# 安装和环境配置

开发平台：windows，mac，linux

应用类型：android，ios，web，desktop

开发平台为windows，需要配置：

PUB\_HOSTED\_URL = https://pub.flutter-io.cn

FLUTTER\_STORAGE\_BASE\_URL = <https://storage.flutter-io.cn>

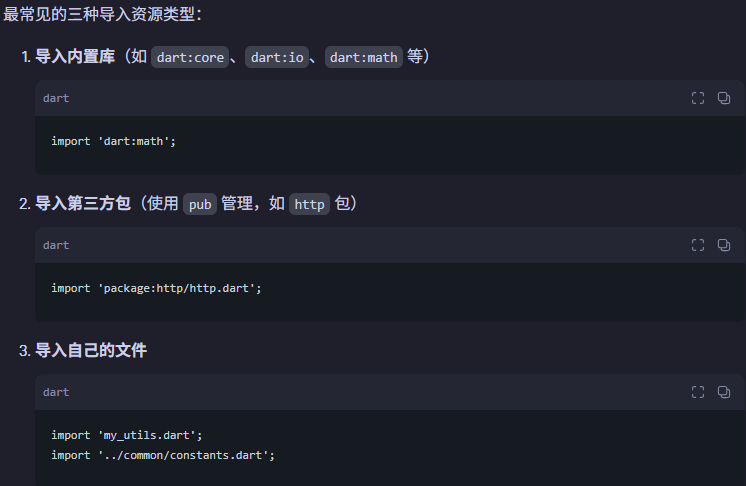
依赖：vs2022，vscode（flutter插件）

在vscode中，用flutter创建项目，如有需要则会提示安装sdk（注意add sdk to PATH）

检查开发环境：flutter doctor

# Dart

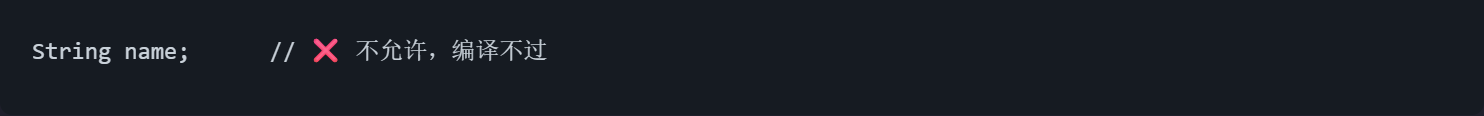
## 导入

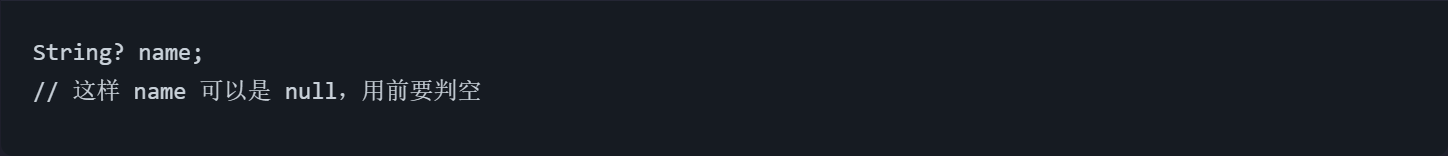


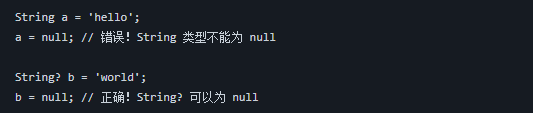
## 空安全（可为空）

所有变量都不能为null，除非你显式声明可以为null。

这样，大多数空错误可以在编译阶段被检测和避免。

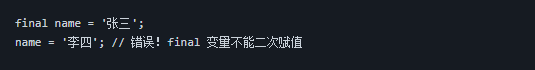






## final

只能赋值一次，但值运行时才能确定。



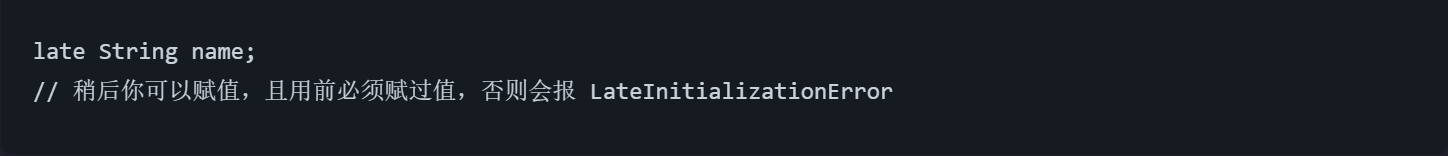
## const

只能赋值一次，值在编译时就已经确定。

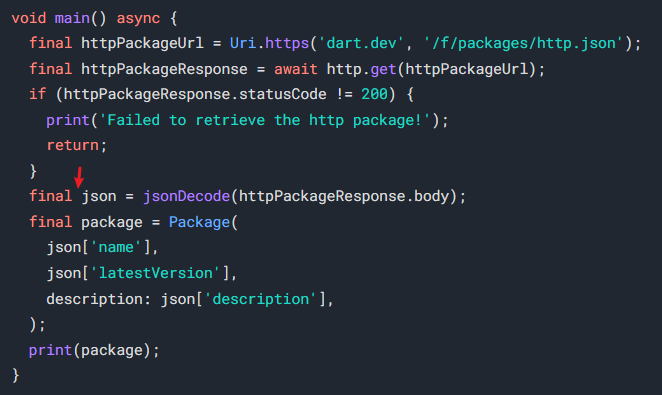


## late

稍后赋值，但不能不赋值。

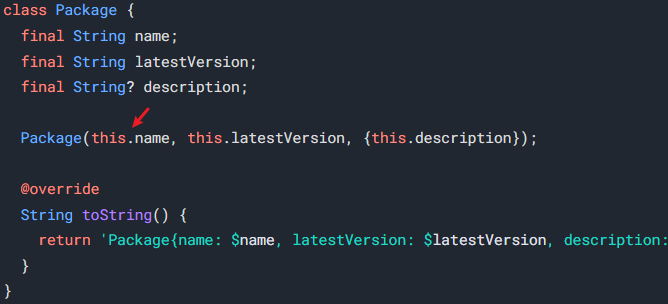


## 自动类型推断

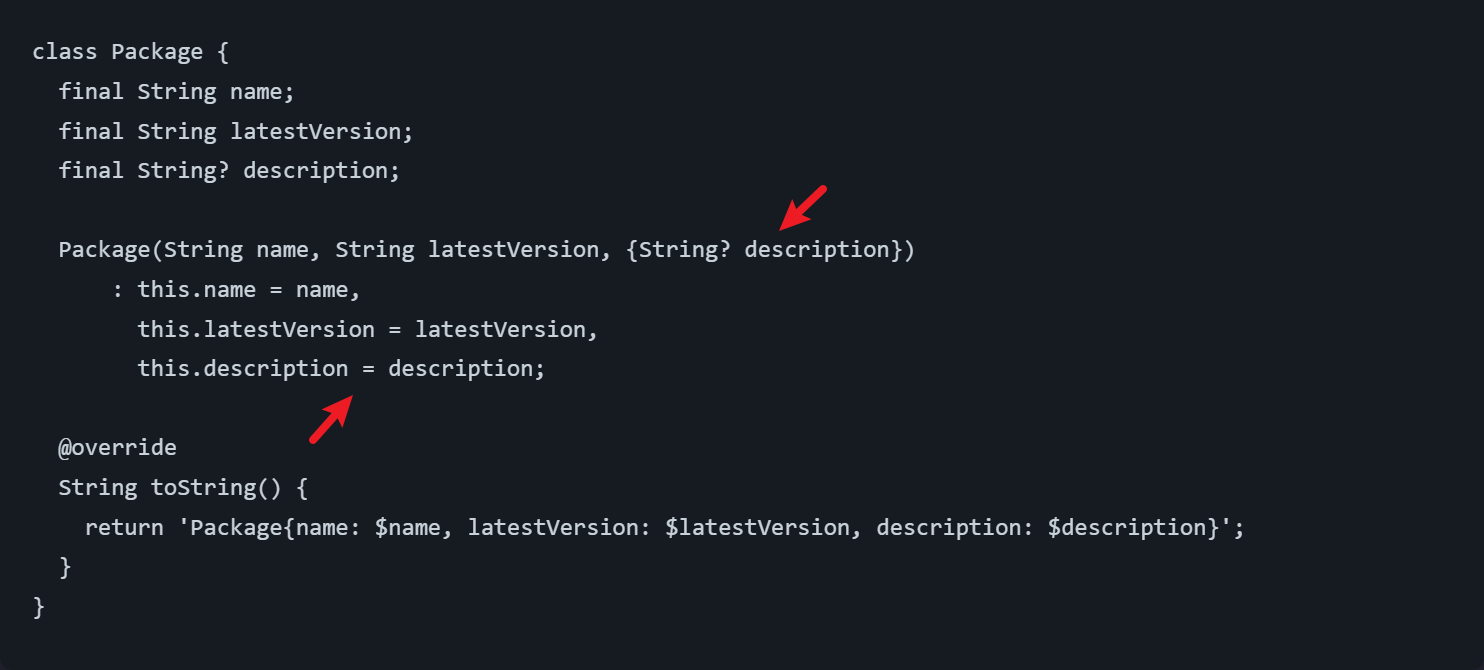


