Stateful y stateless widgets

El diseño de interfaces con Flutter se basa en widgets y los widgets se dividen en dos tipos, stateful y stateless. Vamos a ver cuáles son sus diferencias.

Decimos que **un widget es un stateless widget si es estático**, es decir, si no va a sufrir ningún cambio. Por ejemplo un texto, un icono… en principio son widgets que van a ser pintados en la pantalla y así se van a quedar, sin que ninguna acción del usuario los altere.

Por otro lado tenemos a los**stateful widgets**, o widgets dinámicos. Éstos **sí que pueden sufrir cambios** debidos seguramente a una acción del usuario. Un claro ejemplo puede ser un checkbox, si el usuario pulsa sobre el su aspecto cambiará. Otro ejemplo puede ser un TextField, porque el usuario puede escribir sobre él y por tanto cambiarlo. Este tipo de widgets **tienen un objeto «State» asociado para gestionar su estado** (de ahí el nombre). Si el estado cambia el widget volverá a pintarse en la pantalla con la apariencia actualizada. Esto es muy importante porque cuando creemos nuestro propio stateful widget tendremos también que crear otra clase state.

Como siempre, la mejor forma de ver esto es con un ejemplo. Vamos a crear una pantalla de aplicación super sencilla creando un widget estático y después veremos cómo darle interactividad convirtiéndolo en dinámico.

Creando un stateless widget

En este ejemplo vamos a hacer una pantalla de app muy simple que tenga un botón de play, como si fuera un reproductor de música. Para ello vamos a crear una clase con el contenido del reproductor. Esta clase heredará de StatelessWidget y por tanto será un widget estático.

Cuando creamos una clase de StateWidget, **estamos obligados a implementar el método build**. Este método es el encargado de devolver lo que se va a pintar, es decir, un widget.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'My app title',

home: Scaffold(

body: Center(

child: PlayerWidget()

)

)

);

}

}

class PlayerWidget extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Container(

color: Colors.yellow,

width: double.infinity,

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

IconButton(

icon: Icon(Icons.play\_arrow),

iconSize: 150.0,

onPressed: () {}),

Text('Play',style: new TextStyle(fontSize: 24.0),)

],

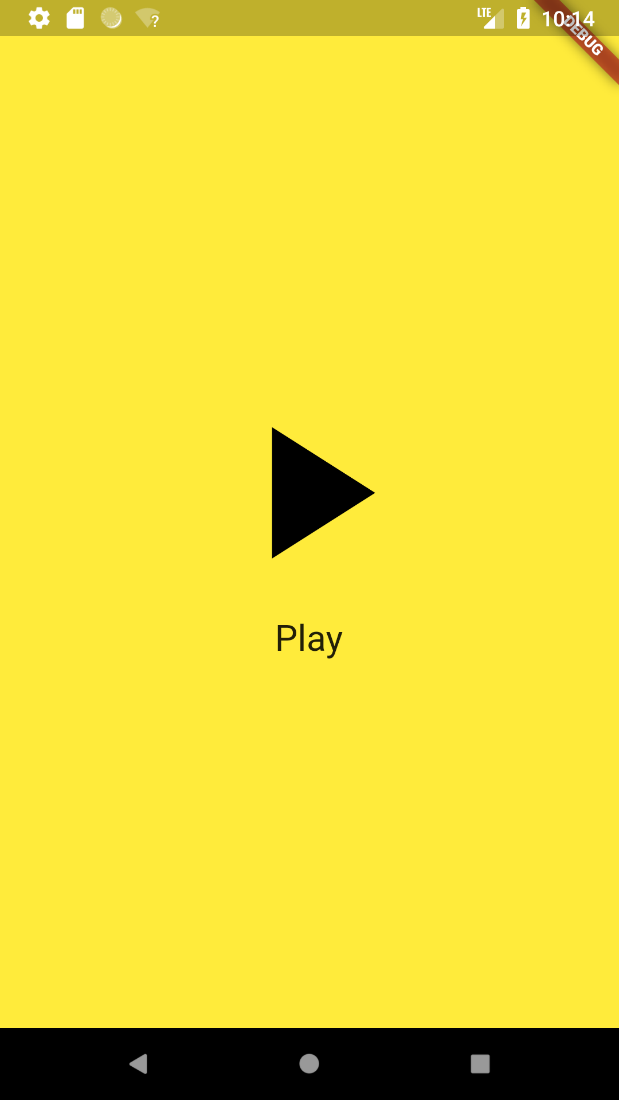
)

);

}

}

Si ejecutamos este código veremos el siguiente resultado:



Como vemos, con un stateful widget hemos podido construir el elemento que necesitábamos para nuestra interfaz, pero en este caso necesitamos que el usuario pueda interactuar con este elemento, queremos que al pulsar sobre el botón (imaginando que un audio comienza a reproducirse), el botón de «play» debe ser sustituido por uno de «pause» y el texto también debería cambiar. Esto no lo podemos hacer con un stateful widget, ya que son estáticos y no pueden cambiar, así que lo que tenemos que hacer es convertir nuestro widget en un stateful widget.

