

# 使用热重载

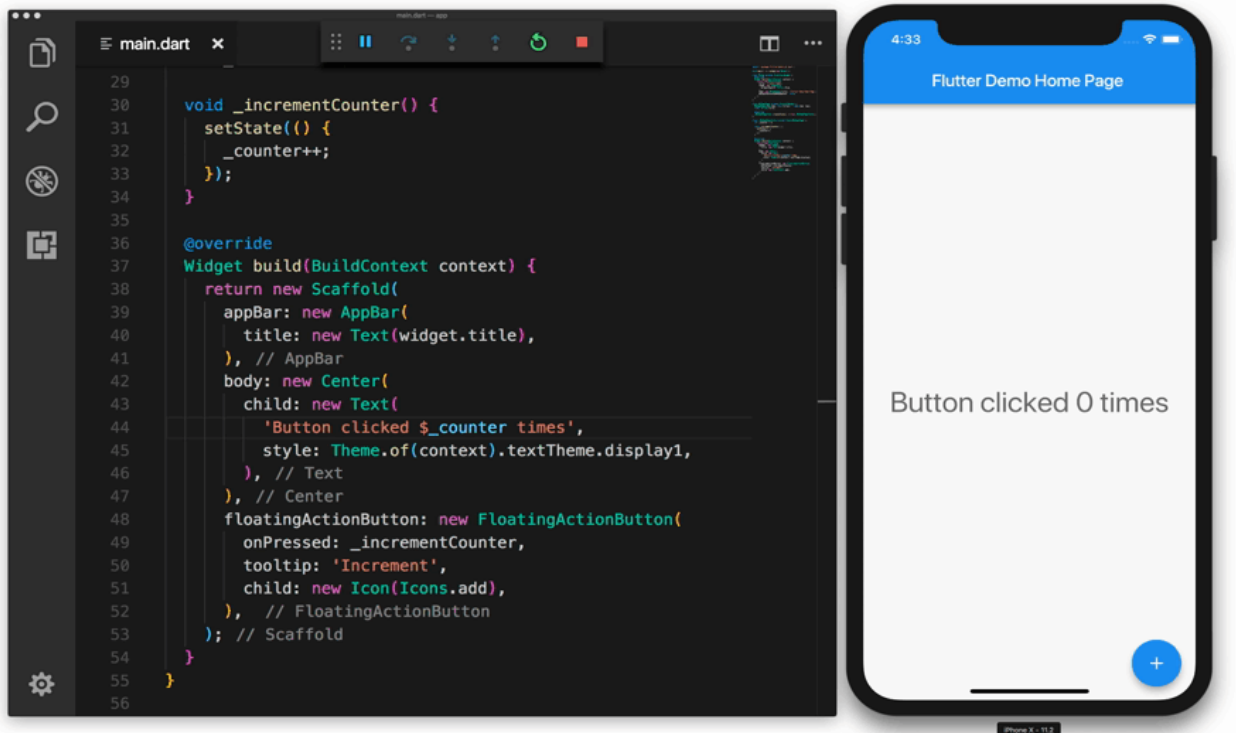
- [使用热重载](#)
- [编译错误](#)
- [之前的状态与新代码结合在一起](#)
- [包含最近的代码更改，但排除了应用程序状态](#)
- [最近的UI更改被排除在外](#)
- [限制](#)

## 使用热重载

Flutter的热重载(hot reload)功能可以帮助您在无需重新启动应用的情况下快速、轻松地进行测试、构建用户界面、添加功能以及修复错误。通过将更新后的源代码文件注入正在运行的Dart虚拟机 ( VM ) 中来实现热重载。在虚拟机使用新的的字段和函数更新类后，Flutter框架会自动重新构建widget树，以便您快速查看更改的效果。

要热重载一个Flutter应用程序:

1. 从受支持的IntelliJ IDE或终端窗口运行应用程序。物理机或虚拟机都可以运行。
2. 修改项目中的一个Dart文件。大多数类型的代码更改可以重新加载; 有关需要完全重新启动的更改列表，请参阅[限制](#)。
3. 如果您使用的是IntelliJ IDE，请选择**Save All** (`cmd-s/ctrl-s`)，或者单击工具栏上的Hot Reload按钮。



alt\_text

如果您正在使用命令行 `flutter run` 运行应用程序，请在终端窗口输入 `r`

成功执行热重载后，您将在控制台中看到类似于以下内容的消息：

```
Performing hot reload...
```

```
Reloaded 1 of 448 libraries in 2,777ms.
```

应用程序已更新以反映您的更改，并且该应用程序的当前状态（以上示例中的计数器变量的值）将保留。您的应用程序将继续执行之前运行热重载命令的位置。代码被更新并继续执行。

A code change has a visible effect only if the modified Dart code is run again after the change. The next sections describe common situations where the modified code will *not* run again after hot reload. In some cases, small changes to the Dart code will enable you to continue using hot reload for your app. 只有修改后的Dart代码再次运行时，代码更改才会有效果。接下来的部分将介绍常见的情况，即修改后的代码在热重载后不会再运行。在某些情况下，对Dart代码的小改动将使您能够继续为您的应用程序使用热重新加载。