## 构造函数参数简写

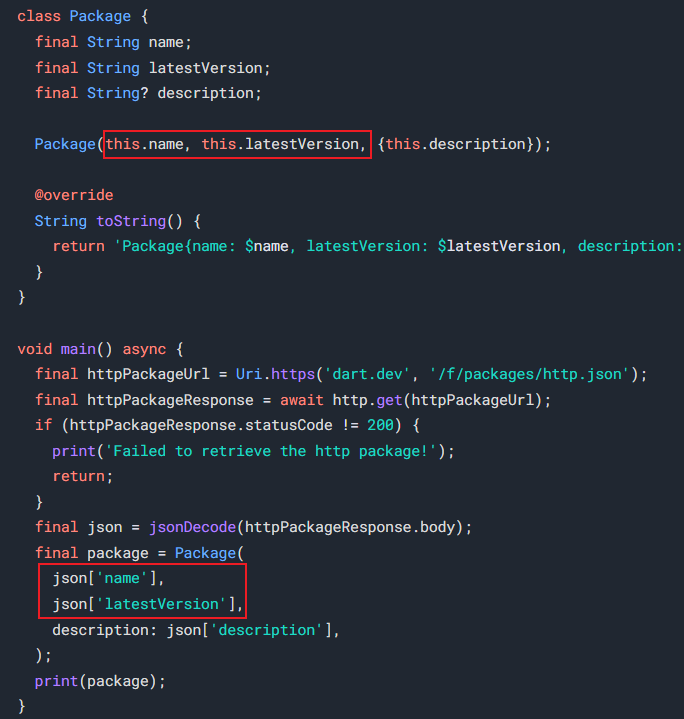
简写代码



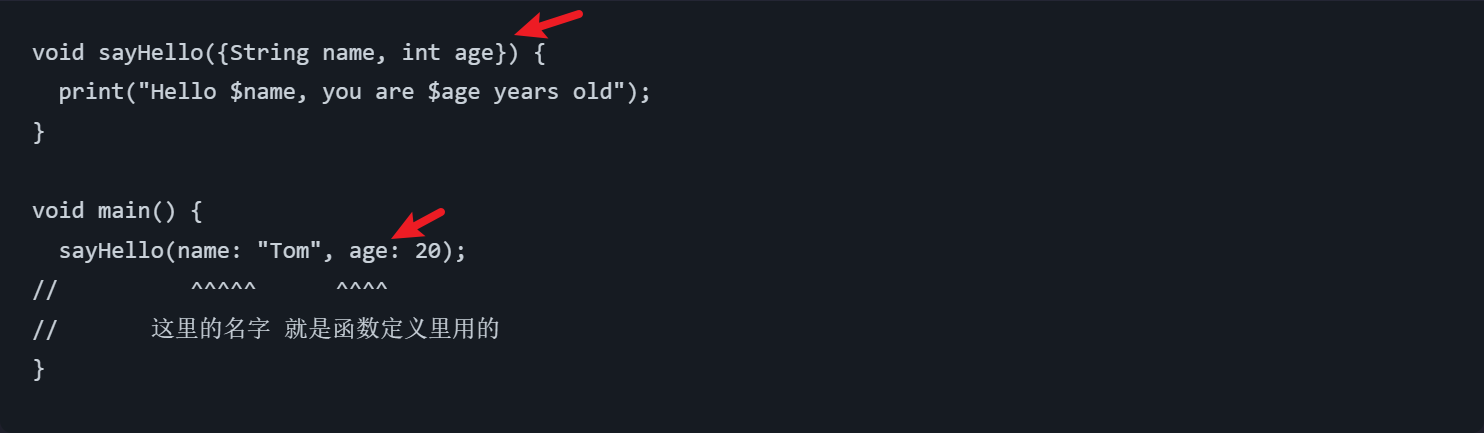
等价代码

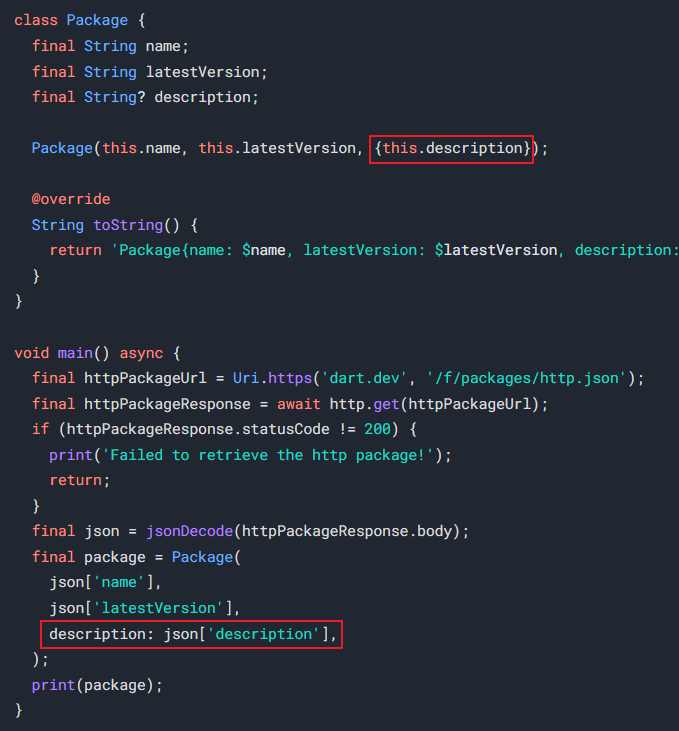


## 位置参数



## 命名参数





## 覆盖基类方法



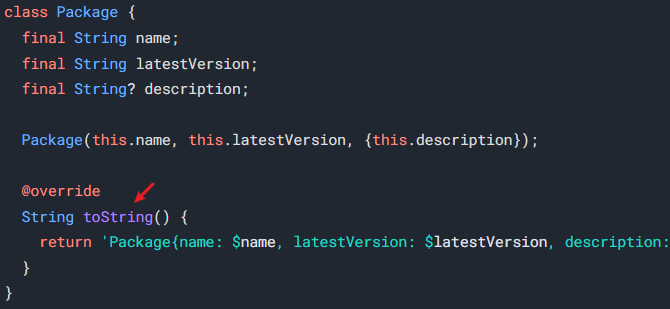
## 字符串插值



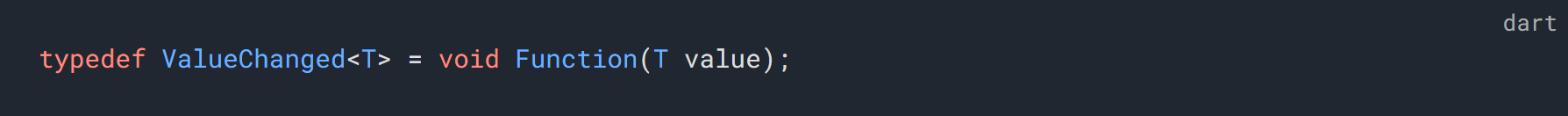
## toString() 方法

每个 Dart 对象默认都有一个 toString() 方法，用于生成该对象的字符串描述。

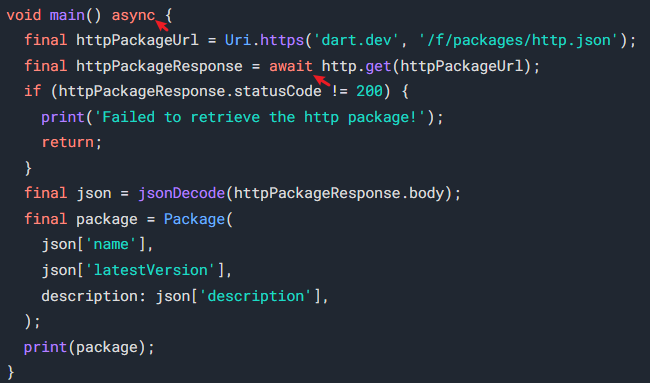
如果不重写，打印该对象时会得到类似 Instance of 'Package' 这种无用信息。



## 类型定义

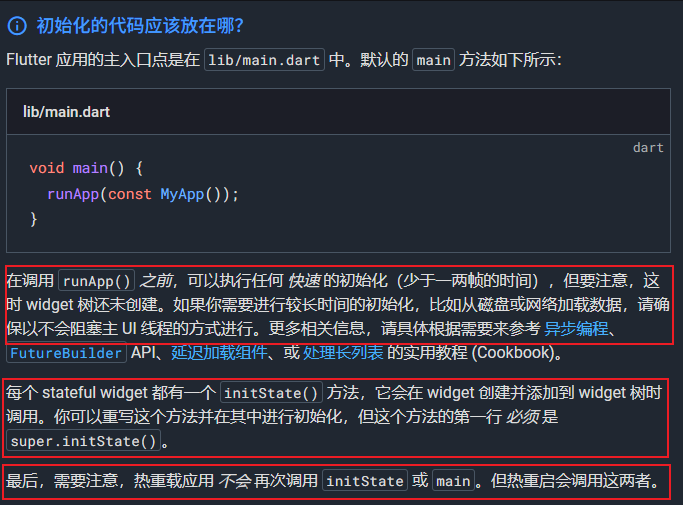


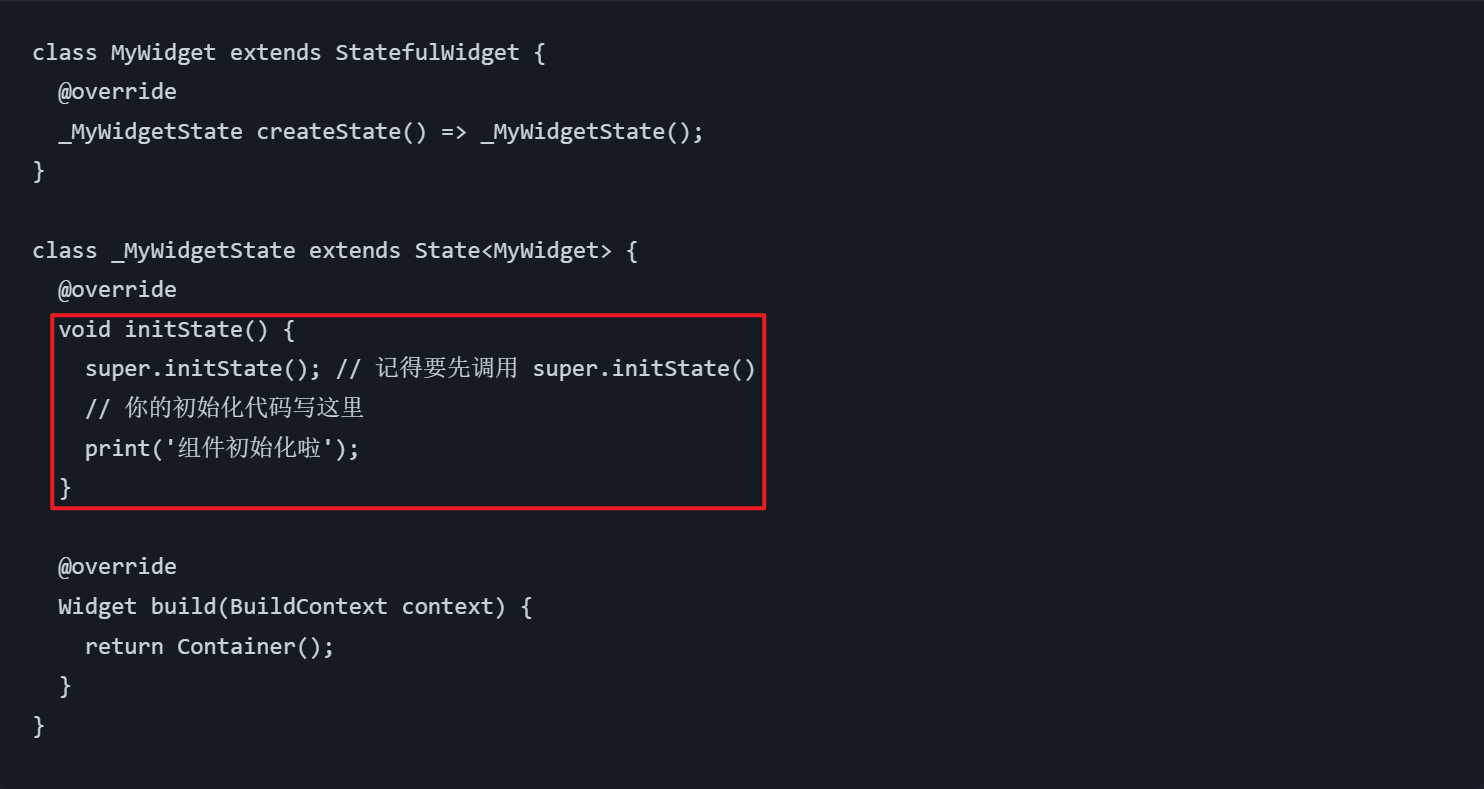
## await必须在async函数中能用



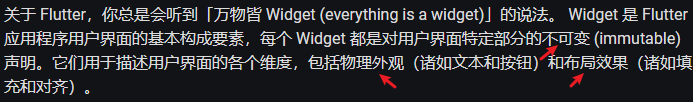


## 初始化代码





# Widget

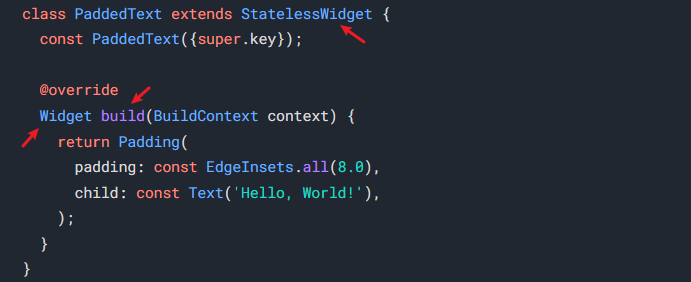


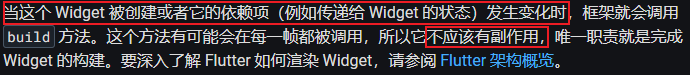
注意StatefulWidget也是不可变的，可变的是State。



## 构建Widget

在 Flutter 中创建用户界面时，你需要重写 Widget 对象的 build 方法。所有的 Widget 都必须包含一个 build 方法，并且该方法必须返回新的 Widget 实例。



