DevTools

docs.flutter.dev

▼ *데비트님, DevTools 자주 사용 하시나요?*

👉 **성능 개선 목적으로 Frames 체크, 메모리 이슈 체크** 등 사용

👉 저는 위젯 트리, UI 디버깅을 위해 `inspector` 제외하고는 자주 사용해 본 적은 없습니다.
하지만 네트워크 확인용으로도 많이 쓰인다고 해요

▼ *그럼 왜 개발하시면서 자주 사용 안 하시나요?*

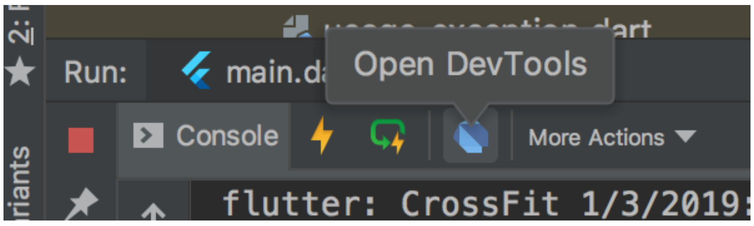
- **Hot Reload, Hot Restart**
- **Logging**
- inspector 위젯이 복잡한 화면에서는 느린 단점이 있음
- 프로젝트가 무거워질수록 devtools도 무거워짐
- UI 위젯 및 레이아웃이 익숙해짐

Overview.

What can I do with DevTools?

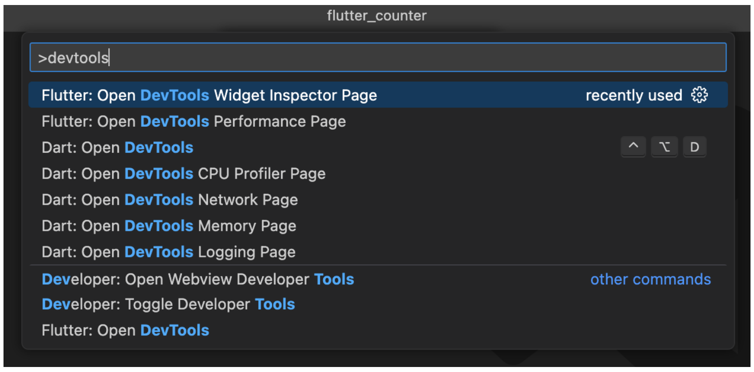
- **Inspect : UI layout, state**
- Diagnose UI jank performance
- CPU profiling
- **Network profiling**
- Source-level debugging
- **Debug memory issues**
- View general log and diagnostics information
- Analyze code and app size

DevTools from Android Studio & IntelliJ.



```
flutter: CrossFit 1/3/2019:
```

DevTools from VSCode.

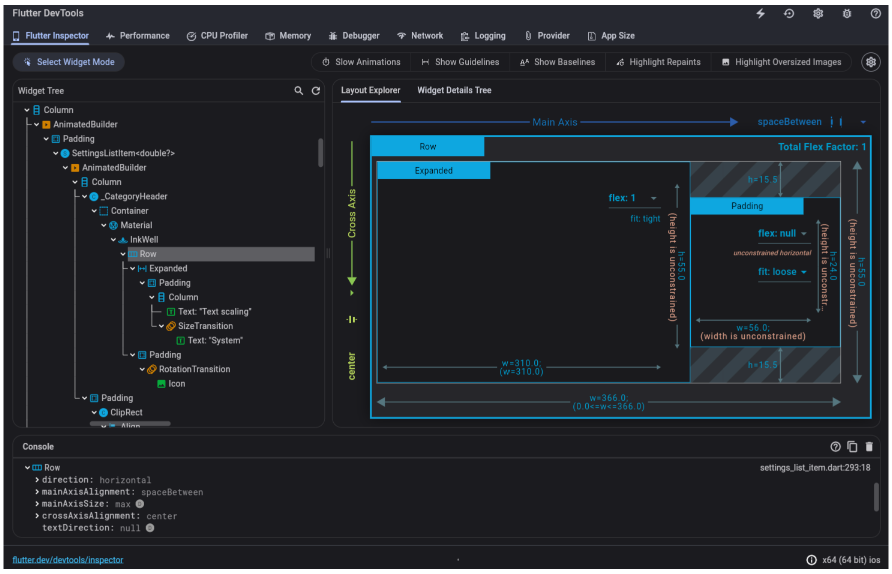


```
>devtools
Flutter: Open DevTools Widget Inspector Page          recently used
Flutter: Open DevTools Performance Page
Dart: Open DevTools                                   ^ ⌥ D
Dart: Open DevTools CPU Profiler Page
Dart: Open DevTools Network Page
Dart: Open DevTools Memory Page
Dart: Open DevTools Logging Page
Developer: Open Webview Developer Tools               other commands
Developer: Toggle Developer Tools
Flutter: Open DevTools
```

Flutter inspector.

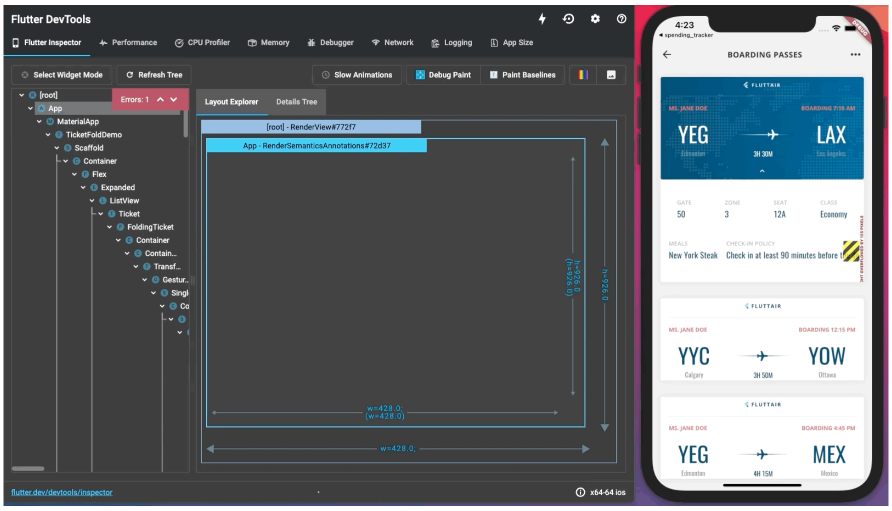
powerful tool for visualizing and exploring Flutter widget trees.

- understanding existing layouts
- diagnosing layout issues



```
Row
 › direction: horizontal
 › mainAxisAlignment: spaceBetween
 › mainAxisSize: max
 › crossAxisAlignment: center
   textDirection: null
```

Using the Layout Explorer.



Visual debugging.

