Widget组件（一）

2021年6月29日 14:26

- 在Flutter中一切的显示都是Widget，Widget是一切的基础，利用响应式模式进行渲染。
- 分为有状态和无状态两种：flutter每个页面都是一帧，无状态就是保持在那一帧，有状态的Widget当数据更新时，其实是创建了新的Widget，通过State实现了跨帧的数据同步。

一、 StatelessWidget

通过build方法返回一个布局好的控件。Widget和Widget之间通过child进行嵌套。

启动组件只能是无状态Widget，否则会报如下错误

```
The following _CastError was thrown during a scheduler callback:
type 'MyApp' is not a subtype of type 'StatelessWidget' in type cast
```

根Widget种类可以有：

1. WidgetsApp：如果需要自定义风格，可以使用
2. MaterialApp：Material Design风格
3. CupertinoApp：IOS风格的根Widget

二、StatefulWidget

创建一个StatefulWidget类需要两个类，一个继承StatefulWidget，一个继承State<>类。

StatefulWidget类本身是不变的，但是State类在Widget中的生命周期是始终存在的。

三、State的生命周期

flutter程序可以看成一个巨大的状态机器，用户的操作、网络请求、系统事件都是推动这个状态机运行的触发点，触发点通过setState来推动状态机的改变。

构造函数 --> initState --> didChangeDepencies --> build --> deactivate --> dispose

构造函数：调用一次

initState：调用一次，相当于onCreate

didChangeDependencies：多次调用

build：多次调用，在setState时会触发调用，所以需要修改界面的地方需要用setState包装一下，不然不会刷新

deactivate：多次调用，界面跳转调用

dispose：调用一次，相当于onDestroy

例子

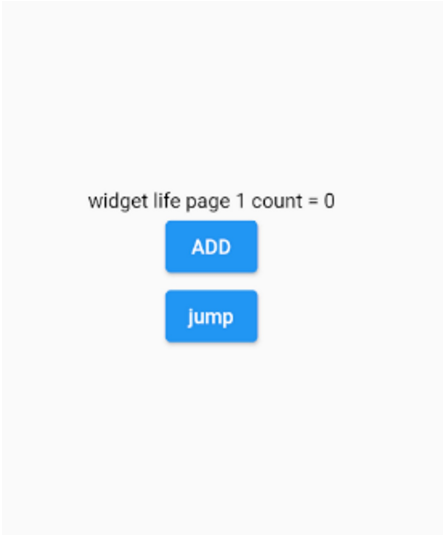
```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primaryColor: Colors.white,
        primarySwatch: Colors.blue
      ),
      home: WidgetLifePage(),
    );
  }
}
class WidgetLifePage extends StatefulWidget {
  const WidgetLifePage({Key? key}) : super(key: key);

  @override
  _WidgetLifePageState createState() => _WidgetLifePageState();
}

class _WidgetLifePageState extends State<WidgetLifePage> {
  int _page1Count = 0;
  Color _color = Colors.black;
  _increase() {
    setState(() {
      _page1Count++;
    });
    _changeColor();
  }
  _changeColor() {
    setState(() {
      _color == Colors.black ? _color = Colors.blue : _color = Colors.black;
    });
  }
  @override
  Widget build(BuildContext context) {
    // return Container();
    print('page1 build');
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text("widget life page 1 count = $_page1Count", style:
TextStyle(color: _color),),
            ElevatedButton(onPressed: () => _increase(), child: Text("ADD")),
            ElevatedButton(onPressed: () {
              Navigator.push(context, MaterialPageRoute(builder: (context) =>
WidgetLifePage2()));
            }, child: Text("jump"))
          ],
        )
      ),
    );
  }

  @override
  void didChangeDependencies() {
    print('page1 didChangeDependencies');
    super.didChangeDependencies();
```

```
  }

  @override
  void dispose() {
    print('page1 dispose');
    super.dispose();
  }

  @override
  void deactivate() {
    print('page1 deactivate');
    super.deactivate();
  }

  @override
  void didUpdateWidget(WidgetLifePage oldWidget) {
    print('page1 didUpdateWidget');
    super.didUpdateWidget(oldWidget);
  }

  @override
  void initState() {
    print('page1 initSate');
    super.initState();
  }
}

class WidgetLifePage2 extends StatefulWidget {
  const WidgetLifePage2({Key? key}) : super(key: key);

  @override
  _WidgetLifePage2State createState() => _WidgetLifePage2State();
}

class _WidgetLifePage2State extends State<WidgetLifePage2> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("hhh"),),
    );
  }

  @override
  void didChangeDependencies() {
    print('page2 didChangeDependencies');
    super.didChangeDependencies();
  }

  @override
  void dispose() {
    print('page2 dispose');
    super.dispose();
  }

  @override
  void deactivate() {
    print('page2 deactivate');
    super.deactivate();
  }
```

```
  @override
  void didUpdateWidget(WidgetLifePage2 oldWidget) {
    print('page2 didUpdateWidget');
    super.didUpdateWidget(oldWidget);
  }

  @override
  void initState() {
    print('page2 initSate');
    super.initState();
  }
}
```

该代码显示的显示的界面如下图所示



此时，控制台打印的内容为

```
page1 initSate
page1 didChangeDependencies
page1 build
```

点击ADD按钮后，page1 build会被打印

点击JUMP按钮后，打印内容如下

```
page2 initSate
page2 didChangeDependencies
```

当从page2返回page1后，

再次注意setState会导致Widget重新绘制，也就是build方法被触发。

四、MaterialApp

官方将Material组件分为了几个类型：

1. 应用程序结构和导航
2. Button
3. 输入和选择
4. 对话框、警告弹窗和面板
5. 信息显示
6. 布局

MaterialApp可以类比为网页中的<html></html>，且自带路由、主题色，<title>等功能。
title：String类型，会在Android应用管理器APP上方显示，对于ios设备无效
home：Widget类型，首先显示的Widget，除非设置initialRoute
routes：Map<String, WidgetBuilder>类型，是应用的顶级路由表。当使用Navigator.pushNamed进行命名路由的跳转时，会在此路由表中进行查找并跳转。

五、Scaffold
Scaffold通常被用作MaterialApp的子Widget，它会填充可用空间，占据整个窗口或设备屏幕。
提供了appBar，bottomNavigationBar，隐藏的侧边栏drawer。
appBar：显示Scaffold的顶部区域
drawer：抽屉侧边栏
bottomNavigationBar：底部tab栏，必须大于2个
body：主体内容