There are many ways to write a render speed test for Flutter. In this article, we give one example that uses e2e (or Flutter driver), the dev/benchmarks/macrobenchmarks app, and the dev/devicelab to automatically collect metrics for every future Flutter commit and send them to flutter/cocoon.

The instructions below are for contributors who want to expose a Flutter SDK (framework or engine) performance issue, or write pull requests to fix such issues. If one only needs to test the performance of a particular Flutter app, please reference

- https://flutter.dev/docs/cookbook/testing/integration/introduction.
- https://flutter.dev/docs/perf/rendering

Since Flutter Web and Flutter Desktop are still in their early stages, the content here is only well tested and supported on mobile platforms (Android/iOS). We'll come up with docs on how to write performance tests for Web/Desktop later.

Throughout this doc, we assume that the render speed test is for some `super_important_case`.

1. Add a page to macrobenchmarks

The macrobenchmarks is a Flutter app that includes many pages each of which corresponds to a specific performance test scenario. It provides some boilerplate code and auto-generated files so when a new scenario needs to be tested, one only needs to add a single page and a handful of files to the Flutter repo instead of adding a new Flutter app with dozens of auto-generated files. (The "macro" means that it's benchmarking a big system, including the whole Flutter framework and engine, instead of just a micro Dart or C++ function.)

To add a new test scenario `super_important_case`, do the following:

1. Create a `super_important_case.dart` inside macrobenchmarks/lib/src to define a `SuperImportantCasePage extends StatelessWidget {...}`. If there's a minimal Flutter app with a single `main.dart` file that reproduces the performance issue in the `super_important_case`, we'd often copy the content of that `main.dart` to `super_important_case.dart`.

2. Add a `const String kSuperImportantCaseRouteName = '/super_important_case'` to macrobenchmarks/lib/common.dart for later use.

3. Open macrobenchmarks/lib/main.dart and add the `kSuperImportantCaseRouteName: (BuildContext conttext) => SuperImportantCasePage(),` to the routes of `MacrobenchmarksApp`.

4. Scroll down to `HomePage`'s `ListView` and add the following `RaisedButton` so manual testers and the Flutter driver can tap it to navigate to the `super_important_case`.

```
RaisedButton(
  key: const Key(kSuperImportantCaseRouteName),
  child: const Text('Super Important Case'),
  onPressed: () {
    Navigator.pushNamed(context, kSuperImportantCaseRouteName);
  },
),
```

2. Add an e2e test

When the `super_important_case` page above is finished and manually tested, one can then add an automated integration test to get some performance metrics as follows.

1. We use macrobenchmarks/test_driver/e2e_test.dart as the host side script. All other tests depends on this file, so discuss with other Flutter members first if you want to change it.

2. Add `super_important_case_e2e.dart` to macrobenchmarks/test with the following content. The `macroPerfTestE2E` function will navigate the macrobenchmarks app to the `super_important_case` page, and starts collecting performance metrics. The optional arguments are:

    - The `pageDelay` is the time delay for loading the page. By default it doesn't wait.
    - The `duration` is the performance metric sampling time.
    - The `timeout` specifies the backstop timeout implemented by the test package, See testWidgets.
    - The `body` provides custom ways of driving that page during the benchmark such as scrolling through lists. When this is used together with `duration`, the test will perform for which ever last longer.
    - The `setup` provides the operation needed to setup before benchmark starts.

```dart
// Copyright 2014 The Flutter Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

import 'package:flutter/gestures.dart';
import 'package:flutter/widgets.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:macrobenchmarks/common.dart';

import 'util.dart';

void main() {
  macroPerfTestE2E(
    'super_important_case',
    kSuperImportantCaseRouteName,
    /* optional */ pageDelay: const Duration(seconds: 1),
    /* optional */ duration: const Duration(seconds: 3),
    /* optional */ timeout: const Duration(seconds: 30),
    /* optional */ body: (WidgetController controller) async {
      ...
    },
    /* optional */ setup: (WidgetController controller) async {
      ...
    },
  );
}
```

Once all steps above are done, one should be able to run `flutter drive -t test/super_important_case_perf.dart --driver test_driver/e2e_test.dart` inside the macrobenchmarks directory. After the driver test finished, the metrics should be written into a json file named `e2e_perf_summary.json` inside a temporary `build` directory under the current macrobenchmarks directory.

Some useful metrics in that json file include

- `average_frame_build_time_millis`
- `average_frame_rasterization_time_millis`
- `worst_frame_build_time_millis`
- `worst_frame_rasterization_time_millis`

2a. Add a driver test (deprecated)

(Skip this if step 2 is sufficient for you.)

When the `super_important_case` page above is finished and manually tested, one can then add an automatic driver test to get some performance metrics as follows.

1. We use macrobenchmarks/test_driver/run_app.dart as the device side app. All other tests depends on this file, so discuss with other Flutter members first if you want to change it.

2. Add `super_important_case_perf_test.dart` to macrobenchmarks/test_driver with the following content. The `macroPerfTest` function will navigate the macrobenchmarks app to the `super_important_case` page, and starts collecting performance metrics. The `driverOps` provides custom ways of driving that page during the benchmark such as scrolling through lists. The `setupOps` provides the operation needed to setup before benchmark starts.