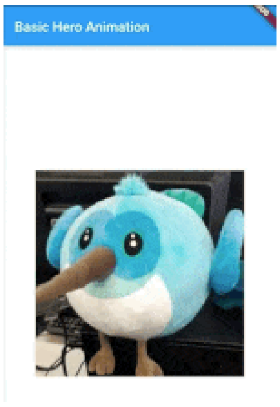
Select widget mode

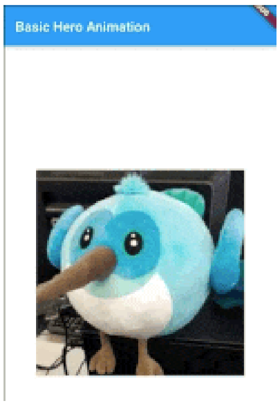
디바이스 위젯 클릭 시, 해당 위젯 Layout Explorer

Refresh tree

Slow animations

animations 5 times slower





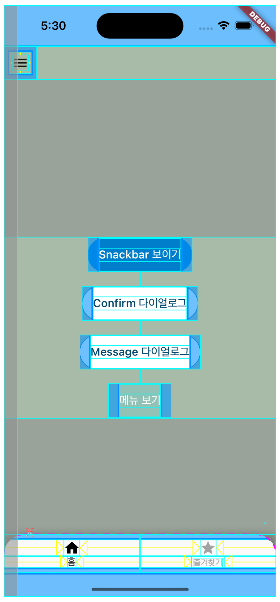
👉 **고도화된 애니메이션(인터랙션)** 구현할 경우, 용이

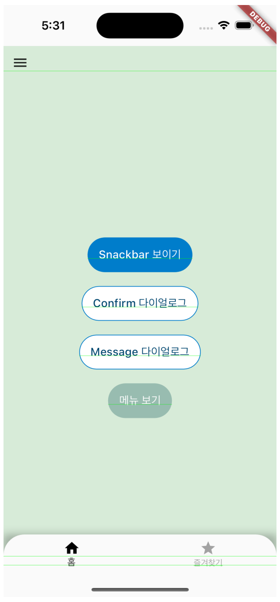
Show guidelines

render boxes, alignments, paddings, scroll views, clippings and spacers.

Show baselines

For aligning text.



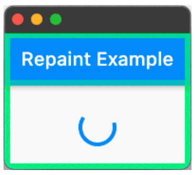


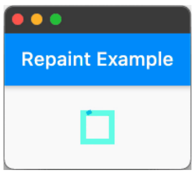
Highlight repaints

draws a border around all render boxes that changes color every time that box repaints.

Useful for finding unnecessary repaints.

`RepaintBoundary`





Highlight oversized images

```
dash.png has a display size of 213×392 but a decode size of 2130×392, which uses an additional 2542KB
```

Programmatically.

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

void setSlowAnimations() {
  timeDilation = 5.0;
}

import 'package:flutter/rendering.dart';

void showLayoutGuidelines() {
  debugPaintSizeEnabled = true;
}

import 'package:flutter/rendering.dart';

