Flutter 之 Key 详解

Abner_XuanYuan

1、Key 作用

在 Flutter 中，Key 是不能重复使用的，所以 Key 一般用来做唯一标识。组件在更新的时候，其状态的保存主要是通过判断组件的类型或者 key 值是否一致。因此，当各组件的类型不同的时候，类型已经足够用来区分不同的组件了，此时我们可以不必使用 key。但是如果同时存在多个同一类型的控件的时候，此时类型已经无法作为区分的条件了，我们就需要使用到 key。

2、示例

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  List<Widget> list = [
    Box(
      color: Colors.blue,
    ),
    Box(
```

```
      color: Colors.red,
    ),
    Box(
      color: Colors.orange,
    )
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          setState(() {
            list.shuffle(); //打乱list的顺序
          });
        },
        child: const Icon(Icons.refresh),
      ),
      appBar: AppBar(
        title: const Text('Title'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: list,
        ),
      ),
    );
  }
}

// ignore: must_be_immutable
class Box extends StatefulWidget {
  Color color;
```

```
  Box({super.key, required this.color});
  @override
  State<Box> createState() => _BoxState();
}

class _BoxState extends State<Box> {
  int _count = 0;
  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 100,
      width: 100,
      child: ElevatedButton(
        style: ButtonStyle(
            backgroundColor:
MaterialStateProperty.all(widget.color)),
        onPressed: () {
          setState(() {
            _count++;
          });
        },
        child: Center(
          child: Text("$_count"),
        ),
      ),
    );
  }
}
```

运行后发现改变 list Widget 顺序后，Widget 颜色会变化，但是每个 Widget 里面的文本内容并没有变化?

当 List 重新排序后 Flutter 检测到了 Widget 的顺序变化，所以重新绘制 ListWidget，但是 Flutter 发现 List Widget 里面

的元素没有变化，所以就没有改变 Widget 里面的内容。
把 List 里面的 Box 的颜色改成一样，这个时候重新对 list 进行排序，就很容易理解了。重新排序后虽然执行了 setState，但是代码和以前是一样的，所以 Flutter 不会重构 List Widget 里面的内容, 也就是 Flutter 没法通过 Box 里面传入的参数来识别 Box 是否改变。如果要让 FLutter 能识别到 List Widget 子元素的改变，就需要给每个 Box 指定一个 key。

3、LocalKey、GlobalKey

1、 Flutter key 子类包含 LocalKey 和 GlobalKey 。

局部键（LocalKey）： ValueKey、ObjectKey、UniqueKey。
全局键（GlobalKey）： GlobalKey、GlobalObjectKey。
ValueKey （值 key）： 把一个值作为 key 。
UniqueKey（唯一 key）： 程序生成唯一的 Key，当我们不知道如何指定 ValueKey 的时候就可以使用 UniqueKey。
ObjectKey（对象 key）： 把一个对象实例作为 key。
GlobalKey：每个 Widget 都对应一个 Element ，我们可以直接对 Widget 进行操作，但是无法直接操作 Widget 对应的 Element 。而 GlobalKey 就是那把直接访问 Element 的钥匙。通过 GlobalKey 可以获取到 Widget 对应的 Element 。

2、key 的使用

LocalKey 使用

```
//替换上述示例中 list 数据
  // List<Widget> list = [
  //   Box(
  //     color: Colors.blue,
  //   ),
  //   Box(
  //     color: Colors.red,
  //   ),
  //   Box(
  //     color: Colors.orange,
  //   )
  // ];
  List<Widget> list = [
    Box(
      key: const ValueKey(1),
      color: Colors.blue,
    ),
    Box(
      key: ObjectKey(Box(color: Colors.red)),
      color: Colors.red,
    ),
    Box(
      key: UniqueKey(), //程序自动生成一个key
      color: Colors.orange,
    )
  ];
```

GlobalKey 的使用

如果把 LocalKey 比作局部变量， GlobalKey 就类似于全局变量。在使用了 LocalKey 之后，在上述示例中，当屏幕状态改变的时候把 Colum 换成了 Row，Box 的状态就会丢失。