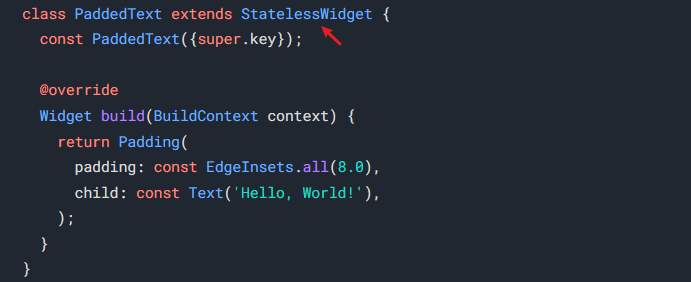


传递给Widget的状态指的是外部状态，如InheritedWidget提供的数据。

## Widget 状态

### 无状态 Widget（StatelessWidget）

自身不包含可变状态，属性一旦初始化后便不可更改，不会自动刷新界面（执行build）



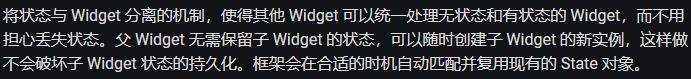
### 有状态 Widget（StatefulWidget）

自身包含可变状态，状态改变会自动刷新界面（执行build），但注意这个并不是Widget本身实现的，而是Widget对应的State实现的。



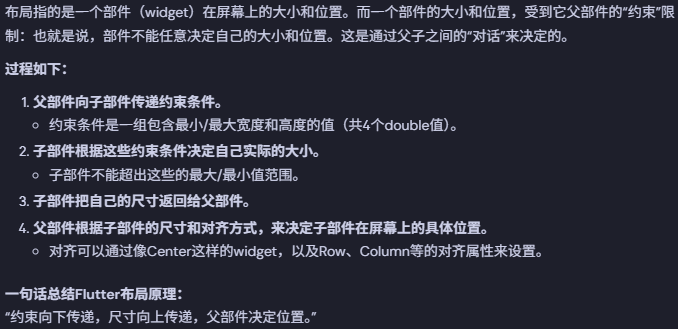




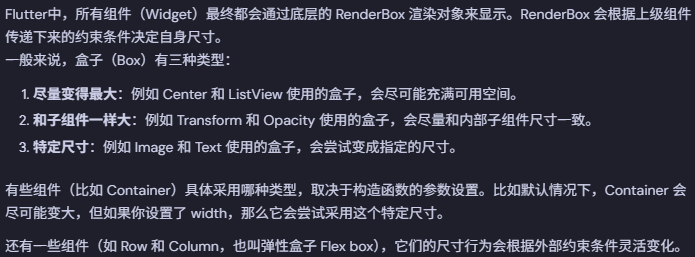


# 布局

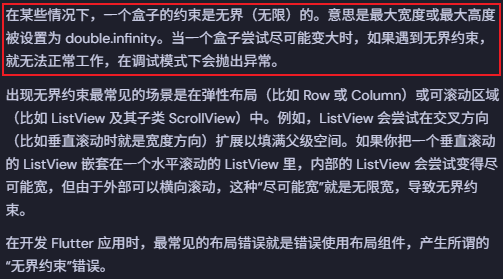
## 约束（Constraints）



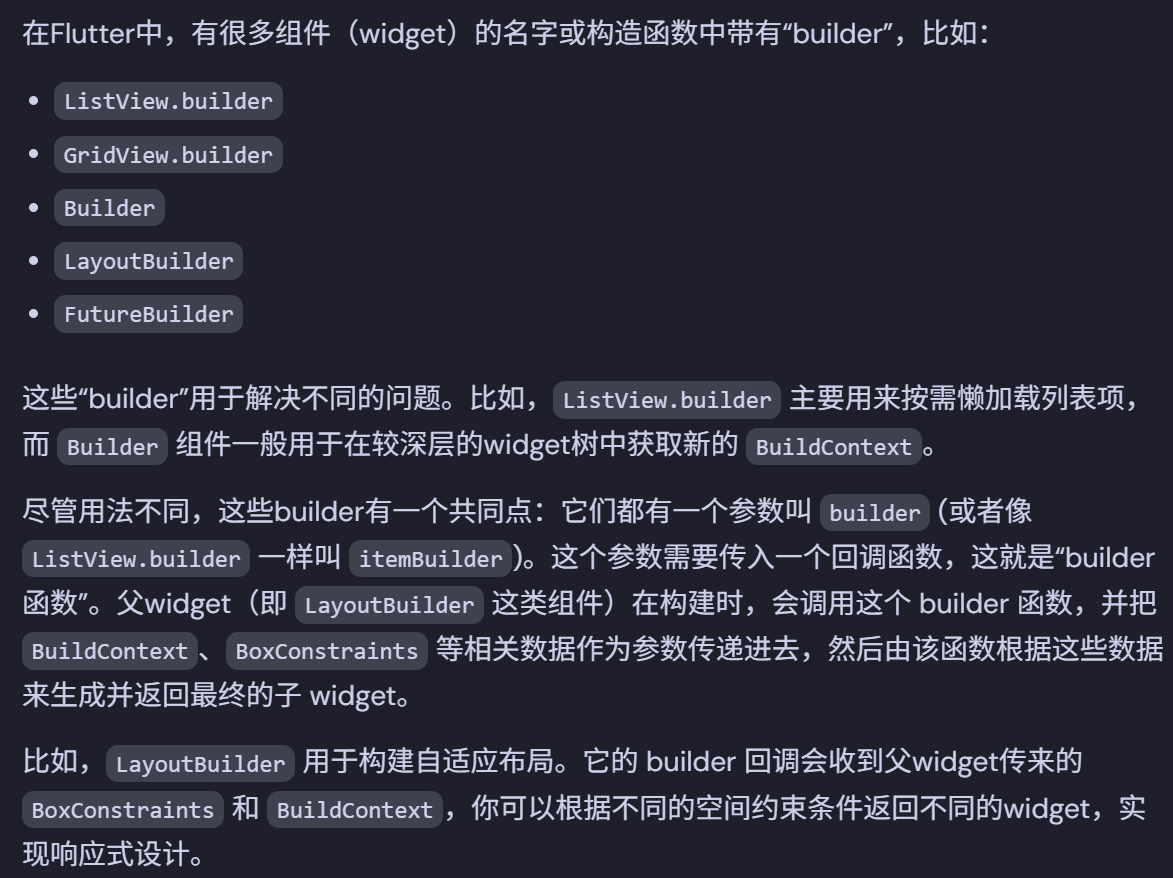
## 盒子类型



## 无界约束（unbounded constraints）



## builder

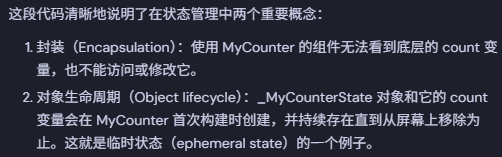


# 状态管理

## StatefulWidget

内部状态

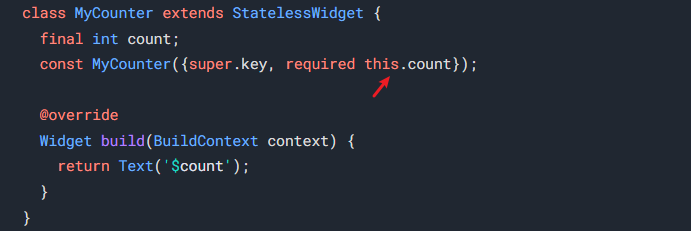




## constructors

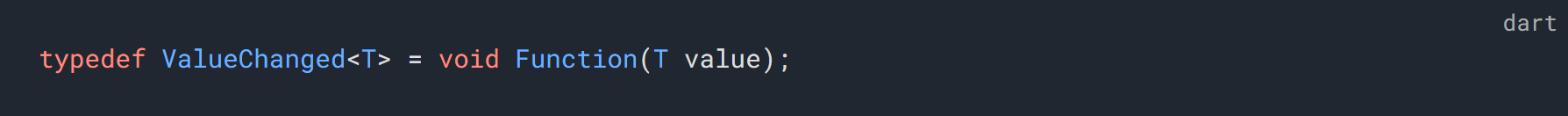
父传子

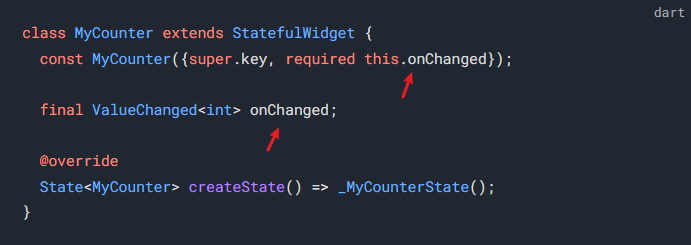


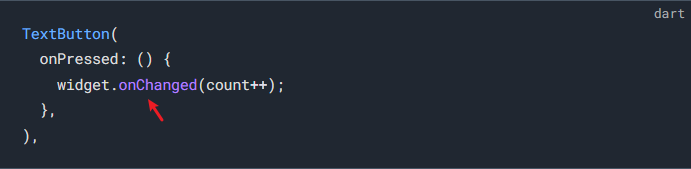


## callbacks

子传父

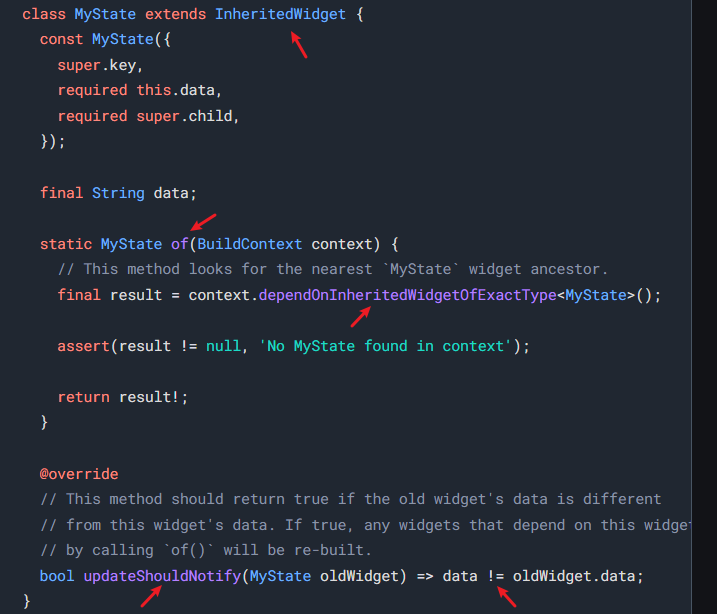






## InheritedWidget

让 widget 树中较低层级的 widget 能轻松访问更高层级 widget 提供的数据，而无需一层一层地手动传递参数。





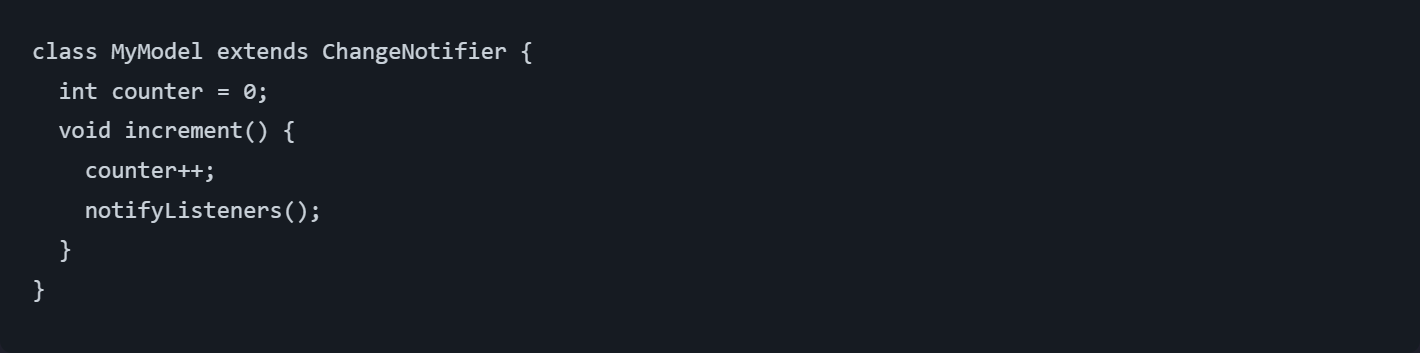
InheritedWidget 本身只是数据载体。要让数据更新，通常需要外部（父 widget，比如 StatefulWidget）调用 setState，构建新的 InheritedWidget 实例来实现。