void showBaselines() {
  debugPaintBaselinesEnabled = true;
}

void showOversizedImages() {
  debugInvertOversizedImages = true;
}
```

👉 **개발자 도구 페이지 등에 활용 가능.**

Performance view.

profile build (Not debug/release mode)

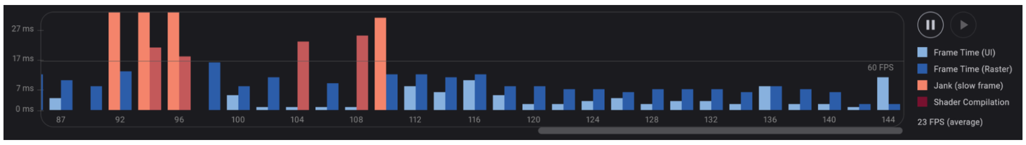
`--profile`

What is a frame in Flutter?

- **60 fps** : render its UI at 60
- **120 fps** : on devices capable of 120Hz updates

약 **16ms(1 / 60 * 1000)** 마다 UI 업데이트 진행함.
이것보다 이상 걸리면 **jank**

Flutter frames chart.



UI

UI thread executes **Dart code in the Dart VM**
sends the layer tree to the raster thread

*Do **not** block this thread.*

Raster (GPU thread)

This thread **takes the layer tree** and **displays** it by talking to the GPU

Graphics library(Skia / Impeller) runs on this thread

- You can't directly access…
- 구성하기 쉬운 layer tree 도 expensive to render

> They might involve unnecessary calls to `saveLayer()`,
> **intersecting opacities with multiple objects, and clips or shadows**

Jank (slow frame)

jank with a red overlay.

> if it takes more than ~16 ms to complete (for 60 FPS devices)

- UI Thread: 위젯트리가 자주 변경 될 경우 이슈 발생, 무거운 작업 실행
- Raster Thread: Opacity, Shadow, Clip

Shader compilation

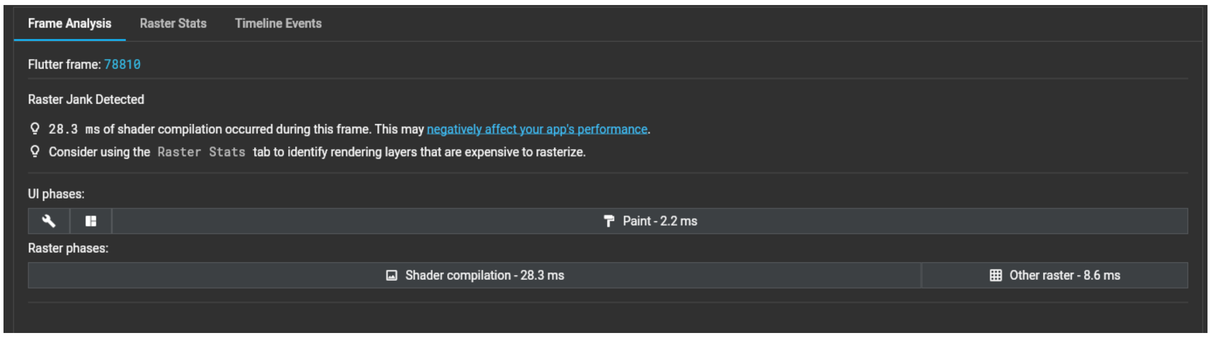
Shader compilation occurs when **a shader is first used** in app

> check out Reduce shader compilation jank on mobile.

특히, iOS 많았음. 👉 **impeller**

Frame analysis tab.

- Selecting a janky frame
- debugging hints



Frame Analysis | Raster Stats | Timeline Events

Flutter frame: 78810

Raster Jank Detected

`28.3 ms` of shader compilation occurred during this frame. This may negatively affect your app's performance.

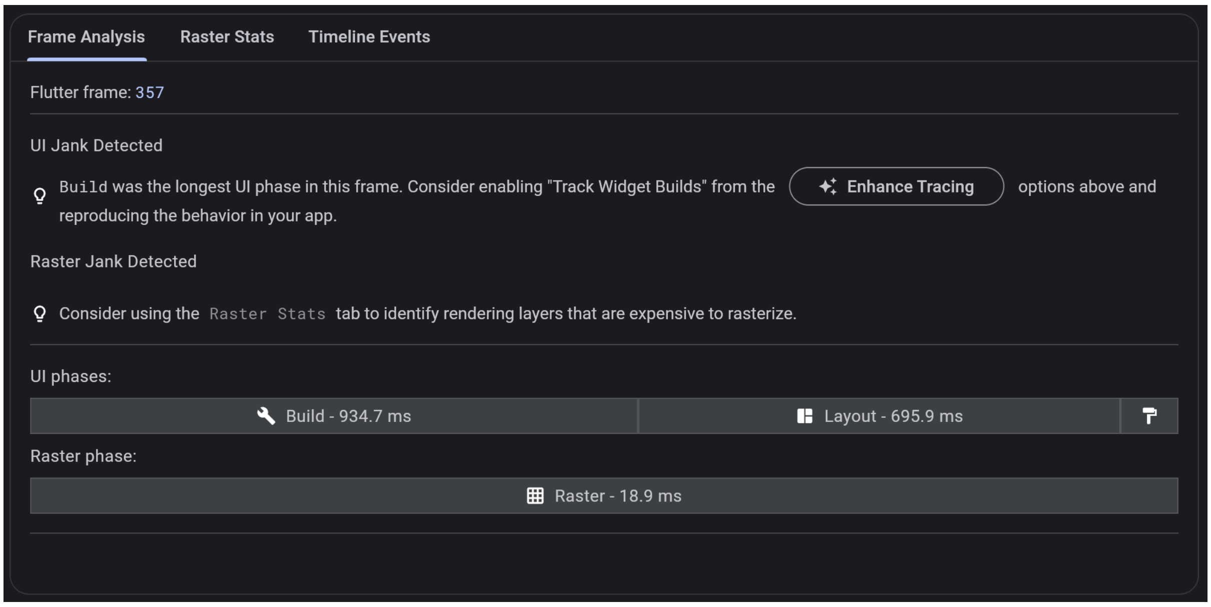
Consider using the `Raster Stats` tab to identify rendering layers that are expensive to rasterize.

UI phases:

Paint - 2.2 ms

Raster phases:

Shader compilation - 28.3 ms | Other raster - 8.6 ms



Frame Analysis | Raster Stats | Timeline Events

Flutter frame: 357

UI Jank Detected

`Build` was the longest UI phase in this frame. Consider enabling "Track Widget Builds" from the Enhance Tracing options above and reproducing the behavior in your app.

Raster Jank Detected

Consider using the `Raster Stats` tab to identify rendering layers that are expensive to rasterize.

UI phases:

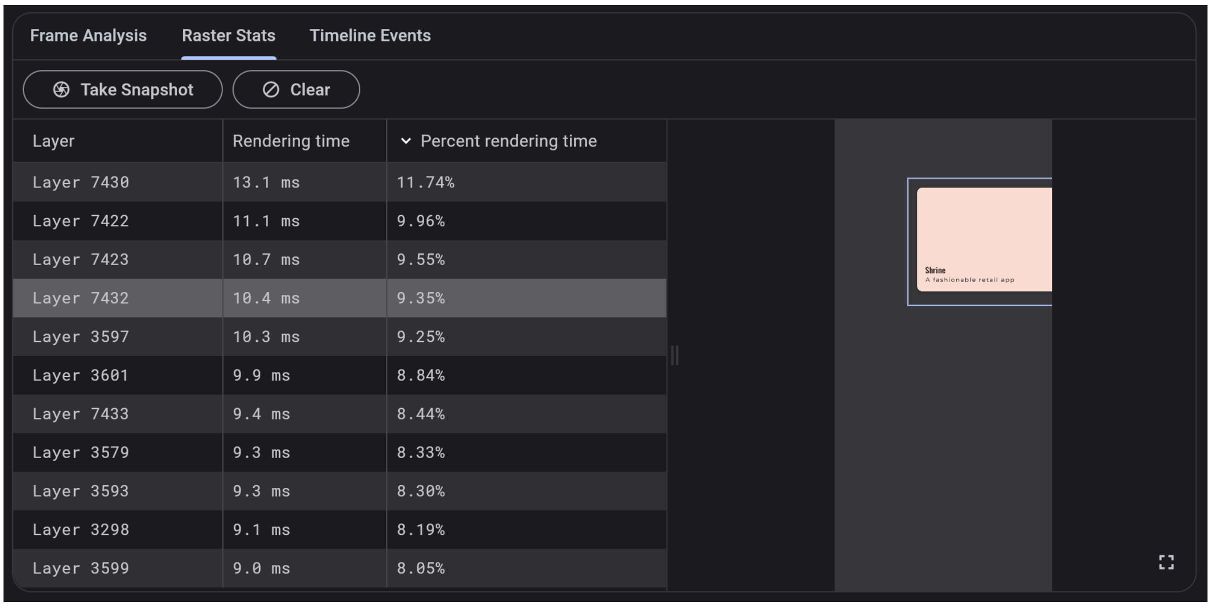
Build - 934.7 ms | Layout - 695.9 ms

Raster phase:

Raster - 18.9 ms

Raster stats tab.

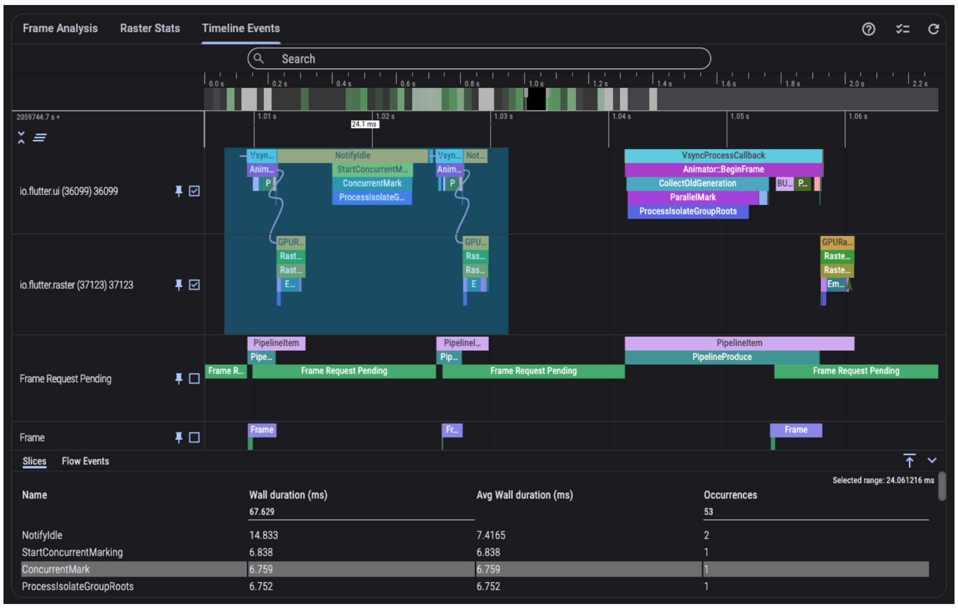
- raster thread jank 화면에서 **Take Snapshot**
- View different layers and their respective rendering times.



Layer	Rendering time	Percent rendering time
Layer 7430	13.1 ms	11.74%
Layer 7422	11.1 ms	9.96%
Layer 7423	10.7 ms	9.55%
Layer 7432	10.4 ms	9.35%
Layer 3597	10.3 ms	9.25%
Layer 3601	9.9 ms	8.84%
Layer 7433	9.4 ms	8.44%
Layer 3579	9.3 ms	8.33%
Layer 3593	9.3 ms	8.30%
Layer 3298	9.1 ms	8.19%
Layer 3599	9.0 ms	8.05%

Timeline events tab.

frames, draw scenes, and track other activity such as HTTP request timings and garbage collection.



Selected range: 24.061216 ms

Name	Wall duration (ms)	Avg Wall duration (ms)	Occurrences
	67.629		53
NotifyIdle	14.833	7.4165	2
StartConcurrentMarking	6.838	6.838	1
ConcurrentMark	6.759	6.759	1
ProcessIsolateGroupRoots	6.752	6.752	1

- **Track widget builds** `build()`
- **Track layouts**

- **Track paints**

성능 최적화.

- `ListView.builder` / `ListView(children: [...])`
- `RepaintBoundary`
- Opacity, Shadow, Clip 최적화

