

测试 Flutter App

- [介绍](#)
- [单元测试](#)
- [Widget 测试](#)
- [集成测试](#)
 - [添加flutter_driver依赖项](#)
 - [创建指令化的Flutter应用程序](#)
 - [编写集成测试](#)
 - [运行集成测试](#)

介绍

应用的功能越多，手动测试的难度就越大。一套完整的自动化测试将帮助您确保您的应用在发布之前正确执行，同时保留您的功能和错误修复速度。

有很多种自动化测试。这些总结如下：

- **单元测试**：测试单一功能、方法或类。例如，被测单元的外部依赖性通常被模拟出来，如[package:mockito](#)。单元测试通常不会读取/写入磁盘、渲染到屏幕，也不会从运行测试的进程外部接收用户操作。单元测试的目标是在各种条件下验证逻辑单元的正确性。
- **widget 测试**：(在其它UI框架称为 *组件测试*) 测试的单个widget。测试widget涉及多个类，并且需要提供适当的widget生命周期上下文的测试环境。例如，它应该能够接收和响应用户操作和事件，执行布局并实例化子widget。widget测试因此比单元测试更全面。然而，就像一个单元测试一样，一个widget测试的环境被一个比完整的UI系统简单得多的实现所取代。小部件测试的目标是验证小部件的UI如预期的那样的外观和交互。
- **集成测试**：测试一个完整的应用程序或应用程序的很大一部分。通常，集成测试可以在真实设备或OS仿真器上运行，例如iOS Simulator或Android Emulator。被测试的应用程序通常与测试驱动程序代码隔离，以避免结果偏差。集成测试的目标是验证应用程序作为一个整体正确运行，它所组成的所有widget如预期的那样相互集成。您还可以使用集成测试来验证应用的性能。

这里是一个表格，总结了在不同类型测试之间进行选择的权衡：

	单元测试	widget测试	集成测试
Confidence	Low	Higher	Highest
维护成本	Low	Higher	Highest
依赖	Few	More	Lots
执行速度	Quick	Slower	Slowest

提示： 作为一个经验法则，经过充分测试的应用程序具有非常多的单元和widget测试，通过代码覆盖([code coverage](#))进行跟踪，以及覆盖所有重要使用场景的大量集成测试。

单元测试

某些Flutter库，如`dart:ui`在独立的Dart VM附带的Dart SDK的中是不可用。该`flutter test`命令允许您在本地Dart VM中运行测试，使用无头版(不会显示UI)的Flutter引擎。使用这个命令你可以运行任何测试，不管它是否依赖于Flutter的库。

使用`package:test`，编写一个Flutter单元测试。编写单元测试使用的`package:test`文档在[这里](#)。

例如：

将此文件添加到 `test/unit_test.dart`：

```
import 'package:test/test.dart';

void main() {
  test('my first unit test', () {
    var answer = 42;
    expect(answer, 42);
  });
}
```

另外，您必须将以下内容添加到您的`pubspec.yaml`：

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

即使你的测试本身没有明确导入`flutter_test`，也需要这样做，因为测试框架本身在后台也使用了它。

要运行测试，从您的项目目录（而不是从`test`子目录）运行 `flutter test test/unit_test.dart`

要运行所有测试，请从项目目录运行`flutter test`

Widget 测试

您以类似于单元测试的方式实现widget测试。要在测试中执行与widget的交互，请使用Flutter提供的[WidgetTester](#)。例如，您可以发送点击和滚动手势。您还可以使用[WidgetTester](#)在widget树中查找子widget、读取文本、验证widget属性的值是否正确。

例子:

将此文件添加到`test/widget_test.dart`：

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

void main() {
  testWidgets('my first widget test', (WidgetTester tester) async {
    // You can use keys to locate the widget you need to test
    var sliderKey = new UniqueKey();
    var value = 0.0;

    // Tells the tester to build a UI based on the widget tree passed to it
    await tester.pumpWidget(
      new StatefulBuilder(
        builder: (BuildContext context, StateSetter setState) {
          return new MaterialApp(
            home: new Material(
              child: new Center(
                child: new Slider(
                  key: sliderKey,
                  value: value,
                  onChanged: (double newValue) {
                    setState(() {
                      value = newValue;
                    });
                  },
                ),
              ),
            ),
          ),
        ),
      );

    expect(value, equals(0.0));

    // Taps on the widget found by key
    await tester.tap(find.byKey(sliderKey));
```