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

import 'util.dart';

void main() {
  macroPerfTest(
    'super_important_case',
    kSuperImportantCaseRouteName,
    pageDelay: const Duration(seconds: 1),
    /* optional */ driverOps: (FlutterDriver driver) async {
        ...
    },
    /* optional */ setupOps: (FlutterDriver driver) async {
        ...
    },
  );
}
```

Once all steps above are done, one should be able to run `flutter drive -t test_driver/run_app.dart --driver test_driver/super_important_case_perf.dart` inside the macrobenchmarks directory. After the driver test finished, the metrics should be written into a json file named `super_important_case_perf__timeline_summary.json` inside a temporary `build` directory under the current macrobenchmarks directory.

Some useful metrics in that json file include

- `average_frame_build_time_millis`

- `average_frame_rasterization_time_millis`
- `worst_frame_build_time_millis`
- `worst_frame_rasterization_time_millis`

3. Update README

Add the new test to the list in macrobenchmarks/README.md.

4. Add a task to devicelab

To keep Flutter performant, running a test locally once in a while and check the metrics manually is insufficient. The following steps let the devicelab run the test automatically for every Flutter commit so performance regressions or speedups for the `super_important_case` can be detected quickly.

1. Add `super_important_case_perf__e2e_summary` to dev/devicelab/manifest.yaml under `tasks`. Follow other tasks to properly set descriptions and choose agent such as `linux/android` (Moto G4) or `mac/ios` (iPhone 6s). Mark it `flaky: true` so that while we observe the test case behavior on devicelab, we don't block the build tree.

2. Add `super_important_case_perf__e2e_summary.dart` to dev/devicelab/bin/tasks with a content like

```
import 'dart:async';

import 'package:flutter_devicelab/tasks/perf_tests.dart';
import 'package:flutter_devicelab/framework/adb.dart';
import 'package:flutter_devicelab/framework/framework.dart';

Future<void> main() async {
  deviceOperatingSystem = DeviceOperatingSystem.android;  // or ios
  await task(createSuperImportantCasePerfE2ETest());
}
```

3. Add the following `createSuperImportantCasePerfTest` function to dev/devicelab/lib/tasks/perf_tests.dart

```
TaskFunction createSuperImportantCasePerfE2ETest() {
  return PerfTest.e2e(
    '${flutterDirectory.path}/dev/benchmarks/macrobenchmarks',
    'test/super_important_case_e2e.dart',
  ).run;
}
```

4. Locally test the devicelab task by running `../../bin/cache/dart-sdk/bin/dart bin/run.dart -t super_important_case_perf__e2e_summary` inside the dev/devicelab directory with an Android or iOS device connected. You should see a success and a summary of metrics being printed out.

5. Submit a pull request of everything above.

6. Finally, remove `flaky: true` once the test is proven to be reliable for a few days. Since this may take a while, creating a reminder calendar event could be a good idea.

4a. Add a task to devicelab for driver tests (deprecated)

(Skip this if you didn't do step 2a.)

To keep Flutter performant, running a test locally once in a while and check the metrics manually is insufficient. The following steps let the devicelab run the test automatically for every Flutter commit so performance regressions or speedups for the `super_important_case` can be detected quickly.

1. Add `super_important_case_perf__timeline_summary` to dev/devicelab/manifest.yaml under `tasks`. Follow other tasks to properly set descriptions and choose agent such as `linux/android` (Moto G4) or `mac/ios` (iPhone 6s).

2. Add `super_important_case_perf__timeline_summary.dart` to dev/devicelab/bin/tasks with a content like

```dart
import 'dart:async';

import 'package:flutter_devicelab/tasks/perf_tests.dart';
import 'package:flutter_devicelab/framework/adb.dart';
import 'package:flutter_devicelab/framework/framework.dart';

Future<void> main() async {
  deviceOperatingSystem = DeviceOperatingSystem.android;  // or ios
  await task(createSuperImportantCasePerfTest());
}
```

3. Add the following `createSuperImportantCasePerfTest` function to dev/devicelab/lib/tasks/perf_tests.dart

```dart
TaskFunction createSuperImportantCasePerfTest() {
  return PerfTest(
    '${flutterDirectory.path}/dev/benchmarks/macrobenchmarks',
    'test_driver/run_app.dart',
    'super_important_case_perf',
    testDriver: 'test_driver/super_important_case_perf_test.dart',
  ).run;
}
```

4. Locally test the devicelab task by running `../../bin/cache/dart-sdk/bin/dart bin/run.dart -t super_important_case_perf__timeline_summary` inside the dev/devicelab directory with an Android or iOS device connected. You should see a success and a summary of metrics being printed out.

5. Submit a pull request of everything above.

6. Finally, remove `flaky: true` once the test is proven to be reliable for a few days. Since this may take a while, creating a reminder calendar event could be a good idea.

5. Set benchmark baseline

Tasks will be run automatically in the devicelab, and the result is shown in flutter-dashboard. Set the baseline in flutter-dashboard once the new test gets enough data. Also for metrics like "vsync_transitions_missed", change the unit from default ms to frames or other suitable units.

Acknowledgement

Big congratulations if you've successfully finished all steps above! You just made a big contribution to Flutter's performance. Please also feel encouraged to improve this doc to help future contributors (which probably include a future yourself that would forget something above in a few months)!