```
Image.network(
  'https://raw.githubusercontent.com/flutter/assets-for-api-docs/master/packages/diagrams/assets/blend_mode_destination.jpeg',
  color: const Color.fromRGBO(255, 255, 255, 0.5),
  colorBlendMode: BlendMode.modulate
)
```

- 무거운 작업은 병렬로 작업하기 위해 **isolate** 활용 (`compute` / `Isolate.run`)

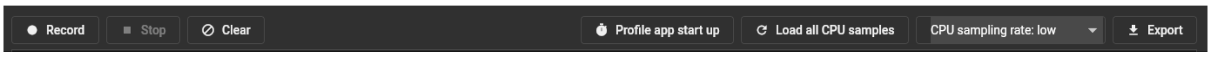
  > Single thread, 독립적인 실행 공간

CPU profiler.

solve performance problems or generally understand your app's **CPU activity.**

Dart VM collects CPU samples (a snapshot of the CPU call stack at a single point in time)
DevTools for visualization

- **record 하는 방식**



Bottom up.

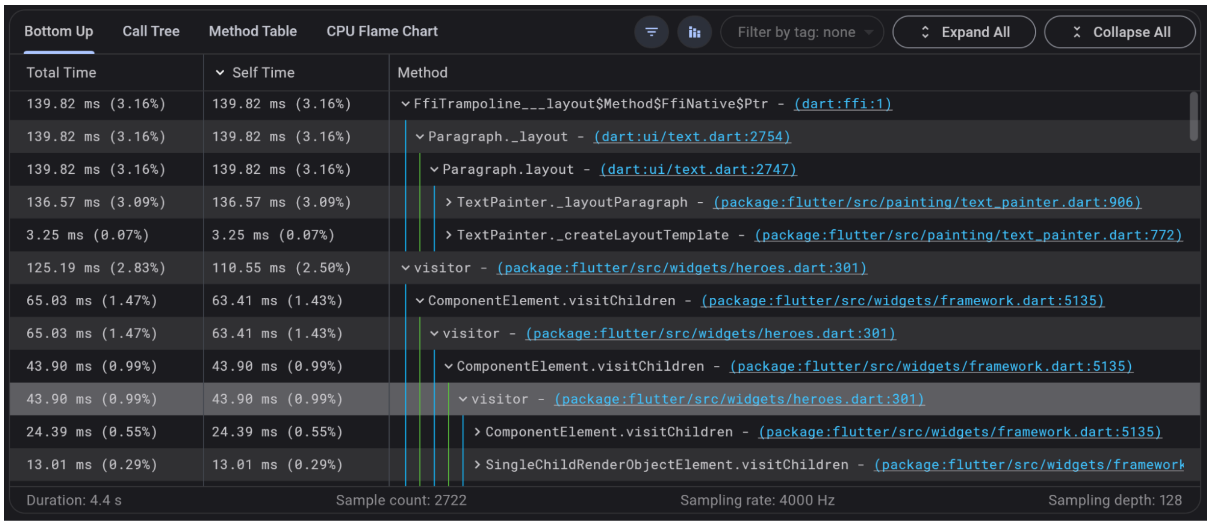
useful for identifying *expensive methods* in a CPU profile

Total time

- 자체 메서드 실행 시간 + 호출된 모든 메서드 포함

Self time

- 자체 메서드 실행 시간



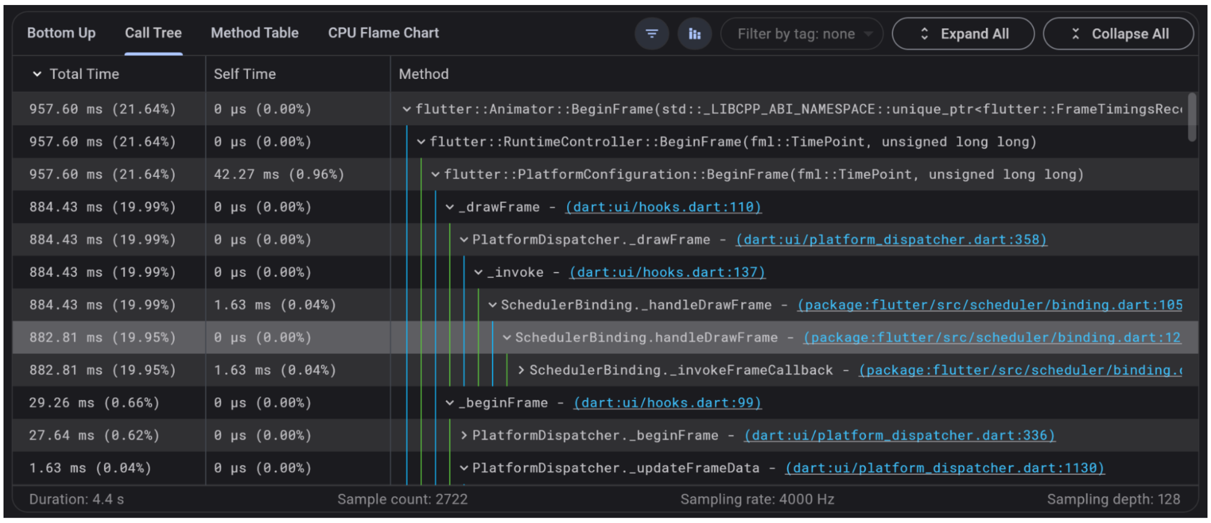
Total Time	Self Time	Method
139.82 ms (3.16%)	139.82 ms (3.16%)	FfiTrampoline___layout$Method$FfiNative$Ptr - (dart:ffi:1)
139.82 ms (3.16%)	139.82 ms (3.16%)	Paragraph._layout - (dart:ui/text.dart:2754)
139.82 ms (3.16%)	139.82 ms (3.16%)	Paragraph.layout - (dart:ui/text.dart:2747)
136.57 ms (3.09%)	136.57 ms (3.09%)	TextPainter._layoutParagraph - (package:flutter/src/painting/text_painter.dart:906)
3.25 ms (0.07%)	3.25 ms (0.07%)	TextPainter._createLayoutTemplate - (package:flutter/src/painting/text_painter.dart:772)
125.19 ms (2.83%)	110.55 ms (2.50%)	visitor - (package:flutter/src/widgets/heroes.dart:301)
65.03 ms (1.47%)	63.41 ms (1.43%)	ComponentElement.visitChildren - (package:flutter/src/widgets/framework.dart:5135)
65.03 ms (1.47%)	63.41 ms (1.43%)	visitor - (package:flutter/src/widgets/heroes.dart:301)
43.90 ms (0.99%)	43.90 ms (0.99%)	ComponentElement.visitChildren - (package:flutter/src/widgets/framework.dart:5135)
43.90 ms (0.99%)	43.90 ms (0.99%)	visitor - (package:flutter/src/widgets/heroes.dart:301)
24.39 ms (0.55%)	24.39 ms (0.55%)	ComponentElement.visitChildren - (package:flutter/src/widgets/framework.dart:5135)
13.01 ms (0.29%)	13.01 ms (0.29%)	SingleChildRenderObjectElement.visitChildren - (package:flutter/src/widgets/framework

Duration: 4.4 s　Sample count: 2722　Sampling rate: 4000 Hz　Sampling depth: 128

Call tree.

- top-down

useful for identifying **<u>expensive *paths*</u>** in a CPU profile.

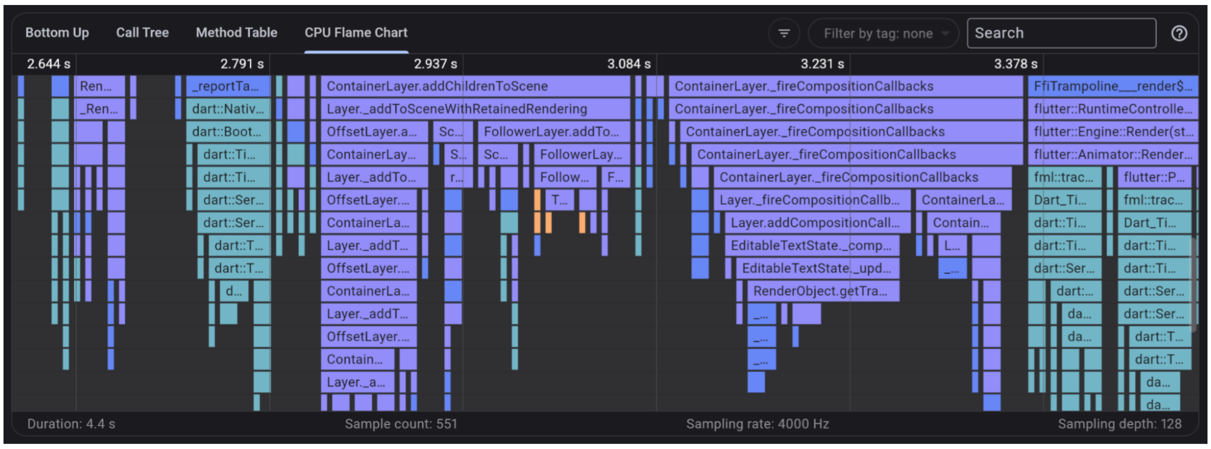


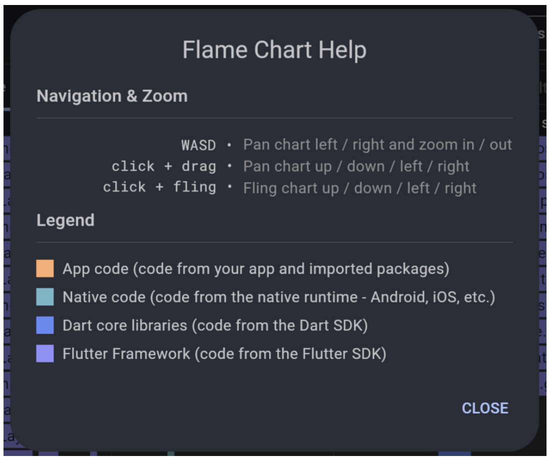
Total Time	Self Time	Method
957.60 ms (21.64%)	0 μs (0.00%)	flutter::Animator::BeginFrame(std::_LIBCPP_ABI_NAMESPACE::unique_ptr<flutter::FrameTimingsRec
957.60 ms (21.64%)	0 μs (0.00%)	flutter::RuntimeController::BeginFrame(fml::TimePoint, unsigned long long)
957.60 ms (21.64%)	42.27 ms (0.96%)	flutter::PlatformConfiguration::BeginFrame(fml::TimePoint, unsigned long long)
884.43 ms (19.99%)	0 μs (0.00%)	_drawFrame - (dart:ui/hooks.dart:110)
884.43 ms (19.99%)	0 μs (0.00%)	PlatformDispatcher._drawFrame - (dart:ui/platform_dispatcher.dart:358)
884.43 ms (19.99%)	0 μs (0.00%)	_invoke - (dart:ui/hooks.dart:137)
884.43 ms (19.99%)	1.63 ms (0.04%)	SchedulerBinding._handleDrawFrame - (package:flutter/src/scheduler/binding.dart:105
882.81 ms (19.95%)	0 μs (0.00%)	SchedulerBinding.handleDrawFrame - (package:flutter/src/scheduler/binding.dart:12
882.81 ms (19.95%)	1.63 ms (0.04%)	SchedulerBinding._invokeFrameCallback - (package:flutter/src/scheduler/binding.
29.26 ms (0.66%)	0 μs (0.00%)	_beginFrame - (dart:ui/hooks.dart:99)
27.64 ms (0.62%)	0 μs (0.00%)	PlatformDispatcher._beginFrame - (dart:ui/platform_dispatcher.dart:336)
1.63 ms (0.04%)	0 μs (0.00%)	PlatformDispatcher._updateFrameData - (dart:ui/platform_dispatcher.dart:1130)

Duration: 4.4 s　Sample count: 2722　Sampling rate: 4000 Hz　Sampling depth: 128

Flame chart.

- top-down (call tree 유사)
- 각 flame element width : the amount of time that a method spent on the call stack.


Bottom Up
Call Tree
Method Table
CPU Flame Chart
Filter by tag: none
Search
2.644 s
2.791 s
2.937 s
3.084 s
3.231 s
3.378 s
ContainerLayer.addChildrenToScene
Layer._addToSceneWithRetainedRendering
ContainerLayer._fireCompositionCallbacks
FfiTrampoline__render$...
flutter::RuntimeControlle...
flutter::Engine::Render(st...
flutter::Animator::Render...
RenderObject.getTra...
Duration: 4.4 s
Sample count: 551
Sampling rate: 4000 Hz
Sampling depth: 128


Flame Chart Help

Navigation & Zoom