## listenables



常见的Listenable子类：ChangeNotifier、ValueNotifier、AnimationController。

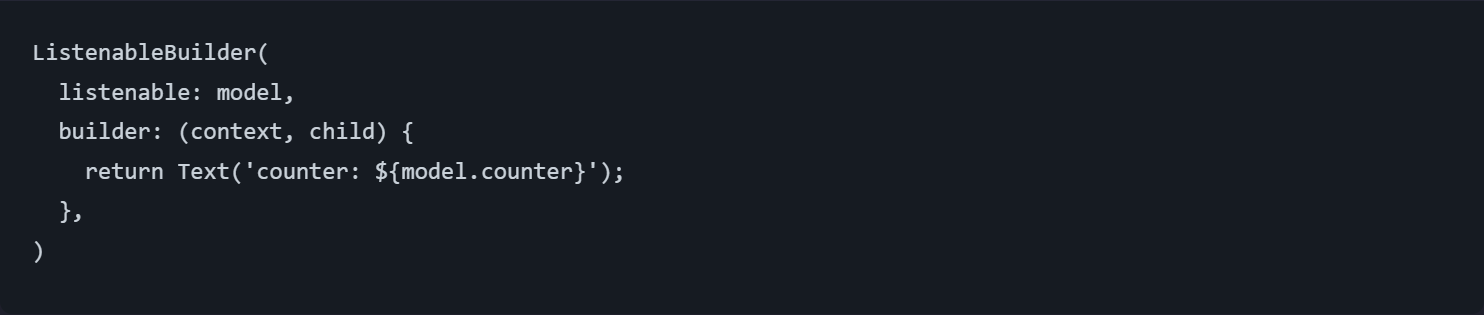
被监听者



监听（手动addListener）



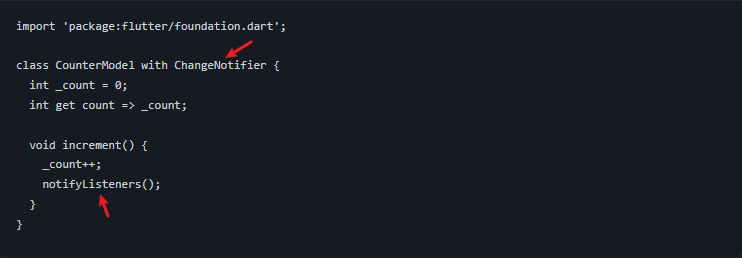
监听（ListenableBuilder）





## Provider（库）

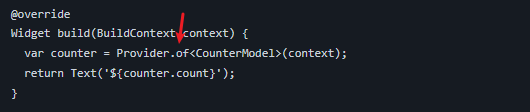
基于 InheritedWidget 的状态管理库。

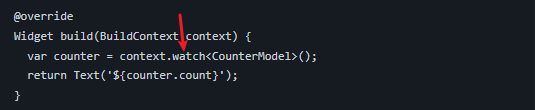


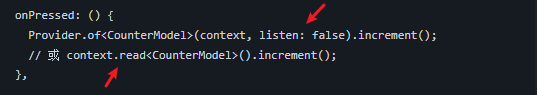


### 子组件访问状态- Provider.of<T>(context) 或 context.watch<T>()

官方强推写法，适合整体rebuild

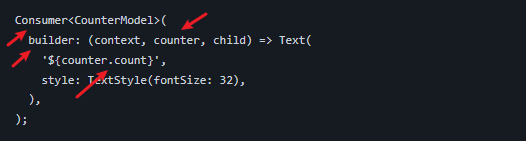






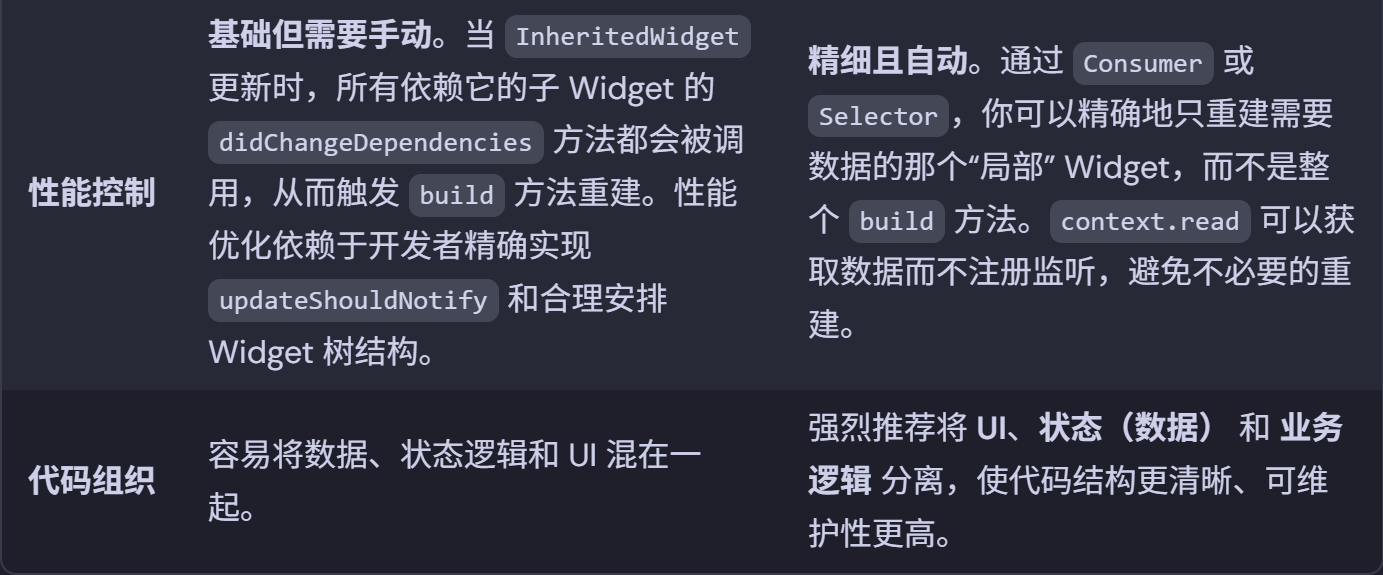
