Flutter测试

原创 coderwhy coderwhy

一. 单元测试

> 单元测试是针对一个函数或者类进行测试

1.1. 添加测试依赖

将 `test` 或者 `flutter_test` 加入依赖文件，默认创建的Flutter程序已经有了依赖：

- Test 包提供了编写测试所需要的核心功能

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

1.2. 创建需要测试的类

单元测试通常是测试一个函数或者类，这个函数或者类被称之为是一个 **单元** 。

在这里，我们按照官方示例，创建一个简单的Counter类来演示：

```
class Counter {
  int value = 0;

  void increment() => value++;
  void decrement() => value--;
}
```

1.3. 创建测试文件

我们在test目录下（注意：不是lib目录下），创建一个测试文件：counter_test.dart

- 通常测试代码都会放在该目录下，并且测试文件不会打包到最终的应用程序中；
- 测试文件通常以 `_test.dart` 命名，这是 test runner 寻找测试文件的惯例；

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:test_demo/counter.dart';

void main() {
  test("Counter Class test", () {
    // 1.创建Counter并且执行操作
    final counter = Counter();
    counter.increment();
    // 2.通过expect来监测结果正确与否
    expect(counter.value, 1);
  });
}
```

1.4. 整合多个测试

如果对同一个类或函数有多个测试，我们希望它们关联在一起进行测试，可以使用group

```dart
import 'dart:math';

import 'package:flutter_test/flutter_test.dart';
import 'package:test_demo/counter.dart';

void main() {
  group("Counter Test", () {
    test("Counter Default Value", () {
      expect(Counter().value, 0);
    });

    test("Counter Increment test", () {
      final counter = Counter();
      counter.increment();
      expect(counter.value, 1);
    });

    test("Counter Decrement test", () {
      final counter = Counter();
      counter.decrement();
      expect(counter.value, -1);
    });
  });
}
```

1.5. 执行测试结果

用 IntelliJ 或 VSCode 执行测试

IntelliJ 和 VSCode 的 Flutter 插件支持执行测试。用这种方式执行测试是最好的，因为它可以提供最快的反馈闭环，而且还支持断点调试。

- **IntelliJ**
  1. 打开文件 `counter_test.dart`
  2. 选择菜单 `Run`
  3. 点击选项 `Run 'tests in counter_test.dart'`
  4. *或者，也可以使用系统快捷键*
- **VSCode**
  1. 打开文件 `counter_test.dart`
  2. 选择菜单 `Debug`
  3. 点击选项 `Start Debugging`
  4. *或者，也可以使用系统快捷键*

在终端执行测试

我们也可以打开终端，在工程根目录输入以下命令来执行测试：

```
flutter test test/counter_test.dart
```

二. Widget测试

> Widget测试主要是针对某一个封装的Widget进行单独测试

1.1. 添加测试依赖

Widget测试需要先给 `pubspec.yaml` 文件的 `dev_dependencies` 段添加 `flutter_test` 依赖。

- 在单元测试中我们已经说过，默认创建的Flutter项目已经添加了

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

1.2. 创建测试Widget

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

class HYKeywords extends StatelessWidget {
  final List<String> keywords;
  HYKeywords(this.keywords);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ListView(
        children: keywords.map((key) {
          return ListTile(
            leading: Icon(Icons.people),
            title: Text(key),
          );
        }).toList(),
      ),
    );
  }
}
```

1.3. 编写测试代码

创建对应的测试文件，编写对应的测试代码：

- testWidgets：flutter_test中用于测试Widget的函数；
- tester.pumpWidget： `pumpWidget` 方法会建立并渲染我们提供的 widget；
- find： `find()` 方法来创建我们的 `Finders` ；
  - findsNothing：验证没有可被查找的 widgets。
  - findsWidgets：验证一个或多个 widgets 被找到。
  - findsNWidgets：验证特定数量的 widgets 被找到。