```
         WASD  •  Pan chart left / right and zoom in / out
 click + drag  •  Pan chart up / down / left / right
click + fling  •  Fling chart up / down / left / right
```

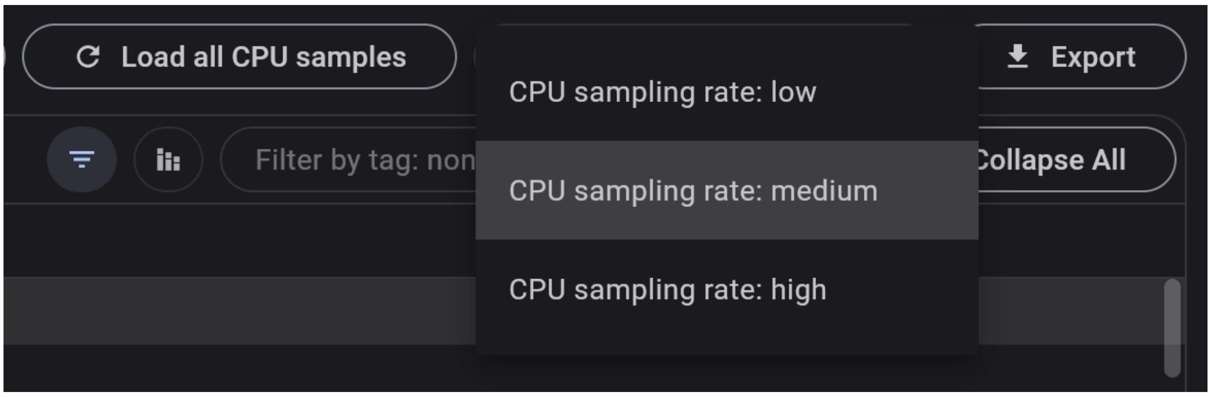
Legend

- 🟧 App code (code from your app and imported packages)
- 🟦 Native code (code from the native runtime - Android, iOS, etc.)
- 🟦 Dart core libraries (code from the Dart SDK)
- 🟪 Flutter Framework (code from the Flutter SDK)

CLOSE

CPU sampling rate.

CPU 샘플을 수집하는 기본 속도를 설정



CPU sampling rate: low

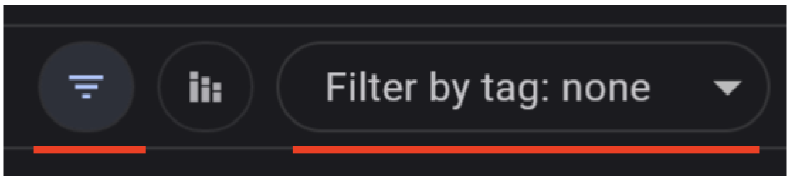
CPU sampling rate: medium

CPU sampling rate: high

- 디폴트 - CPU samples: 1 sample / 250 μs (**microseconds**).
    - 1초에 4000hz
- low: 1,000 Hz
- medium: 4,000 Hz
- high: 20,000 Hz

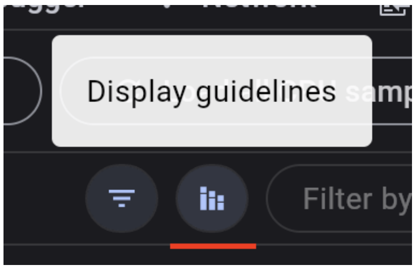
Filtering.

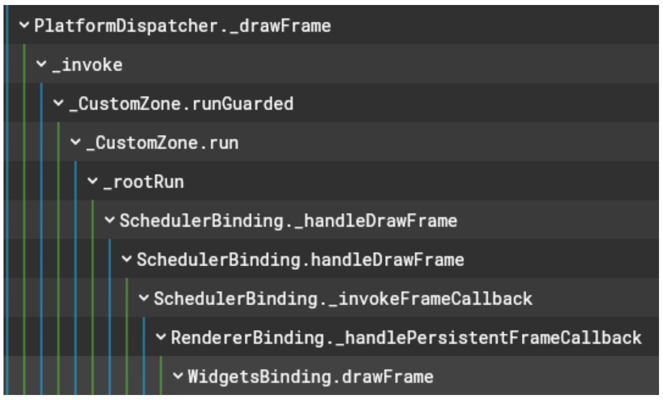
filter the data by library, method name, or `UserTag`.



Guidelines.

call tree, bottom up 볼 때 보기 편하도록 가이드라인 제공.