### 子组件访问状态- Consumer

经典用法，适合局部刷新

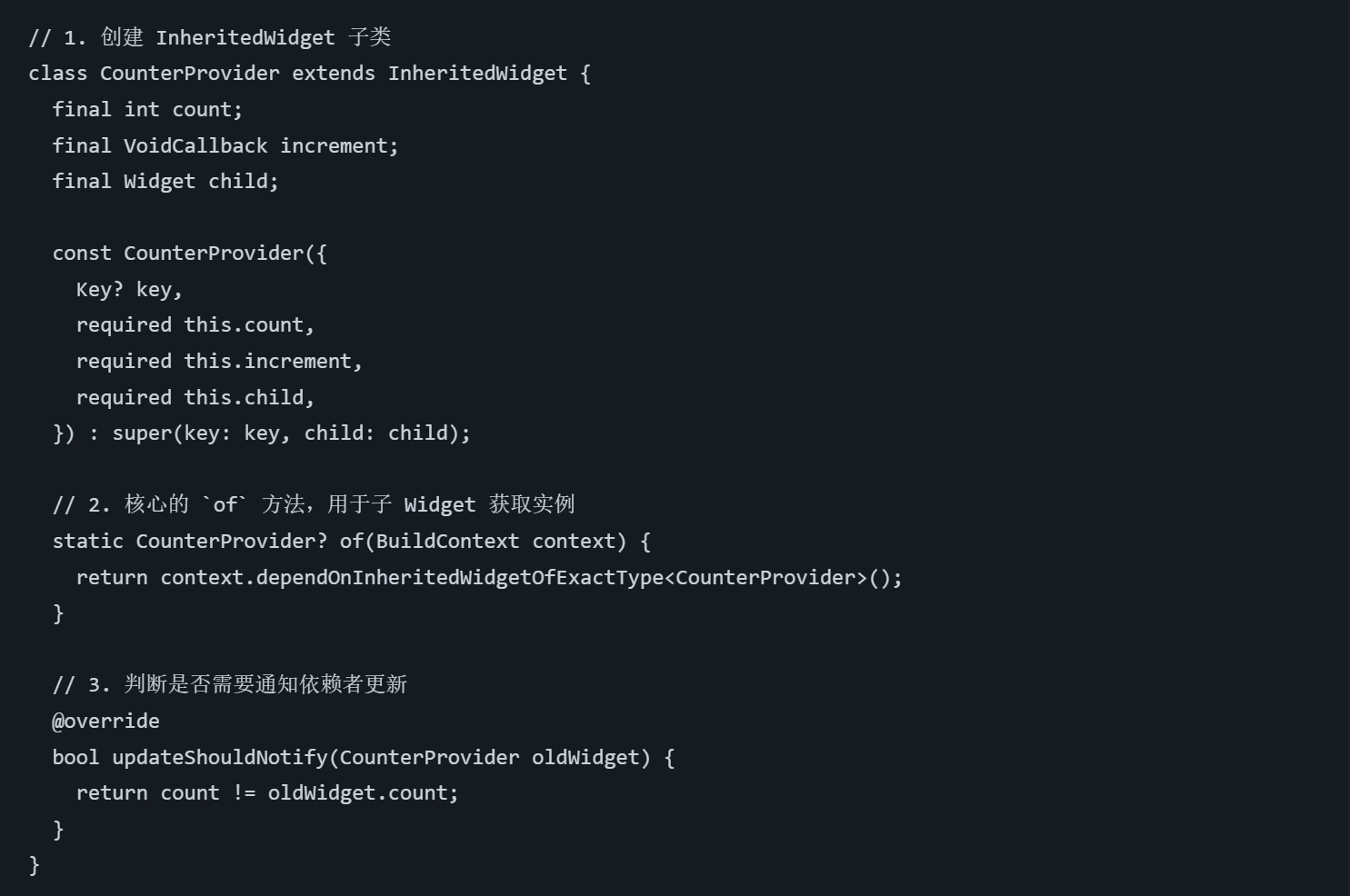


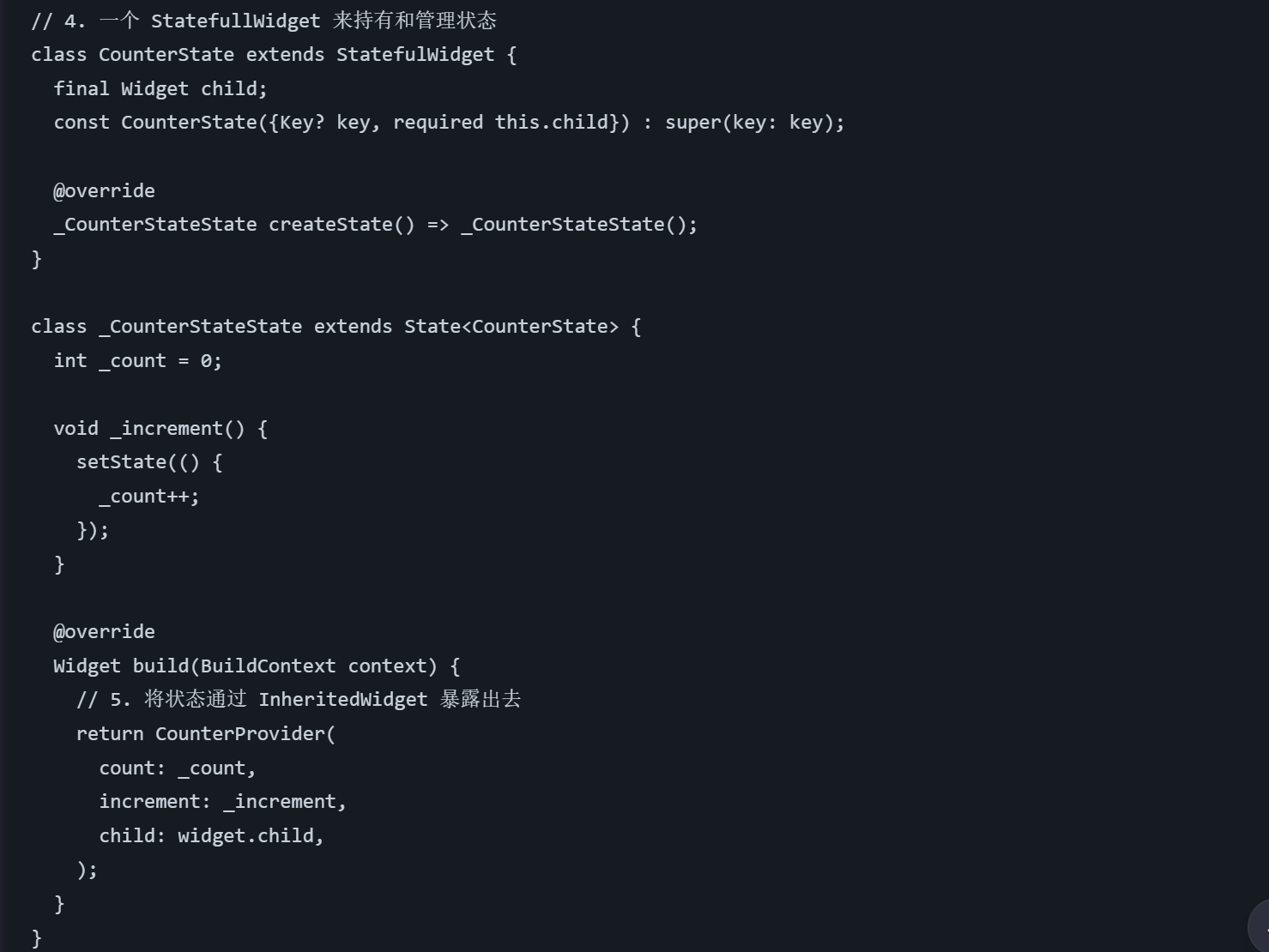
### InheritedWidget和Provider（库）

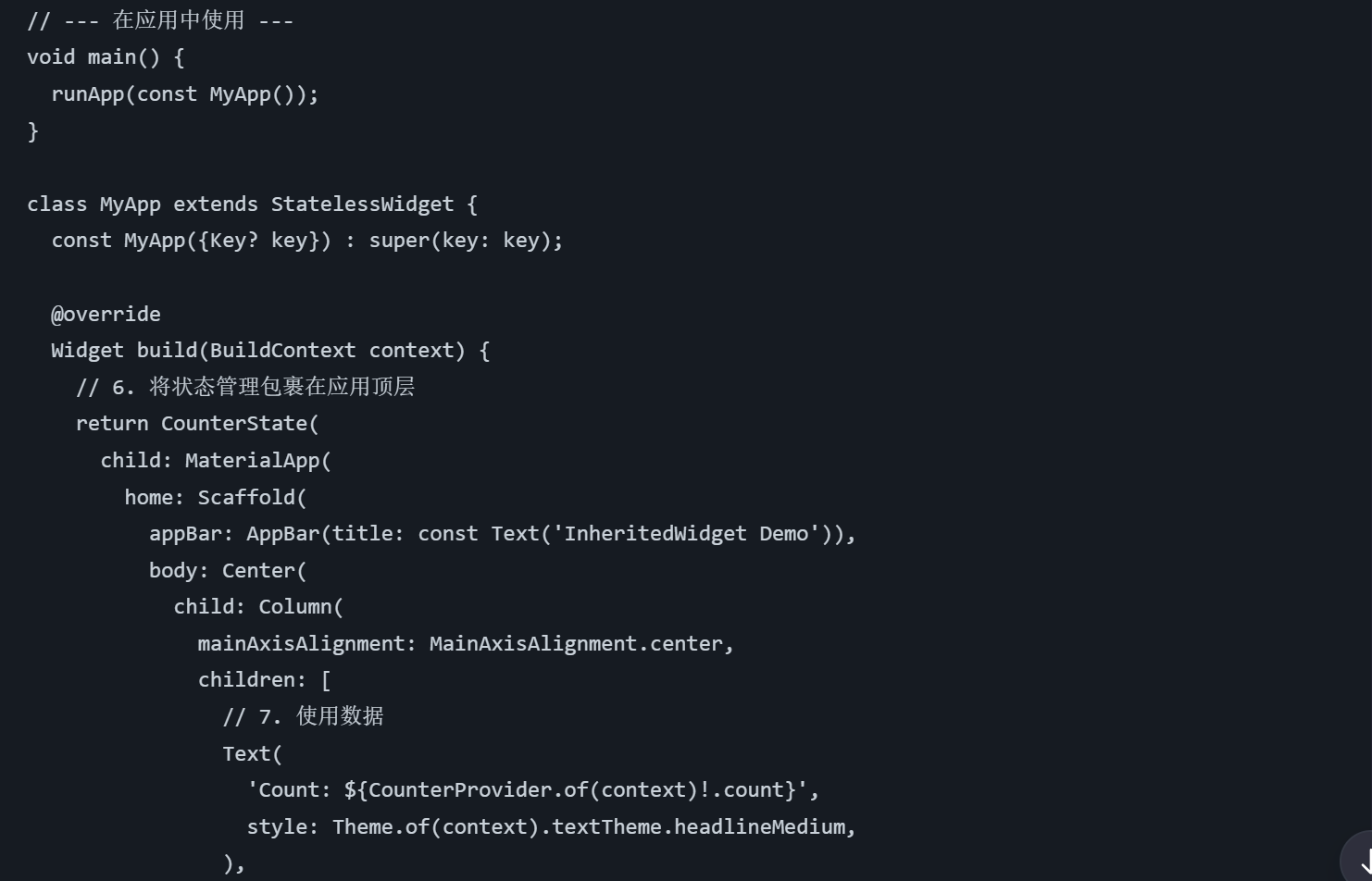


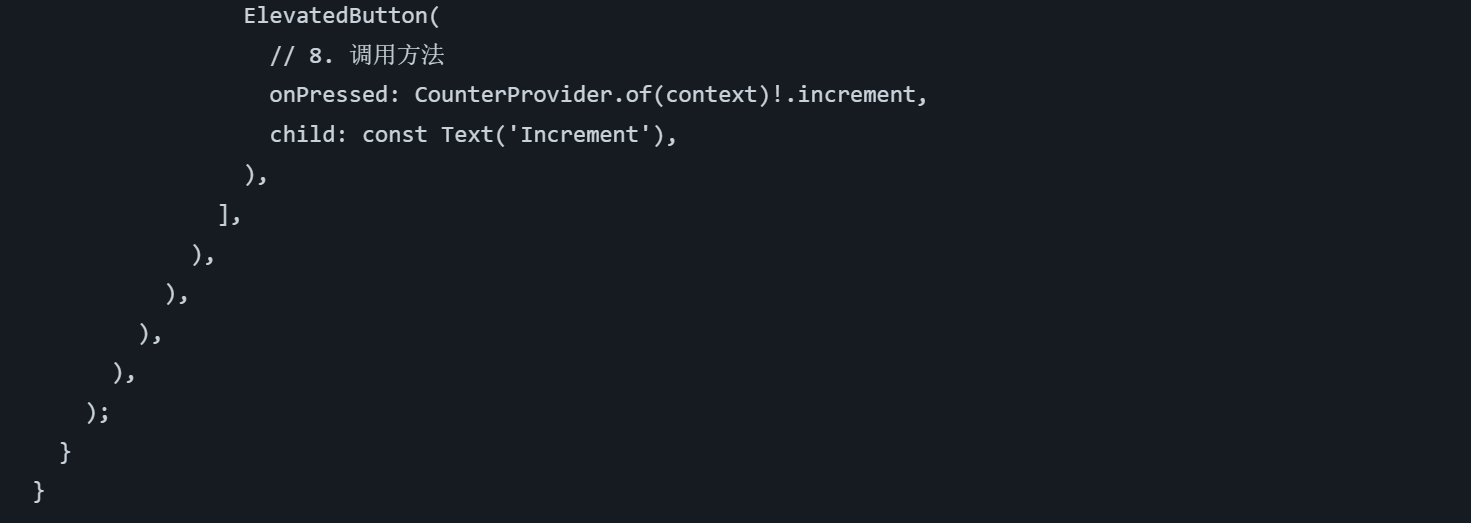


InheritedWidget



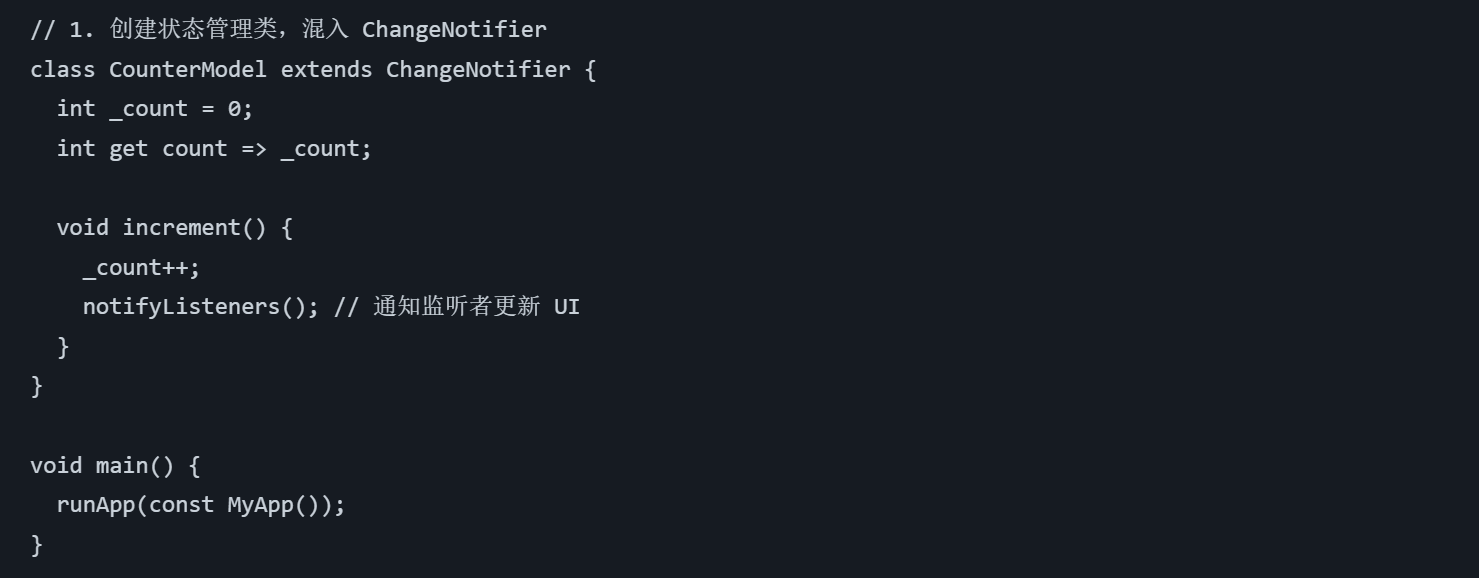


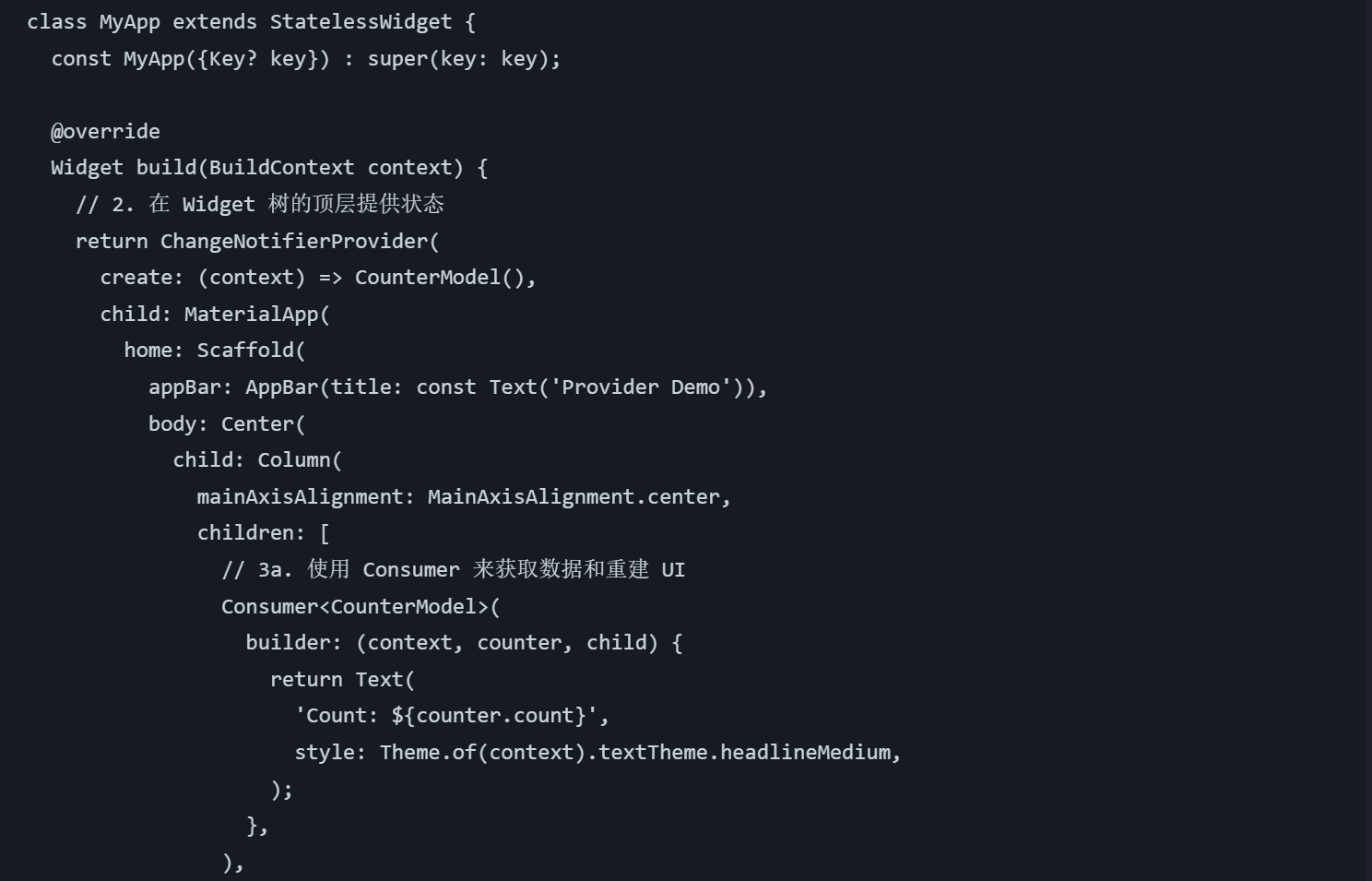




可以看出问题：InheritedWidget需要用State包裹，通过State状态改变后重建InheritedWidget才能改变InheritedWidget的成员并发出通知；子组件想改变InheritedWidget成员，需要调用State的接口改变状态，但子组件又访问不了State，只能把State改变状态的接口传给InheritedWidget，子组件才能通过InheritedWidget调用这个State接口。