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

void main() {
  testWidgets("KeywordWidget Test", (WidgetTester tester) async {
    await tester.pumpWidget(MaterialApp(title: "demo", home: HYKeywords(["abc", "cba", "nba"]),));

    final abcText = find.text("abc");
    final cbaText = find.text("cba");
    final icons = find.byIcon(Icons.people);

    expect(abcText, findsOneWidget);
    expect(cbaText, findsOneWidget);
    expect(icons, findsNWidgets(2));
  });
}
```

官方文档中还有更多关于Widget的测试：

- https://flutter.dev/docs/cookbook/testing/widget/tap-drag

三. 集成测试

> 单元测试和Widget测试都是在测试独立的类或函数或Widget，它们并不能测试单独的模块形成的整体或者获取真实设备或模拟器上应用运行的状态；
>
> 这些测试任务可以交给 `集成测试` 来完成；
>
> 集成测试需要有两个大的步骤
>
> - 发布一个课测试应用程序到真实设备或者模拟器上；
> - 利用独立的测试套件去驱动应用程序，检查仪器是否完好可用；

3.1. 创建可测试应用程序

我们需要创建一个可以运行在模拟器或者真实设备的应用程序。

这里我直接使用了官方的示例程序，但是不同的是我这里给两个Widget添加了两个Key

- 显示数字的Text Widget：ValueKey("counter")
- 点击按钮的FloatingActionButton Widget：key: ValueKey("increment")

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              key: ValueKey("counter"),
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        key: ValueKey("increment"),
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

3.2. 添加flutter_driver依赖

我们需要用到 `flutter_driver` 包来编写 集成测试，所以我们需要把 `flutter_driver` 依赖添加到应用 `pubspec.yaml` 文件的 `dev_dependencies` 位置：

```
dev_dependencies:
  flutter_driver:
    sdk: flutter
  flutter_test:
    sdk: flutter
  test: any
```

3.3. 创建测试文件

和单元测试以及Widget测试不同的是，集成测试的程序和待测试的应用并不在同一个进程内，所以我们通常会创建两个文件：

- 文件一：用于启动带测试的应用程序
- 文件二：编写测试的代码

我们可以将这两个文件放到一个文件中：test_driver

```
  lib/
    main.dart
  test_driver/
    app.dart
    app_test.dart
```

3.4. 编写安装应用代码

安装应用程序代码在app.dart中，分层两步完成：

- 让 flutter driver 的扩展可用
- 运行应用程序

`test_driver/app.dart` 文件中增加以下代码：

```
import 'package:flutter_driver/driver_extension.dart';
import 'package:test_demo/main.dart' as app;

void main() {
  // 开启DriverExtension
  enableFlutterDriverExtension();

  // 手动调用main函数，启动应用程序
  app.main();
}
```

3.5. 编写集成测试代码

现在我们有了待测应用，我们可以为它编写测试文件了。这包含了四个步骤：

- 创建 `SerializableFinders` 定位指定组件
- 在 `setUpAll()` 函数中运行测试案例前，先与待测应用建立连接
- 测试重要场景
- 完成测试后，在 `teardownAll()` 函数中与待测应用断开连接

`test_driver/app_test.dart` 文件中增加以下代码：

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

void main() {
  group("Counter App Test", () {
    FlutterDriver driver;

    // 初始化操作
    setUpAll(() async {
      driver = await FlutterDriver.connect();
    });

    // 测试结束操作
    tearDownAll(() {
      if (driver != null) {
        driver.close();
      }
    });

    // 编写测试代码
    final counterTextFinder = find.byValueKey('counter');
    final buttonFinder = find.byValueKey('increment');

    test("starts at 0", () async {
      expect(await driver.getText(counterTextFinder), "0");
    });

    test("on tap click", () async {
      await driver.tap(buttonFinder);

      expect(await driver.getText(counterTextFinder), "1");
    });
  });
}
```

3.6. 运行集成测试

首先，启动安卓模拟器或者 iOS 模拟器，或者直接把 iOS 或 Android 真机连接到你的电脑上。

接着，在项目的根文件夹下运行下面的命令：

```
flutter drive --target=test_driver/app.dart
```

这个指令的作用：

1. 创建 `--target` 目标应用并且把它安装在模拟器或真机中
2. 启动应用程序
3. 运行位于 `test_driver/` 文件夹下的 `app_test.dart` 测试套件

运行结果：我们会发现正常运行，并且结果app中的FloatingActionButton自动被点击了一次。

> 备注：所有内容首发于公众号，之后除了Flutter也会更新其他技术文章，TypeScript、React、Node、uniapp、mpvue、数据结构与算法等等，也会更新一些自己的学习心得等，欢迎大家关注

coderwhy

微信扫描二维码，关注我的公众号

coderwhy

公众号