```
⌄PlatformDispatcher._drawFrame
  ⌄_invoke
    ⌄_CustomZone.runGuarded
      ⌄_CustomZone.run
        ⌄_rootRun
          ⌄SchedulerBinding._handleDrawFrame
            ⌄SchedulerBinding.handleDrawFrame
              ⌄SchedulerBinding._invokeFrameCallback
                ⌄RendererBinding._handlePersistentFrameCallback
                  ⌄WidgetsBinding.drawFrame
```

Memory view.

memory allocation and tools to detect and debug specific issues.

Reasons to use.

- out of memory ⇒ crash
- 느려짐
- 반응 없음
- 메모리 한계에 도달 할 경우, OS 에 의하여 종료됨.
- memory leak 체크

> 동적으로 할당한 뒤에 메모리 해제를 하지 않아서 계속 남아 있는 경우,
> 계속 쌓이면 결국 **out of memory** 발생

Basic memory concepts.

클래스 객체(object)를 생성하면, 메모리 Heap 영역에 할당됨.

object 가 더 이상 사용되지 않을 때, 메모리에 해지됨(Garbage Collector)

Root object.

Every Dart application creates **a *root object***
that **references**, all other objects the application allocates.

Reachability.

the root object **stops referencing** an allocated object, the object becomes ***unreachable***
which is a signal for the **garbage collector (GC) to deallocate** the object's memory.

Retaining path.

- The sequence of references from root
- 하나의 object 에서 여러 개의 Retaining path 가질 수 있음
- Retaining path 하나라도 있을 시, *reachable*

```
void main() {
  myFunction();
}

class Child{}

class Parent {
  Child? child;
}

Parent parent1 = Parent();

void myFunction() {

  Child? child = Child();
  print(child?.hashCode);

  // The `child` object was allocated in memory.
  // It's now retained from garbage collection
  // by one retaining path (root …-> myFunction -> child).

  Parent? parent2 = Parent()..child = child;
  parent1.child = child;

  // At this point the `child` object has three retaining paths:
  // root …-> myFunction -> child
  // root …-> myFunction -> parent2 -> child
  // root -> parent1 -> child

  child = null;
  parent1.child = null;
  parent2 = null;

  // At this point, the `child` instance is unreachable
  // and will eventually be garbage collected.

  //   print(parent2?.child?.hashCode);
}
```

Shallow size vs retained size.

- **Shallow size** - **object, references 포함**
- **retained size** - **size of the retained objects** 포함

The **retained size** of the **root object** includes all reachable Dart objects.

DevTools 계산에서는 만약 object 가 하나 이상의 retaining path 가질 시,
shortest retaining path 만 사이즈 포함

In this example the object `x` has two retaining paths:

```
root -> a -> b -> c -> x
root -> d -> e -> x (shortest retaining path to `x`)
```

Only members of the shortest path (`d` and `e`) will include `x` into their retaining size.

Memory leaks happen in Dart?

Garbage collector 모든 memory leaks 방지할 수 없음

그래서 개발자가 leak-free lifecycle 를 위해 object 를 감시해야함

Why can’t the garbage collector prevent all leaks?
필요하지 않은 object 들이 global 또는 static 변수로 있으면,
garbage collector 는 인식 할 수없어서 메모리에 남게됨.

Why closures require extra attention.

클로저 형태는 더욱 더 찾기 어려움.

a reference to the designed-to-be short-living `myHugeObject` is implicitly stored in the closure context and passed to `setHandler`

```
final handler = () => print(myHugeObject.name);
setHandler(handler);
```

Why `BuildContext` requires extra attention.

```
// BAD: DO NOT DO THIS
// This code is leak prone:
@override
Widget build(BuildContext context) {
  final handler = () => apply(Theme.of(context));
  useHandler(handler);
```

fix leak prone.

