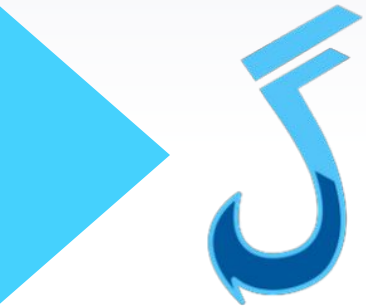

Flutter Hooks

Michael Reyes

Desarrollador en Lean Mind

- BLoC Pattern con Flutter
  https://leanmind.es/es/blog/bloc-pattern
- Flutter, ¡una maravilla! ¿o... no?
  https://leanmind.es/es/blog/flutter_una_maravilla_o_no
- Tips de CSS que podrían ayudarte
  https://leanmind.es/es/blog/tips-css-que-podrian-ayudarte

mreysei.dev

@mreysei

Introducción

React

Si desarrollas en React con Hooks tendrás una mayor curva de aprendizaje

Pub dev

El paquete aún no ha llegado a la versión 1, pero es buen puente para adaptarte a la tecnología

- pub.dev/packages/flutter_hooks
- pub.dev/documentation/flutter_hooks/latest/index.html
- github.com/rrousselGit/flutter_hooks

useState

EN FLUTTER

Lo llamamos dentro de un componente de función para agregarle un estado local.

Flutter se suscribirá a la variable y reconstruirá el widget.

useState devuelve un objeto: el objeto tendrá el valor de estado actual permitiendo modificarlo para reconstruir el widget.

EN REACT

Lo llamamos dentro de un componente de función para agregarle un estado local.

React mantendrá este estado entre re-renderizados.

useState devuelve un par: el valor de estado actual y una función que le permite actualizarlo.

useState

```jsx
import React, { useState } from 'react';

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

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

class UseStateExample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    // Declaración de una variable de estado que llamaremos "count"
    final counter = useState(0);

    return Scaffold(
      appBar: AppBar(
        title: const Text('useState example'),
      ),
      body: Center(
        child: Text('Button tapped ${counter.value} times'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => counter.value++,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

useState

```jsx
import React, { useState } from 'react';

function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

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

class UseStateExample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    // Declaración de una variable de estado que llamaremos "count"
    final counter = useState(0);

    return Scaffold(
      appBar: AppBar(
        title: const Text('useState example'),
      ),
      body: Center(
        child: Text('Button tapped ${counter.value} times'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => counter.value++,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

useEffect

EN FLUTTER

El **useEffect** se llamará sincrónicamente en cada build. En caso de que se haya especificado **keys**, se llamará cuando alguna de esas **keys** cambie su valor.

Los efectos se declaran dentro del componente para que tengan acceso a sus props y estado.

EN REACT

Cuando llamas a **useEffect**, le estás diciendo a React que ejecute tu función de "efecto" después de vaciar los cambios en el DOM.

Los efectos se declaran dentro del componente para que tengan acceso a sus props y estado.

De forma predeterminada, React ejecuta los efectos después de cada renderizado — incluyendo el primer renderizado.

useEffect

```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Recupera los datos en el localstorage al construir el componente
    setCount(localStorage.getItem('myCounter'))
  }, []);

  useEffect(() => {
    // Actualiza los datos en el localstorage cada vez que se actualice el count
    localStorage.setItem('myCounter', count)
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

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


class UseEffectExample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    final counter = useState(0);

    useEffect(() => {
      // Recupera los datos en el localstorage al construir el componente
      SharedPreferences prefs = await SharedPreferences.getInstance();
      counter.value = prefs.getInt('counter') ?? 0
    }, []);

    useEffect(() => {
      // Actualiza los datos en el localstorage cada vez que se actualice el count
      SharedPreferences prefs = await SharedPreferences.getInstance();
      await prefs.setInt('counter', counter);
    }, [counter.value]);

    return Scaffold(
      appBar: AppBar(
        title: const Text('useState example'),
      ),
      body: Center(
        child: Text('Button tapped ${counter.value} times'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => counter.value++,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

useEffect

```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Recupera los datos en el localstorage al construir el componente
    setCount(localStorage.getItem('myCounter'))
  }, []);

  useEffect(() => {
    // Actualiza los datos en el localstorage cada vez que se actualice el count
    localStorage.setItem('myCounter', count)
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

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


class UseEffectExample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    final counter = useState(0);

    useEffect(() => {
      // Recupera los datos en el localstorage al construir el componente
      SharedPreferences prefs = await SharedPreferences.getInstance();
      counter.value = prefs.getInt('counter') ?? 0
    }, []);

    useEffect(() => {
      // Actualiza los datos en el localstorage cada vez que se actualice el count
      SharedPreferences prefs = await SharedPreferences.getInstance();
      await prefs.setInt('counter', counter);
    }, [counter.value]);

    return Scaffold(
      appBar: AppBar(
        title: const Text('useState example'),
      ),
      body: Center(
        child: Text('Button tapped ${counter.value} times'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => counter.value++,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Más Hooks!

- `useAnimation`
- `useAnimationController`
- `useContext`
- `useEffect`
- `useFocusNode`
- `useFuture`
- `useIsMounted`
- `useListenable`
- `useMemoized`
- `usePageController`
- `usePrevious`
- `useReassemble`
- `useReducer`
- `useScrollController`
- `useSingleTickerProvider`
- `useState`
- `useStream`
- `useStreamController`
- `useTabController`
- `useValueChanged`
- `useValueListenable`
- `useValueNotifier`

- pub.dev/documentation/flutter_hooks/latest/flutter_hooks/flutter_hooks-library.html

Custom Hooks

TANTO EN REACT COMO EN FLUTTER

Puedes construir tus propios Hooks que te permite extraer la lógica del componente en funciones reutilizables.

Custom Hooks

```
class CustomHookExample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    final StreamController<int> countController = _useLocalStorageInt('counter');

    return Scaffold(
      appBar: AppBar(
        title: const Text('Custom Hook example'),
      ),
      body: Center(
        child: HookBuilder(
          builder: (context) {
            final AsyncSnapshot<int> count =
                useStream(countController.stream, initialData: 0);

            return !count.hasData
                ? const CircularProgressIndicator()
                : GestureDetector(
                    onTap: () => countController.add(count.data + 1),
                    child: Text('You tapped me ${count.data} times.'),
                  );
          },
        ),
      ),
    );
  }
}
```

```
StreamController<int> _useLocalStorageInt(String key, { int defaultValue = 0 }) {
  final controller = useStreamController<int>(keys: [key]);

  useEffect(
    () {
      final sub = controller.stream.listen((data) async {
        final prefs = await SharedPreferences.getInstance();
        await prefs.setInt(key, data);
      });
      return sub.cancel;
    },
    [controller, key],
  );

  useEffect(
    () {
      SharedPreferences.getInstance().then<void>((prefs) async {
        final int valueFromStorage = prefs.getInt(key);
        controller.add(valueFromStorage ?? defaultValue);
      }).catchError(controller.addError);
      return null;
    },
    [controller, key],
  );

  return controller;
}
```

¿Te gusta React Hooks?
Te gusta Flutter Hooks