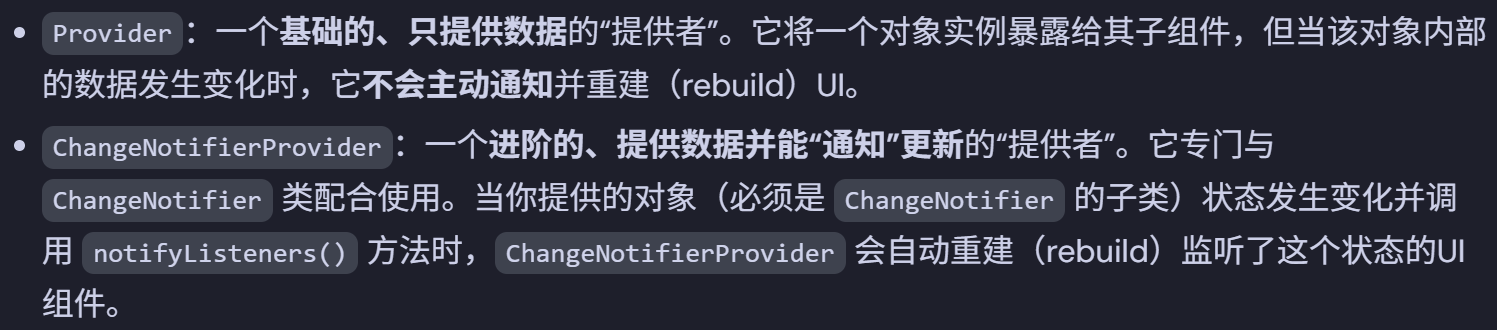
Provider（库）







### Provider（类）和ChangeNotifierProvider（类）



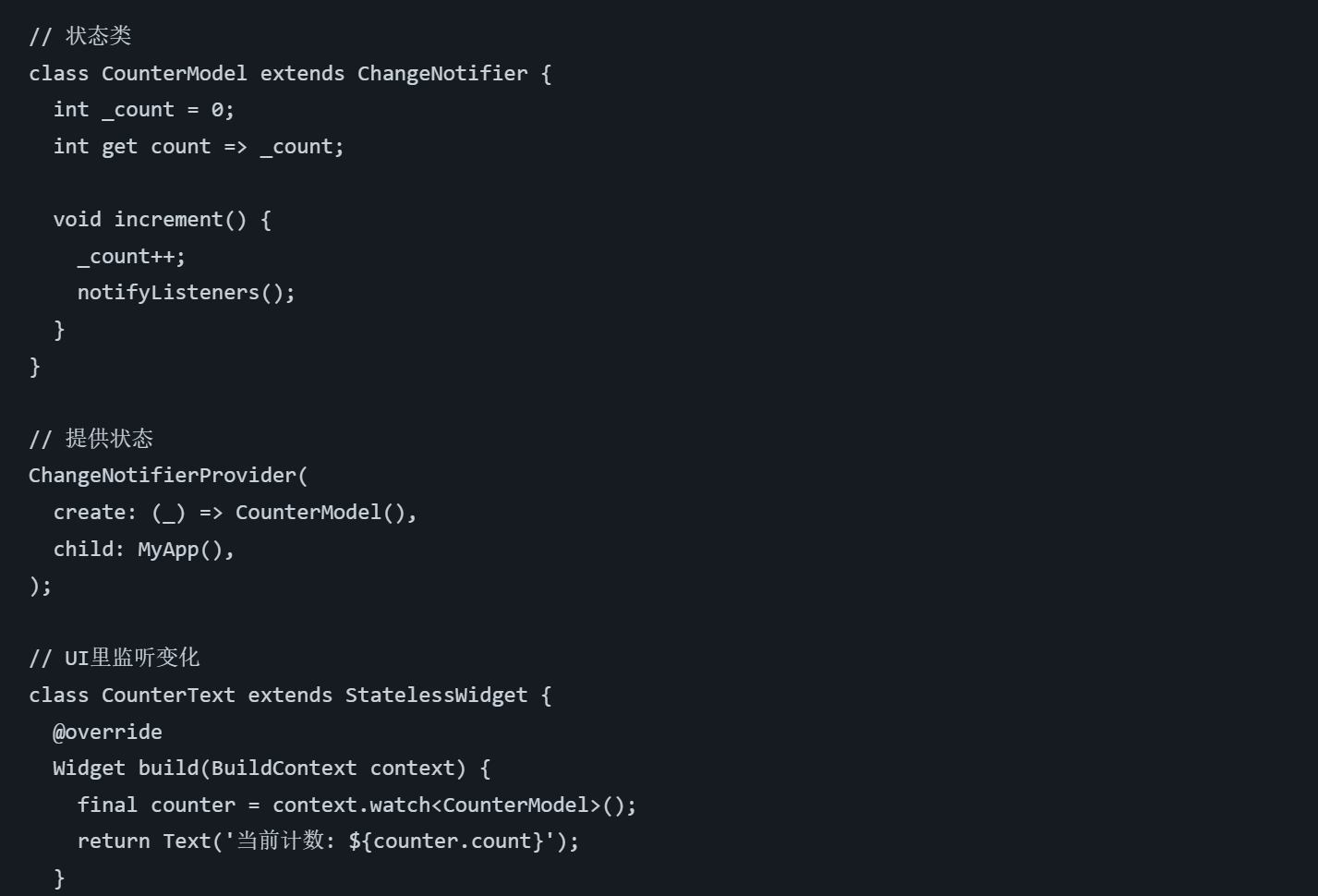


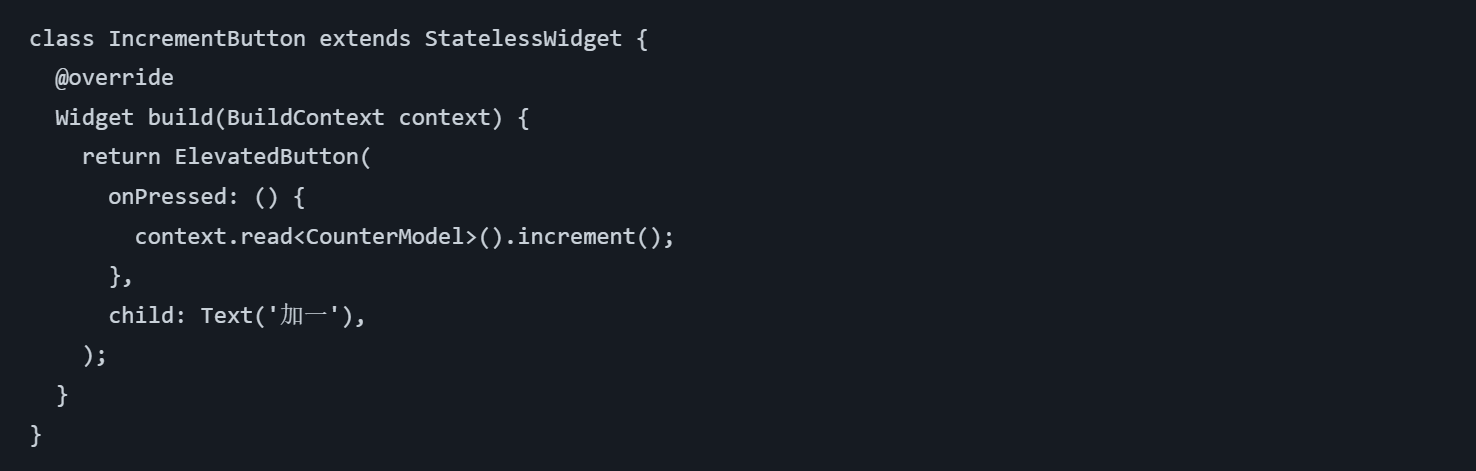
Provider（类）



AppConfig 变了，VersionShow 这个UI不会刷新。Provider只是“注入”，不卡着UI联动刷新。

ChangeNotifierProvider（类）

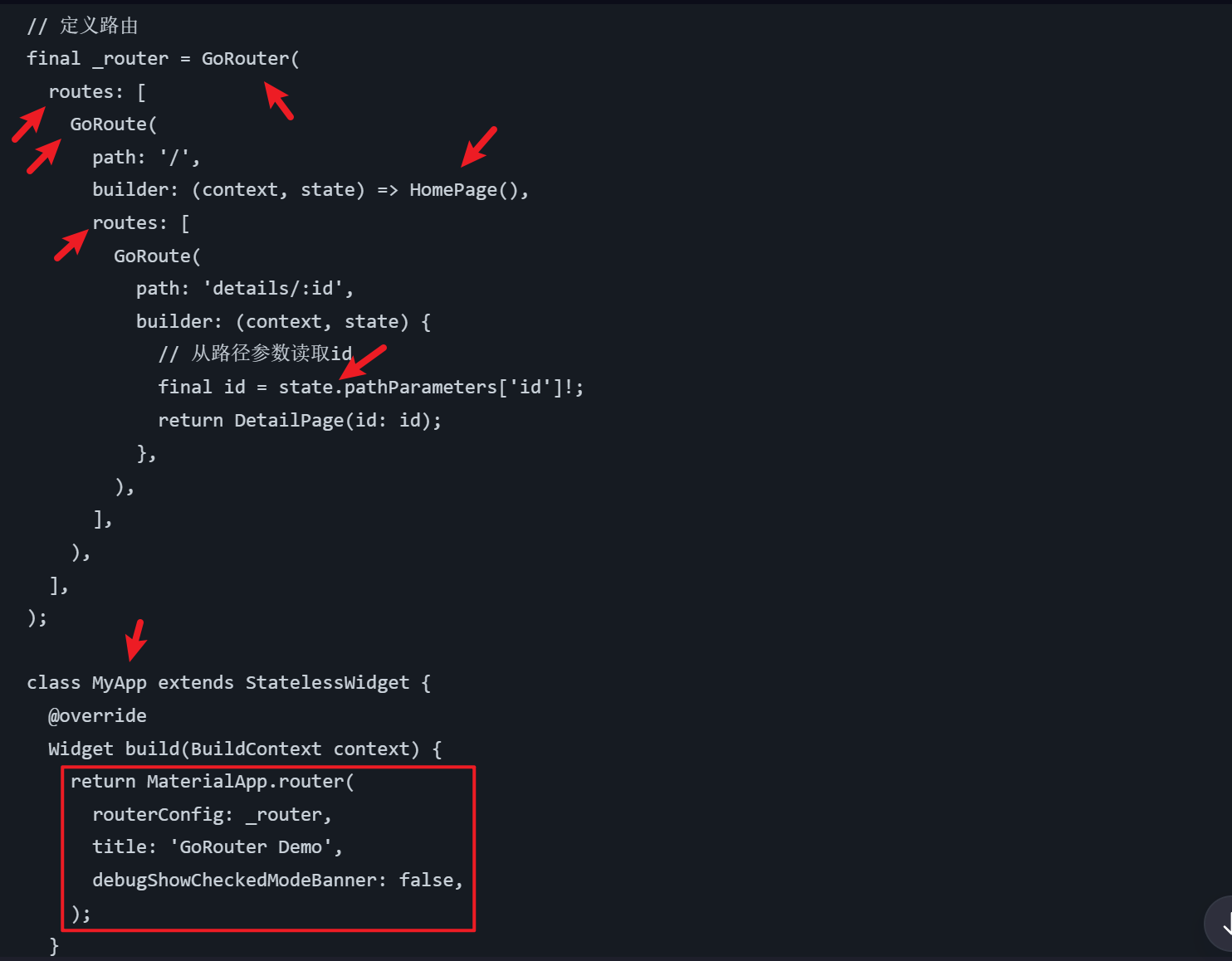