```
// GOOD
@override
Widget build(BuildContext context) {
  final theme = Theme.of(context);
  final handler = () => apply(theme);
  useHandler(handler);
…
```

General rule for `BuildContext`.

closure 가 위젯보다 오래 유지되지 않는다면, 내부에 context 전달해도 됨.

비슷한 예시로 Stateful widget 은 Widget 과 State 두 개의 클래스로 구성됨.
위젯은 short living, state 는 long living.

따라서, state 는 widget 의 context 를 참조해서는 안됨!

Memory leak vs memory bloat.

Memory bloat **uses more memory than is necessary** for optimal performance

by **using overly large images** or **keeping streams open** through their lifetime.

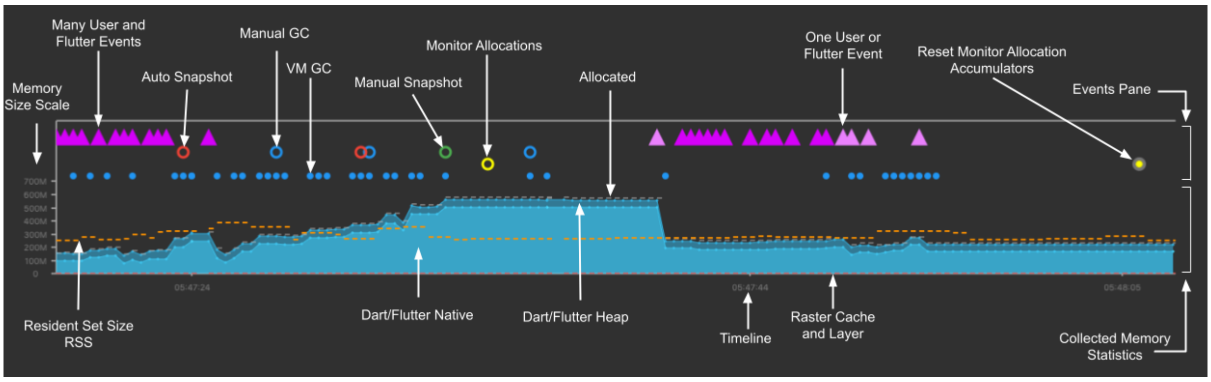
결국 둘 다 많아지면, `out-of-memory` 크래시 발생

Memory view guide.

investigate **memory allocations** (both in the heap and external), **memory leaks, memory bloat**, and more.

Expandable chart.

Memory anatomy



Dart/Flutter Heap

Objects (Dart and Flutter objects) in the heap.

Dart/Flutter Native

네이티브 관련 영역, 예를 들면 파일 읽기

Raster Cache

The size of the Flutter engine's raster cache layer(s) or picture(s)

Allocated

The current capacity of the heap is typically slightly larger than the total size of all heap objects.

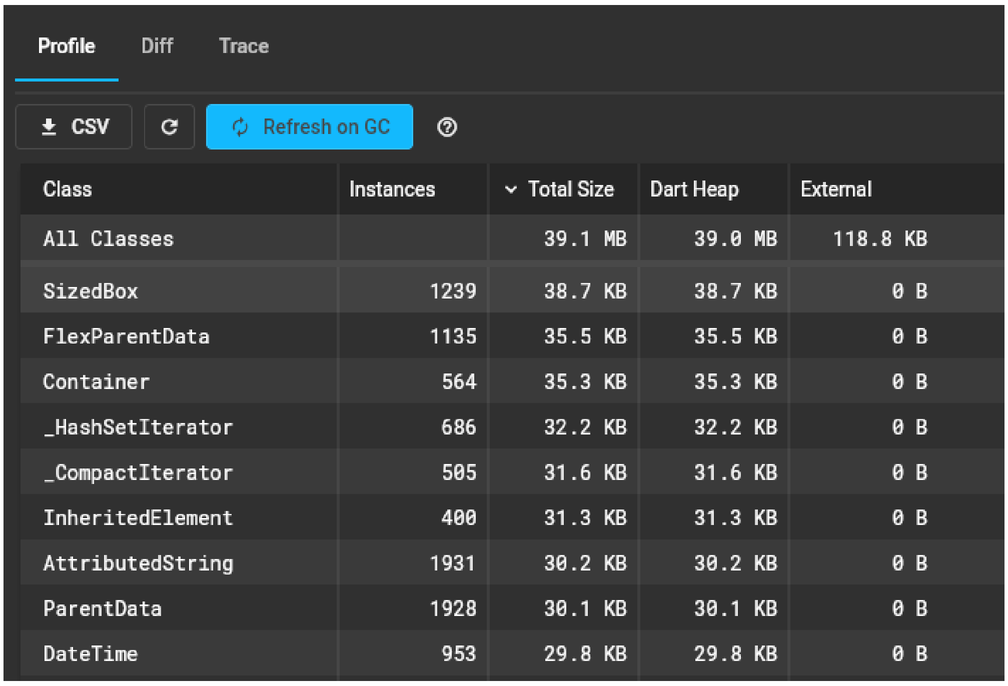
RSS - Resident Set Size

It includes memory from **shared libraries** that are loaded, as well as all stack and heap memory. Dart VM internals.

Profile Memory tab.

current memory allocation by **class and memory type**.

- CSV 다운로드
- Toggle **Refresh on GC**
- 현재 메모리 할당되어 있는 클래스를 찾기 편함

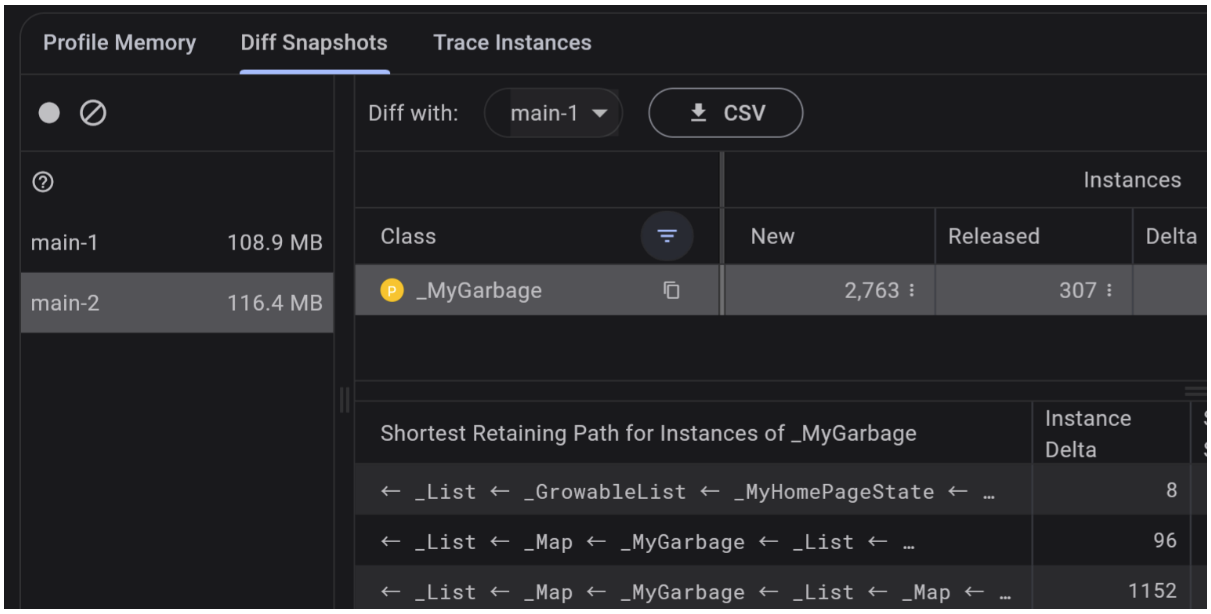


Class	Instances	Total Size	Dart Heap	External
All Classes		39.1 MB	39.0 MB	118.8 KB
SizedBox	1239	38.7 KB	38.7 KB	0 B
FlexParentData	1135	35.5 KB	35.5 KB	0 B
Container	564	35.3 KB	35.3 KB	0 B
_HashSetIterator	686	32.2 KB	32.2 KB	0 B
_CompactIterator	505	31.6 KB	31.6 KB	0 B
InheritedElement	400	31.3 KB	31.3 KB	0 B
AttributedString	1931	30.2 KB	30.2 KB	0 B
ParentData	1928	30.1 KB	30.1 KB	0 B
DateTime	953	29.8 KB	29.8 KB	0 B

Diff Snapshots tab.

snapshots 활용하여 비교 가능

- 가장 활용도 높아보임
- 화면 이동하면서 메모리 제대로 해지되었는지 체크 가능



Diff with: main-1

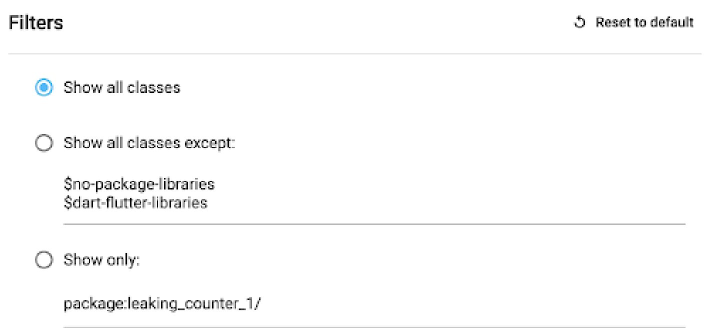
main-1	108.9 MB
main-2	116.4 MB

	Instances		
Class	New	Released	Delta
_MyGarbage	2,763	307	

Shortest Retaining Path for Instances of _MyGarbage	Instance Delta
← _List ← _GrowableList ← _MyHomePageState ← …	8
← _List ← _Map ← _MyGarbage ← _List ← …	96
← _List ← _Map ← _MyGarbage ← _List ← _Map ← …	1152

Filter classes.

Show only: 우리의 package


Filters

Reset to default

[✓ Show all classes]

Show all classes except:

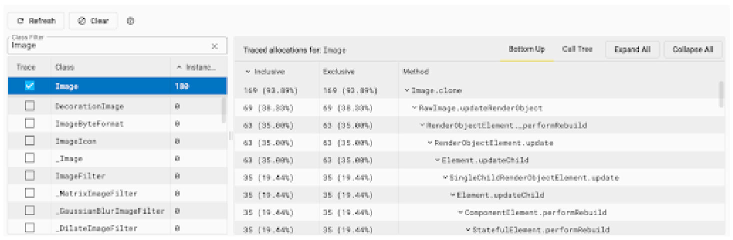
$no-package-libraries
$dart-flutter-libraries

Show only:

package:leaking_counter_1/

Trace Instances tab.

- [임시] 현재는 디버깅 모드에서 실행해야함.
    - 현재 Trace 제대로 작동 안되는듯 함.



Class Filter: Image

Trace	Class	Instanc...
✓	Image	180
☐	DecorationImage	0
☐	ImageByteFormat	0