```
//示例中要使用添加过 LocalKey 之后的 list
//更改 Scaffold 中 body 里面的内容
     // body: Center(
     //   child: Column(
     //     mainAxisAlignment: MainAxisAlignment.center,
     //     children: list,
     //   ),
     // ),
     body: Center(
       child: MediaQuery.of(context).orientation ==
Orientation.portrait
           ? Column(
               mainAxisAlignment: MainAxisAlignment.center,
               children: list,
             )
           : Row(
               mainAxisAlignment: MainAxisAlignment.center,
               children: list,
             ),
     ),
```

分析：由于一个 Widget 状态的保存主要是通过判断组件的类型或者 key 值是否一致。LocalKey 只在当前的组件树有效，所以把 Colum 换成了 Row 的时候 Widget 的状态就丢失了。为了解决这个问题我们就可以使用 GlobalKey。

```dart
//在以上代码的基础上更改 list 数据
///1.不适用 key
 // List<Widget> list = [
 //   Box(
 //     color: Colors.blue,
 //   ),
 //   Box(
 //     color: Colors.red,
 //   ),
 //   Box(
 //     color: Colors.orange,
 //   )
 // ];

 ///2.使用 LocalKey
 // List<Widget> list = [
 //   Box(
 //     key: const ValueKey(1),
 //     color: Colors.blue,
 //   ),
 //   Box(
 //     key: ObjectKey(Box(color: Colors.red)),
 //     color: Colors.red,
 //   ),
 //   Box(
 //     key: UniqueKey(), //程序自动生成一个key
 //     color: Colors.orange,
 //   )
 // ];

 ///3.GlobalKey
 List<Widget> list = [];
 final GlobalKey _key1 = GlobalKey();
 final GlobalKey _key2 = GlobalKey();
```

```
final GlobalKey _key3 = GlobalKey();
@override
void initState() {
  super.initState();
  list = [
    Box(
      key: _key1,
      color: Colors.blue,
    ),
    Box(
      key: _key2,
      color: Colors.red,
    ),
    Box(
      key: _key3,
      color: Colors.orange,
    )
  ];
}
```

3、 GlobalKey 获取子组件

globalKey.currentState 可以获取子组件的状态，执行子组件的方法。

globalKey.currentWidget 可以获取子组件的属性。

_globalKey.currentContext!.findRenderObject() 可以获取渲染的属性。

```
class _HomePageState extends State<HomePage> {
 final GlobalKey _globalKey = GlobalKey();
 @override
```

```
 Widget build(BuildContext context) {
   return Scaffold(
     floatingActionButton: FloatingActionButton(
       child: const Icon(Icons.add),
       onPressed: () {
         //1、获取子组件的状态 调用子组件的属性
         var state = (_globalKey.currentState as
_BoxState);
         setState(() {
           state._count++;
         });

         //2、获取子组件的属性
         var box = (_globalKey.currentWidget as Box);
         print(box.color);

         //3、获取子组件渲染的属性
         var renderBox =
             (_globalKey.currentContext!.findRenderObject()
as RenderBox);
         print(renderBox.size);
       },
     ),
     appBar: AppBar(
       title: const Text('Title'),
     ),
     body: Center(
       child: Box(
         key: _globalKey,
         color: Colors.red,
       ),
     ),
   );
 }
}
```

4、 Widget Tree、Element Tree 、RenderObject Tree

Flutter 应用是由是 Widget Tree、Element Tree 和 RenderObject Tree 组成。

Widget：Widget 就是一个类， 是 Element 的配置信息。与 Element 的关系可以是一对多，一份配置可以创造多个 Element 实例。

Element：Widget 的实例化，内部持有 Widget 和 RenderObject。

RenderObject：负责渲染绘制。

默认情况下面，当 Flutter 同一个 Widget 的大小，顺序变化的时候，FLutter 不会改变 Widget 的 state。