点击按钮后，counter.count 就会变，CounterText 这个UI会立即响应变化自动刷新。

# 路由

## go\_router库

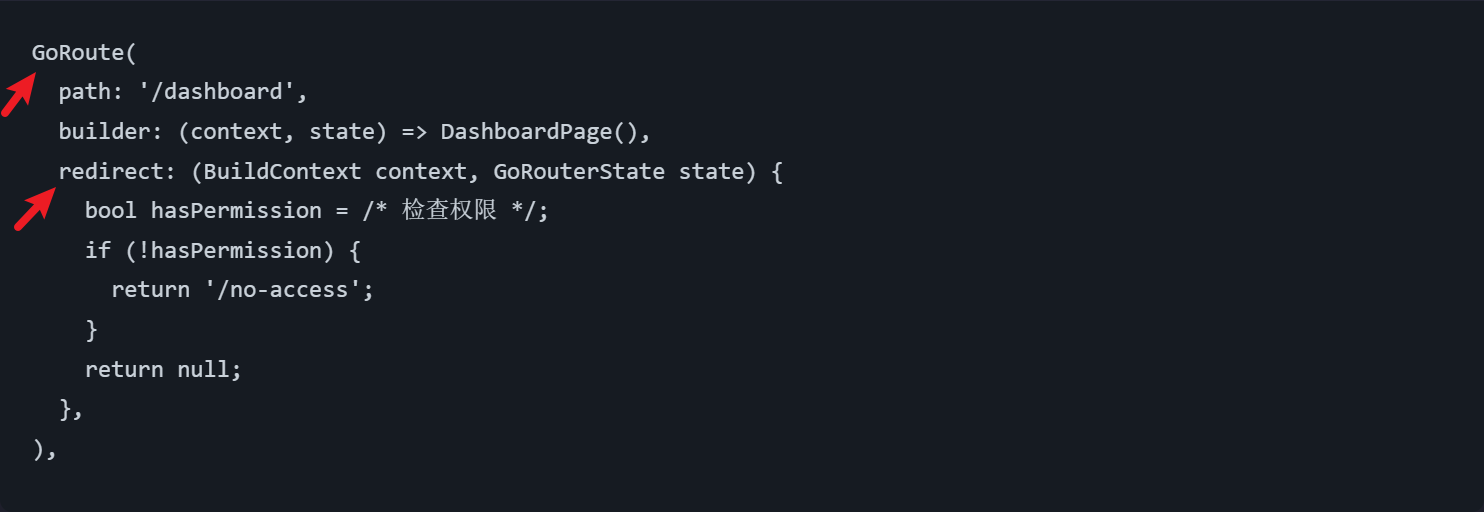


## 重定向

### 全局重定向



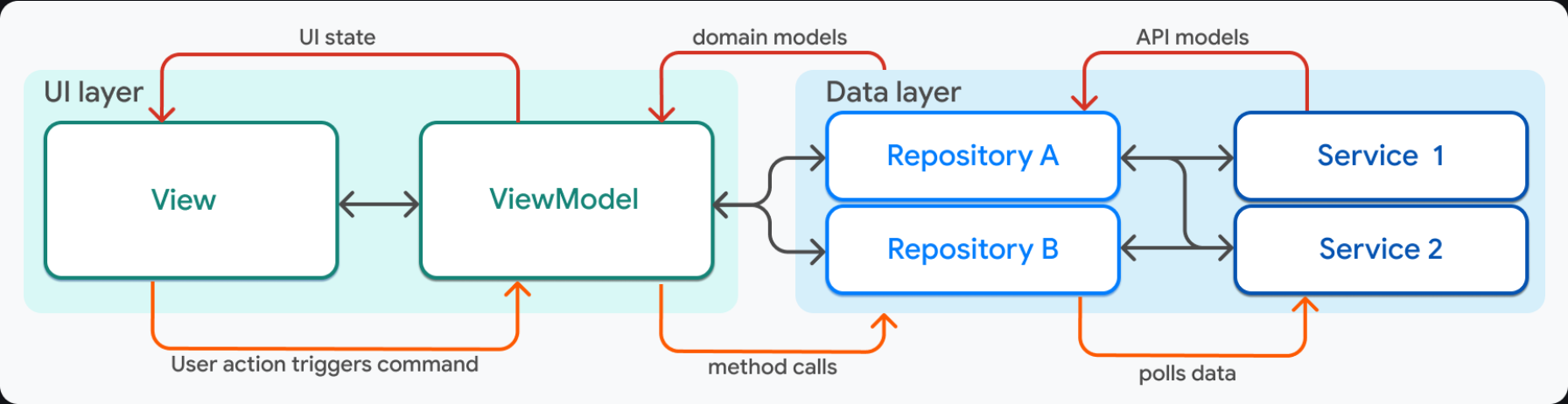
### 路由分支（分路由）重定向



### 动态参数重定向



# MVVM架构

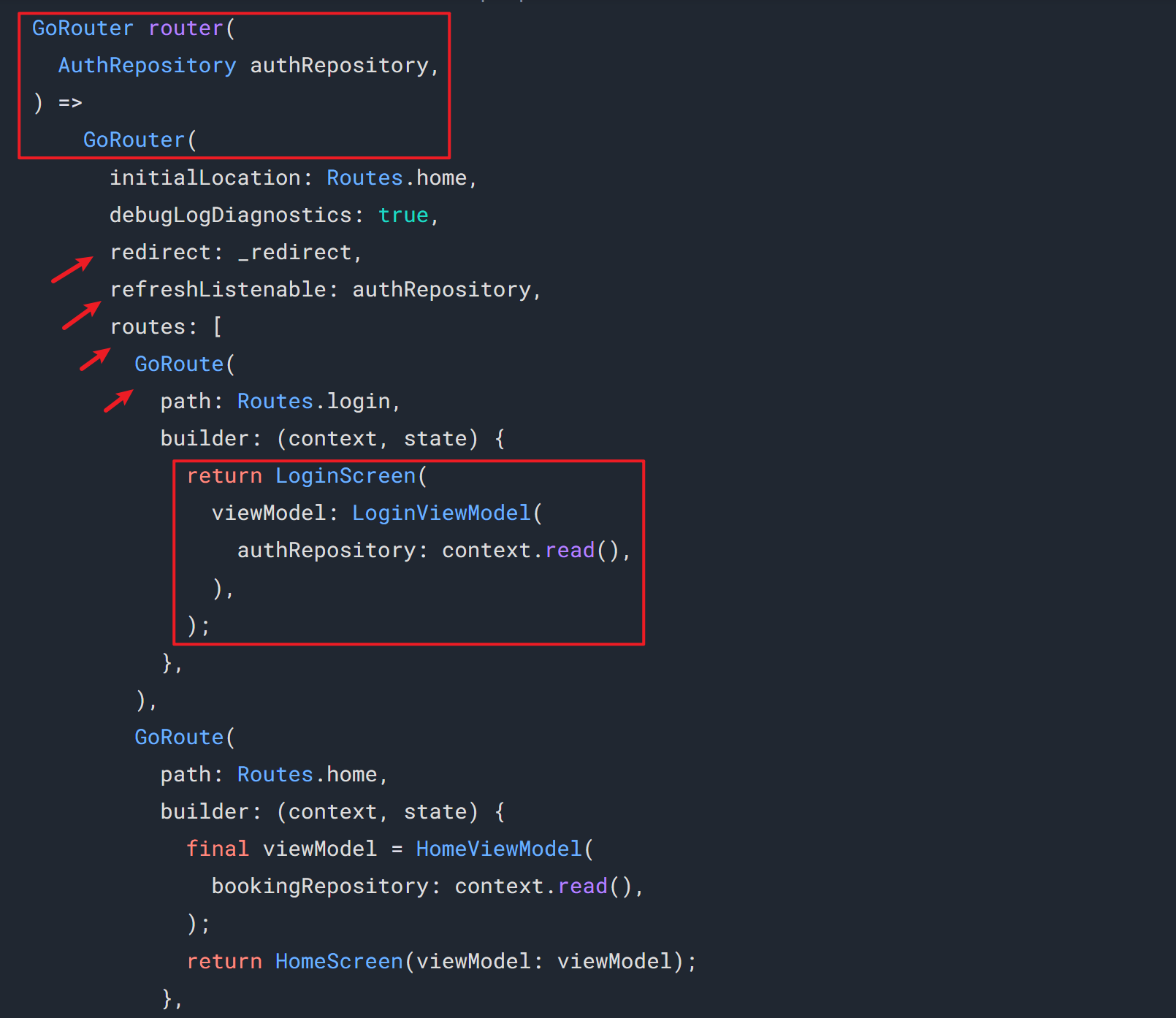


MVC通过Controller手动管理View和Model之间的交互，而MVVM通过数据绑定机制实现了View和ViewModel之间的自动同步，从而达到更彻底的解耦。