```
// Verifies that the widget updated the value correctly
expect(value, equals(0.5));
});
}
```

运行 `flutter test test/widget_test.dart`.

查看所有可用于widget测试的[package:flutter_test API](#)

为了帮助调试widget测试，您可以使用[debugDumpApp\(\)](#) 函数来可视化测试的UI状态，或者只是简单的在您的首选运行时环境（例如模拟器或设备）中运行`flutter run test/widget_test.dart`以查看您的测试运行。在运行`flutter run`的测试的会话期间，您还可以交互式地点击Flutter工具的部分屏幕来打印建议的[Finder](#)。

集成测试

如果您熟悉Selenium/WebDriver (web)，Espresso(Android)或UI Automation(iOS)，那么Flutter Driver就是Flutter与这些集成测试工具的等价物。此外，Flutter Driver还提供API以记录测试执行的操作的性能跟踪（又名时间轴）。

Flutter的Driver是：

- 一个命令行工具 `flutter drive`
- 一个包 `package:flutter_driver` ([API](#))

这两者允许你：

- 为集成测试创建指令化的应用程序
- 写一个测试
- 运行测试

添加flutter_driver依赖项

要使用`flutter_driver`，您必须将以下块添加到您的`pubspec.yaml`：

```
dev_dependencies:
  flutter_driver:
    sdk: flutter
```

创建指令化的Flutter应用程序

一个指令化的应用程序是一个Flutter应用程序，它启用了Flutter Driver 扩展。启用扩展请调用[enableFlutterDriverExtension\(\)](#)。

例：

假设你有一个入口点的应用程序`my_app/lib/main.dart`。要创建它的指令化版本，请在`my_app/test_driver/`下创建一个Dart文件。在您正在测试的功能之后命名它；接下来定位

到my_app/test_driver/user_list_scrolling.dart：

```
// 这一行导入扩展
```

```
import 'package:flutter_driver/driver_extension.dart';
```

```
void main() {
```

```
  // 启用扩展
```

```
  enableFlutterDriverExtension();
```

```
  // Call the `main()` of your app or call `runApp` with whatever widget
```

```
  // you are interested in testing.
```

```
}
```

编写集成测试

集成测试是一个简单的package:test测试，它使用Flutter Driver API告诉应用程序执行什么操作，然后验证应用程序是否执行了此操作。

例子:

为了有意思起见，我们也让我们的测试记录下性能跟踪(performance timeline)。我们创建一个user_list_scrolling_test.dart测试文件位于my_app/test_driver/下：

```
import 'dart:async';
```

```
// Imports the Flutter Driver API
```

```
import 'package:flutter_driver/flutter_driver.dart';
```

```
import 'package:test/test.dart';
```

```
void main() {
```

```
  group('scrolling performance test', () {
```

```
    FlutterDriver driver;
```

```
    setUpAll(() async {
```

```
      // 连接app
```

```
      driver = await FlutterDriver.connect();
```

```
    });
```

```
    tearDownAll(() async {
```

```
      if (driver != null) {
```

```
        // 关闭连接
```

```
        driver.close();
```

```
      }
```

```
    });
```

```
    test('measure', () async {
```

```
      // 记录闭包中的performance timeline
```

```
      Timeline timeline = await driver.traceAction(() async {
```

```

        // Find the scrollable user list
        SerializableFinder userList = find.byValueKey('user-list');

        // Scroll down 5 times
        for (int i = 0; i < 5; i++) {
            // Scroll 300 pixels down, for 300 millis
            await driver.scroll(
                userList, 0.0, -300.0, new Duration(milliseconds: 300));

            // Emulate a user's finger taking its time to go back to the original
            // position before the next scroll
            await new Future<Null>.delayed(new Duration(milliseconds: 500));
        }

        // Scroll up 5 times
        for (int i = 0; i < 5; i++) {
            await driver.scroll(
                userList, 0.0, 300.0, new Duration(milliseconds: 300));
            await new Future<Null>.delayed(new Duration(milliseconds: 500));
        }
    });

    // The `timeline` object contains all the performance data recorded during
    // the scrolling session. It can be digested into a handful of useful
    // aggregate numbers, such as "average frame build time".
    TimelineSummary summary = new TimelineSummary.summarize(timeline);
    summary.writeSummaryToFile('stocks_scroll_perf', pretty: true);
    summary.writeTimelineToFile('stocks_scroll_perf', pretty: true);
});
});
}

```

运行集成测试

要在Android设备上运行测试，请通过USB将设备连接到计算机并启用USB调试。然后运行以下命令：

```
flutter drive --target=my_app/test_driver/user_list_scrolling.dart
```

该命令将：

- 构建 `--target` 应用，并将其安装在设备上
- 启动应用
- 运行 `my_app/test_driver/` 下的 `user_list_scrolling_test.dart`

您可能想知道该命令如何找到正确的测试文件。`flutter drive` 命令使用一种约定来查找与`--target`应用程序在同一目录中具有相同文件名但是具有`_test`后缀的测试文件。