## 编译错误

当代码更改后引入了编译错误时，热重载会生成类似于以下内容的错误消息

```
Hot reload was rejected:
```

```
'/Users/obiwan/Library/Developer/CoreSimulator/Devices/AC94F0FF-16F7-46C8-B4BF-218B73C547AC/data/Containers/Data/Application/4F72B076-42AD-44A4-A7CF-57D9F93E895E/tmp/ios_testWIDYdS/ios_test/lib/main.dart': warning: line 16 pos 38: unbalanced ' {' opens here
```

```
Widget build(BuildContext context) {
```

```
~/Users/obiwan/Library/Developer/CoreSimulator/Devices/AC94F0FF-16F7-46C8-B4BF-218B73C547AC/data/Containers/Data/Application/4F72B076-42AD-44A4-A7CF-57D9F93E895E/tmp/ios_testWIDYdS/ios_test/lib/main.dart': error: line 33 pos 5:
unbalanced ')'
```

```
    );
  ~
```

在这种情况下，只需纠正代码错误，就可以继续使用热重载。

## 之前的状态与新代码结合在一起

Flutter的热重载功能（有时称为有\_状态热重载\_）可保留您的应用程序的状态。这种设计使您只能查看最近更改的效果，而不会丢弃当前状态。例如，如果您的应用需要用户登录，则可以在导航层次结构中向下几个级别修改并重新加载页面，而无需重新输入登录凭据。状态保持不变，这通常是期望的行为。

如果代码更改会影响应用程序（或其依赖）的状态，则应用程序必须使用的数据可能与它从头开始执行的数据不完全一致。在热重载和完全重启之后，结果可能是不同的行为。例如，如果您将某个StatelessWidget类改为StatefulWidget（或相反），则在热重载之后，应用程序的以前状态将保留。但是，该状态可能与新的更改不兼容。

考虑下面的代码：

```
class myWidget extends StatelessWidget {
  Widget build(BuildContext context) {
    return new GestureDetector(onTap: () => print('T'));
  }
}
```

运行该应用程序后，如果进行以下更改：

```
class myWidget extends StatefulWidget {
  @override
  State createState() => new myWidgetState();
}

class myWidgetState {
  ...
  ...
}
```

然后热重载，控制台将显示类似于以下内容的断言失败：

```
myWidget is not a subtype of StatelessWidget
```

在这些情况下，需要完全重新启动以查看更新的应用程序。

## 包含最近的代码更改，但排除了应用程序状态

在Dart中，[静态字段惰性初始化](#)。这意味着你第一次运行一个Flutter应用程序并读取一个静态字段时，它的初始值设为初始表达式的结果。全局变量和静态字段被视为状态，因此在热重载期间不会重新初始化。

如果更改全局变量和静态字段的初始值设定项，则需要完全重启以查看更改。例如，请考虑以下代码

```
final sampleTable = [
  new Table("T1"),
  new Table("T2"),
  new Table("T3"),
  new Table("T4"),
];
```

运行该应用程序后，如果进行以下更改：

```
final sampleTable = [
  new Table("T1"),
  new Table("T2"),
  new Table("T3"),
  new Table("T10"), //modified
];
```

然后热重载，这个改变并没有生效

相反，在下面的例子中：

```
const foo = 1;
final bar = foo;
void onClick() {
  print(foo);
  print(bar);
}
```

第一次运行应用程序打印1和1。然后，如果您进行以下更改：

```
const foo = 2; //modified
final bar = foo;
void onClick() {
  print(foo);
  print(bar);
}
```

然后热重载，它现在打印2和1。对`const`字段值的更改始终会重新加载，但不会重新运行静态字段初始值设定语句(初始值可能是一个表达式的值)。从概念上讲，`const`字段被视为别名而不是状态。

Dart VM在一组更改需要完全重启才能生效的时候，会检测初始化程序更改和标志。

上述示例中的大部分初始化工作都会触发标记机制，但不适用于以下情况：

```
final bar = foo;
```

要能够在热重载后更新和查看`foo`的更改，请考虑将字段重新定义为`const`或使用getter来返回值，而不是使用`final`。例如：

```
const bar = foo;
```

or:

```
get bar => foo;
```

了解更多关于[const和final关键字的区别](#) in Dart.

## 最近的UI更改被排除在外

即使热重载操作看起来成功了并且没有抛出异常，但某些代码更改可能在刷新的UI中不可见。这种行为在更改应用程序的main()方法后很常见。

作为一般规则，如果修改后的代码位于根widget的构建方法的下游，则热重载将按预期运行。但是，如果修改后的代码不会因重建构建widget树而重新执行的话，那么在热重载后您将看不到其效果。

例如，请考虑以下代码：

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

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

```
class MyApp extends StatelessWidget {  
  Widget build(BuildContext context) {  
    return new GestureDetector(  
      onTap: () => print('tapped');  
    );  
  }  
}
```

运行这个应用程序后，您可能会更改代码，如下所示：

```
import 'package:flutter/widgets.dart';  
void main() {  
  runApp(  
    const Center(  
      child: const Text('Hello', textDirection: TextDirection.ltr)));  
}
```

完全重启后，程序从头开始执行新版本main()，并构建一个widget树显示文本“Hello”。

但是，如果您在此更改后重新加载应用程序，main()则不会重新执行，并且会使用未修改的实例MyApp作为新构建的widget树的根，热重载后结果没有变化。

## 限制

您可能还会遇到极少数情况下根本不支持热重载的情况。这些包括：

- 枚举类型更改为常规类或常规类更改为枚举类型。例如，如果您更改：

```
enum Color {  
  red,  
  green,
```

```
    blue  
}
```

改为:

```
class Color {  
    Color(this.i, this.j);  
    final Int i;  
    final Int j;  
}
```

- 泛型类型声明被修改。例如，如果您更改:

```
class A<T> {  
    T i;  
}
```

改为:

```
class A<T, V> {  
    T i;  
    V v;  
}
```

在这些情况下，热重载会生成诊断消息，并会在未提交任何更改的情况下失败。