.value++,
  tooltip: 'Increment',
  child: Icon(Icons.add),
), // This trailing comma makes auto-formatting nicer for build
```

NO SET STATE…
Automatically triggers refresh on change

# Predefined and unspecific hooks

| Name |
| --- |
| useEffect |
| useState |
| useMemoized |
| useRef |
| useCallback |
| useContext |
| useValueChang |

dart:async rel

| Nam |
| --- |
| useStream |
| useStreamC |
| useFuture |

Listenable related

| Name |
| --- |
| useListenable |
| useValueNotifier |
| useValueListenab |

Animation rel

| Name |
| --- |
| useSingleTickerProvider |
| useAnimationController |
| useAnimation |

| Name |
| --- |
| useReducer |
| usePrevious |
| useTextEditingController |
| useFocusNode |
| useTabController |
| useScrollController |
| usePageController |
| useAppLifecycleState |
| useOnAppLifecycleStateChange |
| useTransformationController |
| useIsMounted |

# The END