## 数据层通过provider共享给UI层

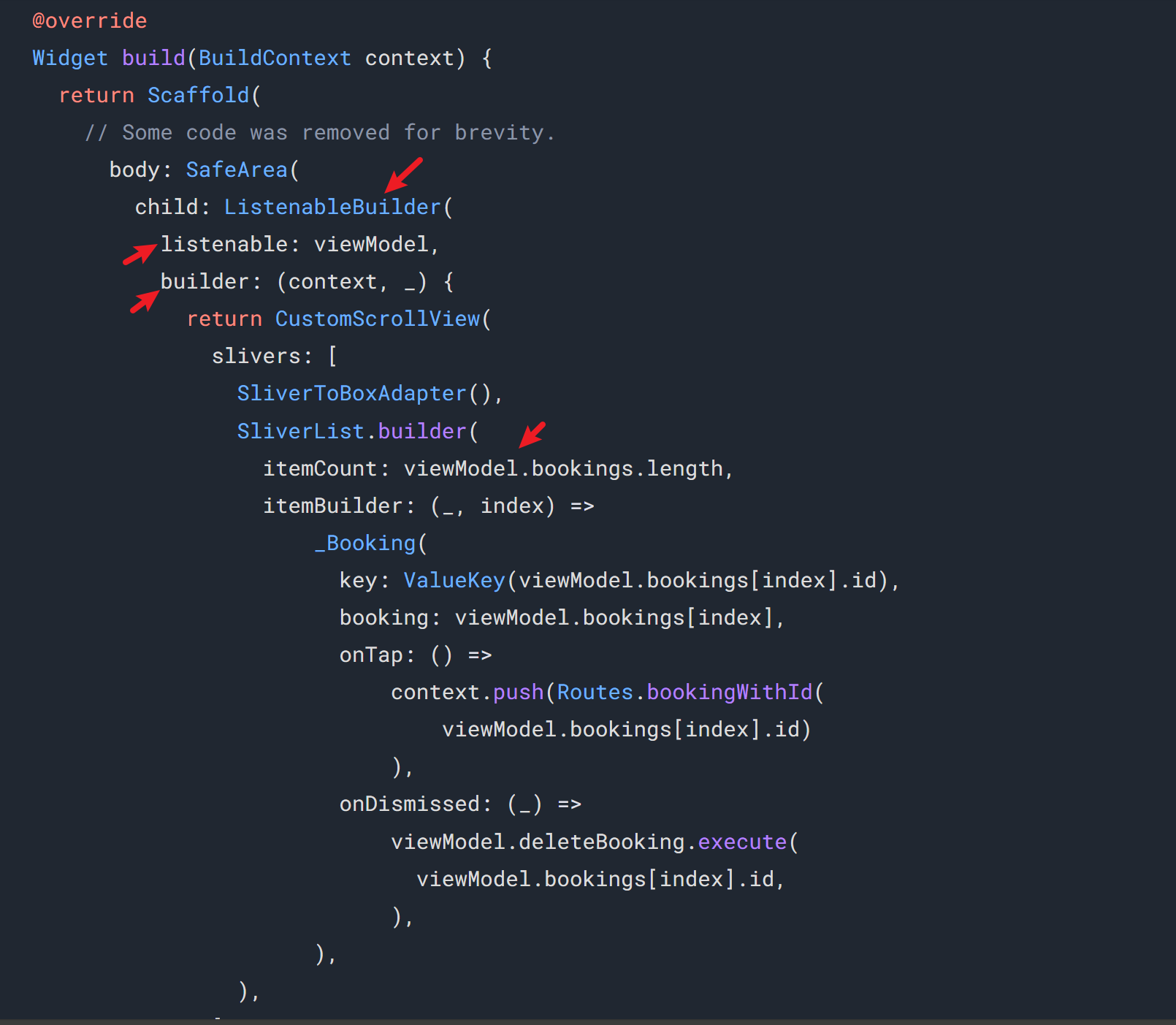


## 界面层路由及重定向



## View和ViewModel

### View监听和获取ViewModel中的状态



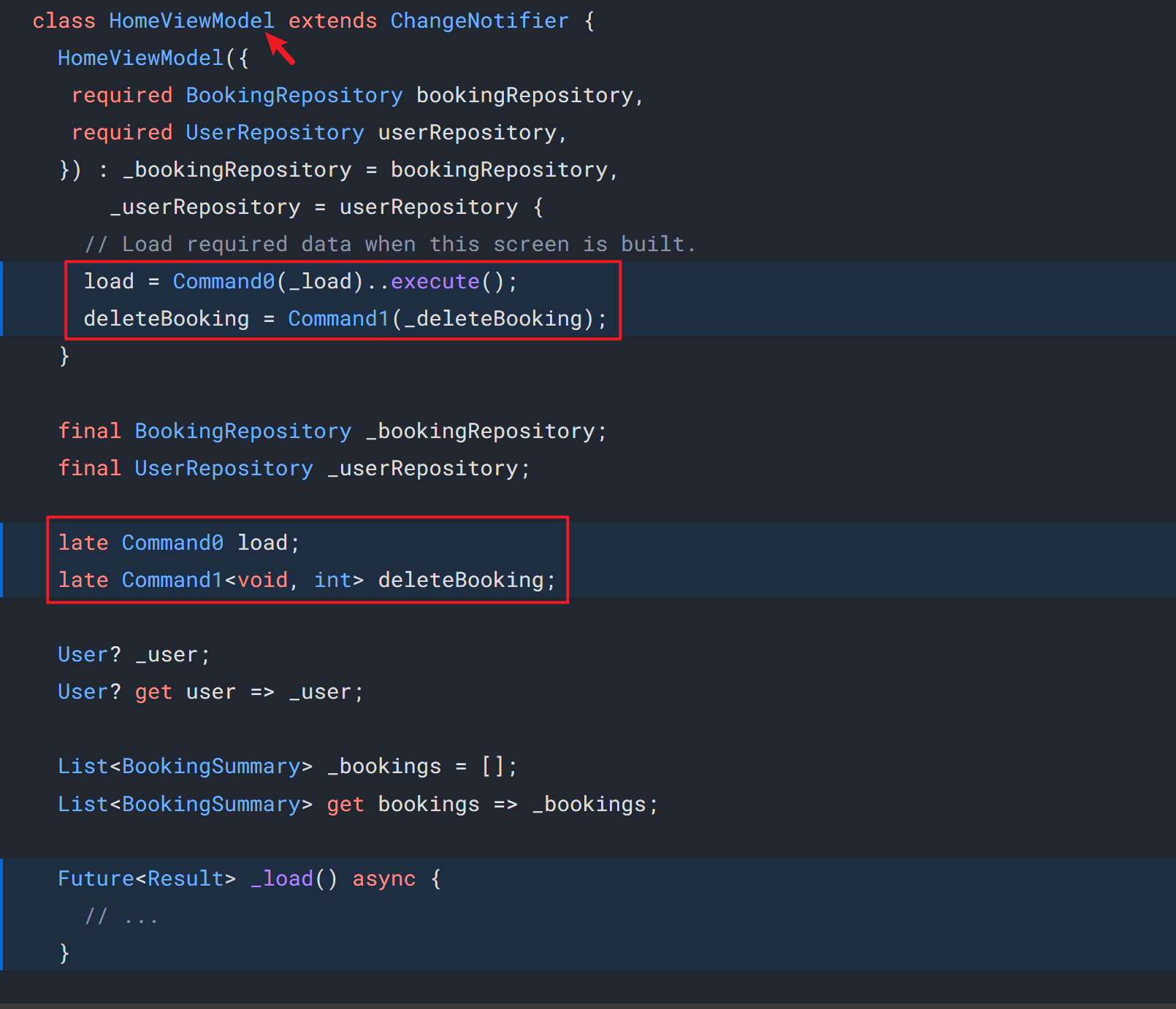
### View使用VIewModel处理业务

通过命令模式，可以统一处理异步调用、命令的状态等，支持排队和撤销，也可以根据命令不同的状态，展示不同的界面。

定义命令



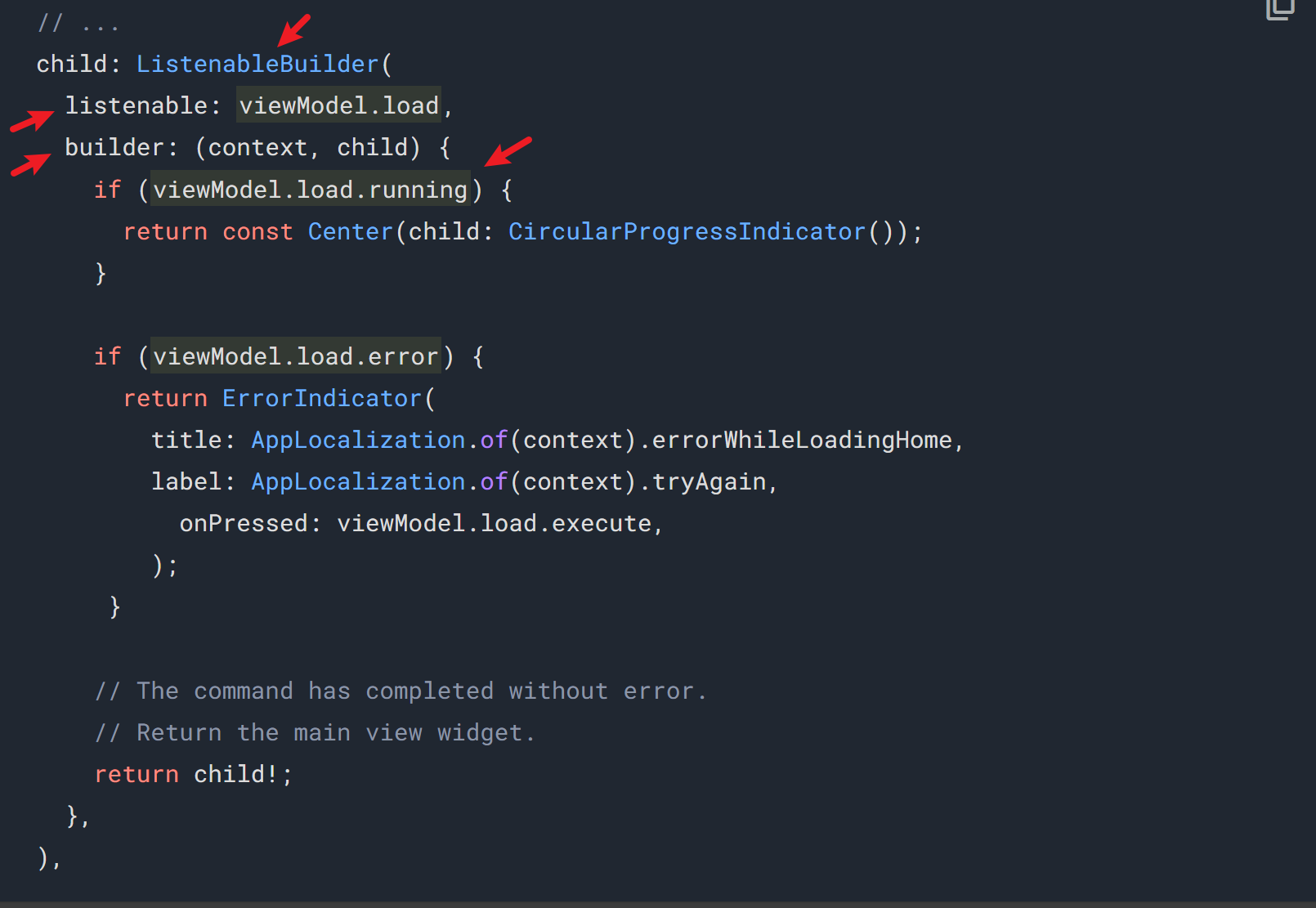
VIewModal创建和持有命令



View使用命令



View监听命令和获取命令状态

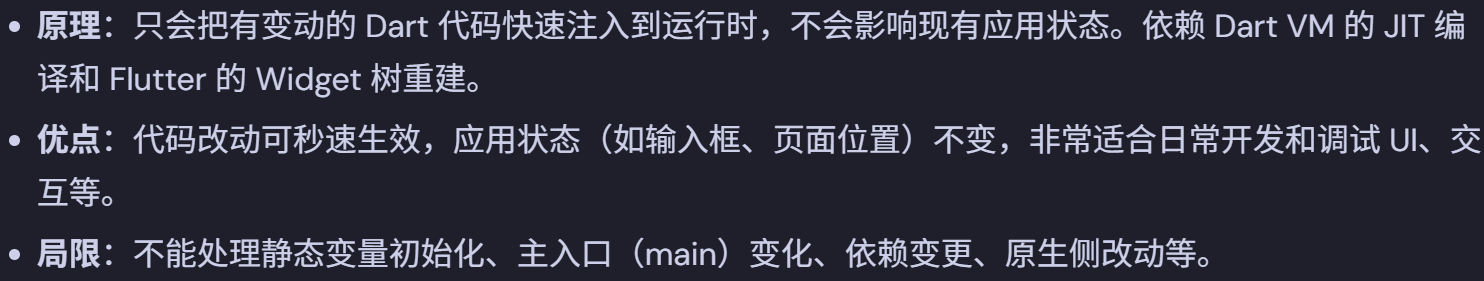


# Flutter相关概念

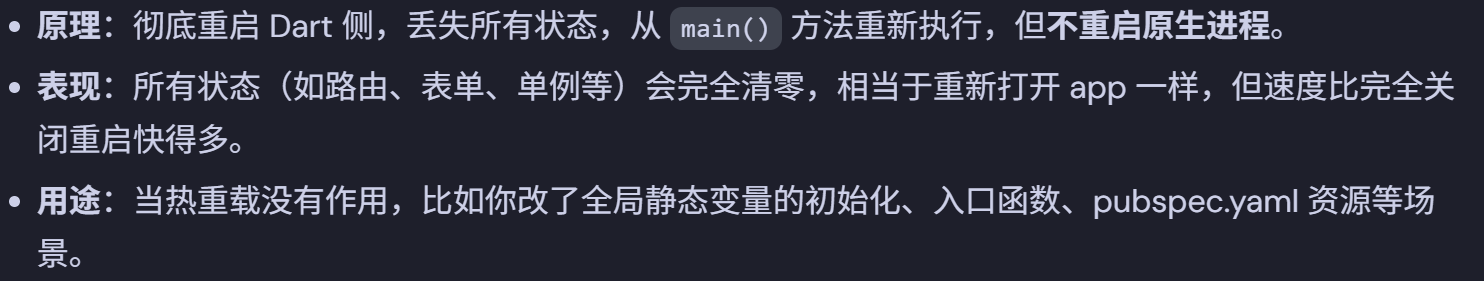
## Dart 侧与原生侧



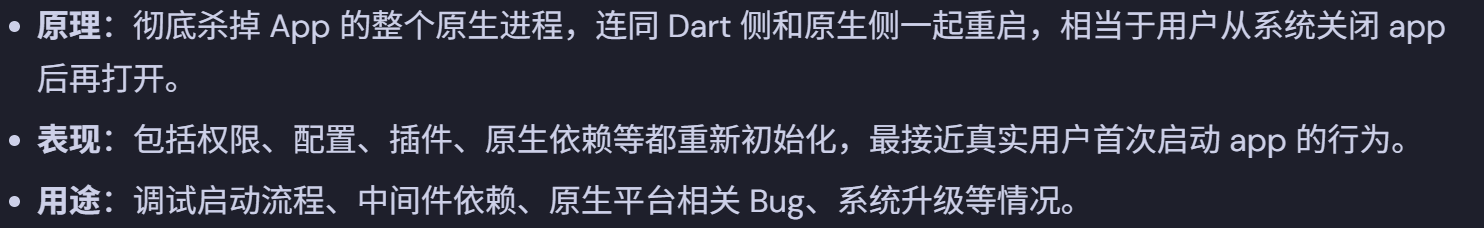
## 热重载（Hot Reload）



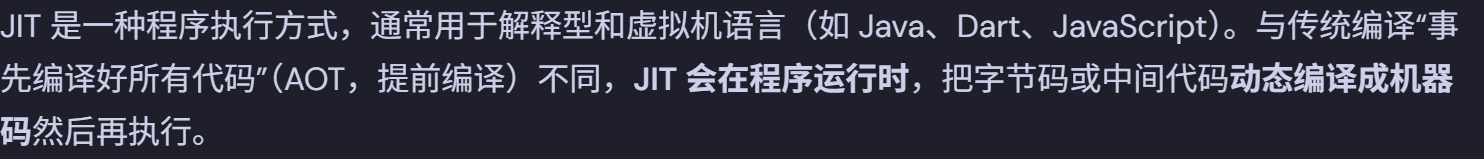
## 热重启（Hot Restart）



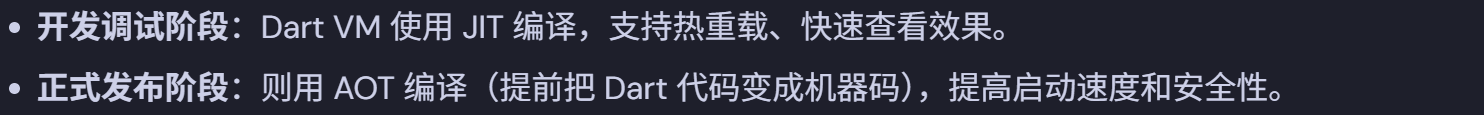
## 冷重启（Cold Restart）



## JIT（Just-In-Time Compilation（即时编译））







## 与RN性能对比

