

声明式UI 的编程思维

如果你之前是 Android 或 iOS 开发，你最熟悉的应该是命令式的 UI 开发，但是 Flutter 的 UI 开发是声明式的，所以需要你的编程思维从命令式向声明式转变。

从 命令式UI 到 声明式UI

假设要实现一个界面。

在 命令式UI 里，你需要手动创建所有 UI 的实例，如果界面有变化的话，就需要调用 UI实例的方法，例如 get、set 等方法。换句话说，当 UI 从一个状态转变到另一个状态，都需要对 UI 的实例操作一番，估计大家对这个过程也痛苦不已，尤其是 UI 有频繁、大规模的变动时。

当 UI 状态切换时，命令式UI 就需要写代码来实现，为了减轻开发者在不同 UI 状态之间切换写代码的负担，Flutter 采用了 声明式UI：开发者只需描述当前的 UI 状态，Framework 会自动将 UI 从上一个状态切换到下一个状态，而不需要额外的编码。

而且在 命令式UI 里，你想都不敢想的一些方法，在 声明式UI 里完全没问题，例如，Flutter 更改 UI，不是通过 set 方法，改变 Widget 的值，而是重新 build 一个全新的 Widget，而且可以做到每帧都创建新的 Widget 也没问题，因为创建新的 Widget 在 Flutter 里的消耗足够小；而每帧都创建新的 UI，这个在 命令式UI 里，是没法想象的。

声明式的 Flutter

Flutter 是声明式的，就意味着 Flutter 的 UI 反应的是当前的状态，就像下面的函数：

$$\text{UI} = f(\text{state})$$

The layout on the screen Your build methods The application state

这个函数的输入是 状态（state），函数 f 会重建 Widget，然后输出 UI，状态指的就是 APP 生命周期内变化的数据，而Flutter 的 UI 就是状态的可视化展示。

例如，假设设置菜单中有一个开关，这个开关的功能是将 Flutter APP 的主题色由红色变为蓝色，用户打开这个开关，因为值变了，所以当前 Flutter APP 的状态也发生了变化，会触发 UI 从头到尾的重建，在重建的时候，会读取当前状态的值，例如，当前的主题色是蓝色，从而让 UI 的显示发生变化。

在前面实现城市选择页面时，就是更改了变量 curCity 的值，来实现 UI 内容的刷新：

```
setState(() {  
  curCity = selectCity;  
});
```

声明式UI 在更改 UI 时，不需要对 UI 进行操作，使用的还是原来 UI 的代码，只需要更改值，使状态发生变化，就能触发 UI 刷新。

声明式UI 有很多好处：

1. 不管 UI 有多少种状态，写 UI 的代码只有一份。

2. 你想要 UI 显示成什么样子，只要描述当前的状态就行
3. 相对 UI 进行更改，也只需要描述更改后的 UI 状态就行，不需要像命令式一样进行很多操作，因为 Framework 会自动处理

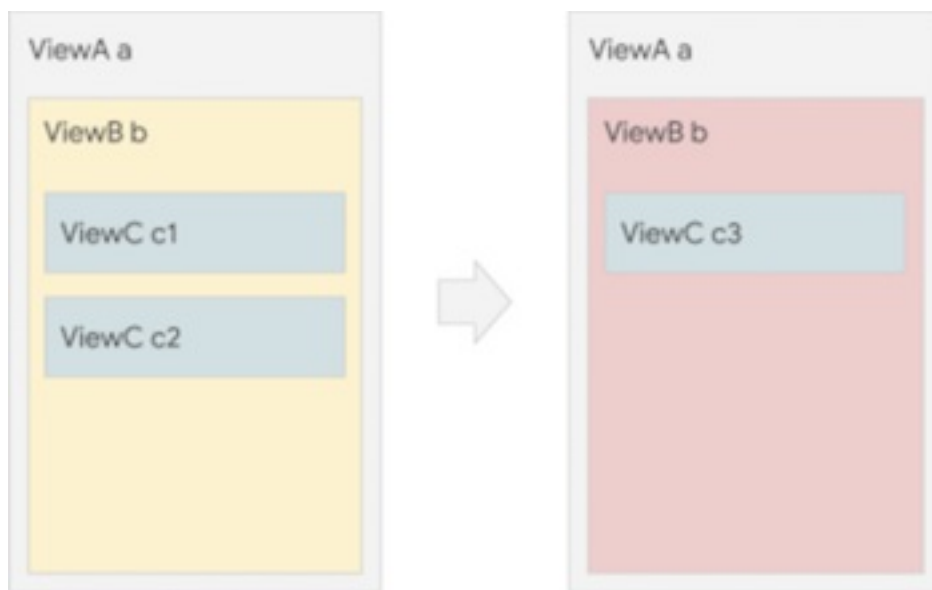
当然除了好处之外，也是会有坏处的，主要坏处就是：

1. 当 UI 的整个结构发生变化时，就不能复用原来的 UI 代码，需要重新实现。请看下面的详细对比说明。

命令式UI 和 声明式UI 实现 UI 更改的对比

声明式UI 的编程风格不像 命令式UI 的编程风格那么直观，现在通过如下的一个例子来说明。

假设要实现一个如下的功能：



先实现左边的页面，然后将 ViewB b 里的子 View 替换为 ViewC c3，且将 ViewB b 的背景由黄色变为红色。

使用命令式实现

如果用命令式的代码来实现，那么代码就会是这样子的（）：

首先实现左边的页面：

```
// 实现左边的页面
ViewA a = new ViewA(...)
ViewB b = new ViewB(...)
ViewC c1 = new ViewC(...)
ViewC c2 = New ViewC(...)
a.add(b)
b.add(c1)
b.add(c2)
```

对左边的页面进行修改，实现右边的页面：

```
//实现右边的页面
b.setColor(red)
b.clearChildren()
ViewC c3 = new ViewC(...)
b.add(c3)
```

你首先要通过 View 的构造函数创建 View 的实例，然后在组织 View 的层级修改，如果要对已有的页面进行修改，则首先要获取 View 的实例，然后对 View 的实例进行操作，更改 View 的配置，这样一步一步操作 View，就叫做 代码命令式。

使用声明式实现

如果是用 Flutter 实现，那么 Flutter 声明式的代码就是这样的：

首先实现左边的页面：

```
Widget build(BuildContext context) {  
  return ViewA{  
    child: ViewB{  
      color: yellow,  
      children: [  
        ViewC(...),  
        ViewC(...)  
      ]  
    }  
  }  
}
```

那这种用声明式写的 UI 怎么更改呢？

前面讲过 UI 如果会变化的话，就要用 `StatefulWidget`，`setState()` 会触发 `StatefulWidget` 重新构建 `Widget`。所以要改成右边的页面，Flutter 不是更改原来页面的 `Widget`，而是重新创建新的 `Widget` 实例：

```

boolean isChanged = false;

void change(){
    setState((){
        isChanged = true;
    })
}

Widget build(BuildContext context) {
    if(!isChanged){
        return ViewA{
            child: ViewB{
                color: red,
                children: [
                    ViewC(...),
                ]
            }
        }
    }else{
        return ViewA{
            child: ViewB{
                color: red,
                children: [
                    ViewC(...),
                    ViewC(...)
                ]
            }
        }
    }
}
}

```

通过变量 `isChanged` 的值来返回不同的 `Widget`，从而达到将左边的页面改成右边页面的目的。

当 UI 结构发生变化时，就需要重新写一份。