Convirtiéndolo en stateful widget

Para convertir nuestra clase PlayerWidget en un stateful widget tenemos que hacer lo siguiente:

* PlayerWidget heredará de StatefulWidget en lugar de de StatelessWidget
* Creamos otra clase \_PlayerWidgetState que herede de State. Si recuerdas, antes comenté que los stateful widgets tendrían que tener una clase asociada para gestionar su estado
* Tenemos que implementar el método createState en PlayerWidget devolviendo una instancia de la clase que gestiona el estado
* Todo lo que antes teníamos en el buid del widget dinámico ahora pasará a la clase \_PlayerWidgetState

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'My app title',

home: Scaffold(

body: Center(

child: PlayerWidget()

)

)

);

}

}

class PlayerWidget extends StatefulWidget {

@override

State<PlayerWidget> createState() {

return new \_PlayerWidgetState();

}

}

class \_PlayerWidgetState extends State<PlayerWidget> {

@override

Widget build(BuildContext context) {

return Container(

color: Colors.yellow,

width: double.infinity,

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

IconButton(

icon: Icon(Icons.play\_arrow),

iconSize: 150.0,

onPressed: () {}),

Text('Play',style: new TextStyle(fontSize: 24.0),)

],

)

);

}

}

Si ejecutamos el código vemos que muestra exactamente el mismo contenido que antes. La diferencia es que ahora tenemos un widget dinámico, por lo que podemos empezar a trabajar en los cambios de estado.

Cambiar el estado del widget

Lo primero que haremos es crear un atributo booleano para la clase del estado que será el que se encarga de decirnos si el reproductor está en play o en pausa.

class \_PlayerWidgetState extends State<PlayerWidget> {

bool \_isPlaying = false;

...

}

Después crearemos un método dentro de la misma clase al que llamar cuando hagamos click en el botón del play. Básicamente lo que hace este método es darle valor al atributo que acabamos de crear, en concreto el valor contrario al que tenía (si está en pausa y pulsamos es que ahora está reproduciendo y al revés). Muy importe es que esta acción de dar valor se hace dentro de una función llamada setState. **Cuando usamos setState se ejecuta el código que contiene y después se vuelve a pintar el contenido del widget asociado al estado**. La idea es que tras cambiar el valor de \_isPlaying se vuelva a pintar el widget.

class \_PlayerWidgetState extends State<PlayerWidget> {

...

void \_playerClick(){

setState((){

\_isPlaying = !\_isPlaying;

});

}

...

}

Lo siguiente que tenemos que hacer es que el icono y el texto de nuestro reproductor de mentira dependan del valor del atributo \_isPlaying. Es tan sencillo como poner un par de condiciones. También invocamos a \_isPlaying al pulsar sobre el icono. Así quedaría el código entero.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'My app title',

home: Scaffold(

body: Center(

child: PlayerWidget()

)

)

);

}

}

class PlayerWidget extends StatefulWidget {

@override

State<PlayerWidget> createState() {

return new \_PlayerWidgetState();

}

}

class \_PlayerWidgetState extends State<PlayerWidget> {

bool \_isPlaying = false;

void \_playerClick(){

setState((){

\_isPlaying = !\_isPlaying;

});

}

@override

Widget build(BuildContext context) {

return Container(

color: Colors.yellow,

width: double.infinity,

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

IconButton(

icon: \_isPlaying ? Icon(Icons.pause) : Icon(Icons.play\_arrow),

iconSize: 150.0,

onPressed: () {\_playerClick();}),

Text(\_isPlaying ?'Pause':'Play',style: new TextStyle(fontSize: 24.0),)

],

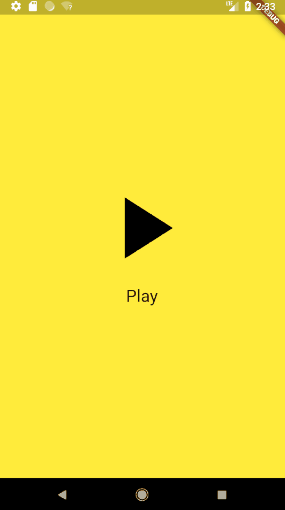
)

);

}

}

También h-Como tras pulsar el Si lo ejecutamos y pulsamos sobre el icono, veremos como cambia el contenido de nuestro widget dinámico.



Pues ya lo tenemos, un pequeño ejemplo de cómo usar los dos tipos de widget, stateful cuando no van a cambiar o stateful cuando necesitamos que sea dinámico.

Si quieres tener el código para lo que sea, [lo tienes en gihub](https://github.com/guillegarcia/flutter_samples).