☐	ImageIcon	0
☐	_Image	0
☐	ImageFilter	0
☐	_MatrixImageFilter	0
☐	_GaussianBlurImageFilter	0
☐	_DilateImageFilter	0

Traced allocations for: Image

Inclusive	Exclusive	Method
169 (93.89%)	169 (93.89%)	Image.clone
69 (38.33%)	69 (38.33%)	RawImage.updateRenderObject
63 (35.00%)	63 (35.00%)	RenderObjectElement._performRebuild
63 (35.00%)	63 (35.00%)	RenderObjectElement.update
63 (35.00%)	63 (35.00%)	Element.updateChild
35 (19.44%)	35 (19.44%)	SingleChildRenderObjectElement.update
35 (19.44%)	35 (19.44%)	Element.updateChild
35 (19.44%)	35 (19.44%)	ComponentElement.performRebuild
35 (19.44%)	35 (19.44%)	StatefulElement.performRebuild

1. 점검하려는 코드가 있는 화면으로 이동
2. **Refresh** 버튼 탭
3. Select a traced class
4. Review the collected data

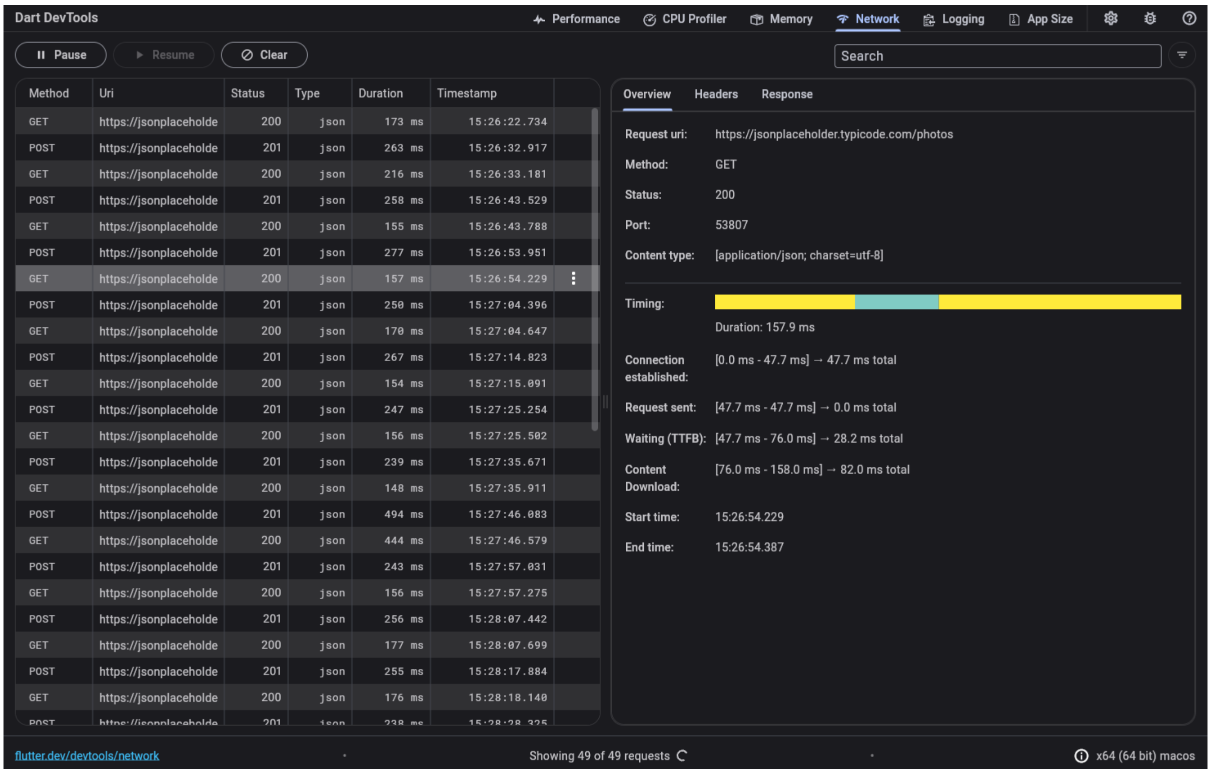
Network View.

What is it?

HTTP, HTTPS, and web socket

- Network View 진입 이후부터 확인 가능

Request, Response 상세 정보



Method	Uri	Status	Type	Duration	Timestamp
GET	https://jsonplaceholde	200	json	173 ms	15:26:22.734
POST	https://jsonplaceholde	201	json	263 ms	15:26:32.917
GET	https://jsonplaceholde	200	json	216 ms	15:26:33.181
POST	https://jsonplaceholde	201	json	258 ms	15:26:43.529
GET	https://jsonplaceholde	200	json	155 ms	15:26:43.788
POST	https://jsonplaceholde	201	json	277 ms	15:26:53.951
GET	https://jsonplaceholde	200	json	157 ms	15:26:54.229
POST	https://jsonplaceholde	201	json	250 ms	15:27:04.396
GET	https://jsonplaceholde	200	json	170 ms	15:27:04.647
POST	https://jsonplaceholde	201	json	267 ms	15:27:14.823
GET	https://jsonplaceholde	200	json	154 ms	15:27:15.091
POST	https://jsonplaceholde	201	json	247 ms	15:27:25.254
GET	https://jsonplaceholde	200	json	156 ms	15:27:25.502
POST	https://jsonplaceholde	201	json	239 ms	15:27:35.671
GET	https://jsonplaceholde	200	json	148 ms	15:27:35.911
POST	https://jsonplaceholde	201	json	494 ms	15:27:46.083
GET	https://jsonplaceholde	200	json	444 ms	15:27:46.579
POST	https://jsonplaceholde	201	json	243 ms	15:27:57.031
GET	https://jsonplaceholde	200	json	156 ms	15:27:57.275
POST	https://jsonplaceholde	201	json	256 ms	15:28:07.442
GET	https://jsonplaceholde	200	json	177 ms	15:28:07.699
POST	https://jsonplaceholde	201	json	255 ms	15:28:17.884
GET	https://jsonplaceholde	200	json	176 ms	15:28:18.140
POST	https://jsonplaceholde	201	json	238 ms	15:28:28.325

Request uri: https://jsonplaceholder.typicode.com/photos

Method: GET

Status: 200

Port: 53807

Content type: [application/json; charset=utf-8]

Timing:

Duration: 157.9 ms

Connection established: [0.0 ms - 47.7 ms] → 47.7 ms total

Request sent: [47.7 ms - 47.7 ms] → 0.0 ms total

Waiting (TTFB): [47.7 ms - 76.0 ms] → 28.2 ms total

Content Download: [76.0 ms - 158.0 ms] → 82.0 ms total

Start time: 15:26:54.229

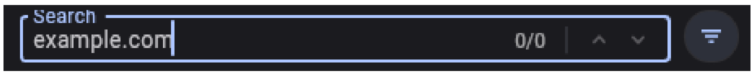
End time: 15:26:54.387

flutter.dev/devtools/network

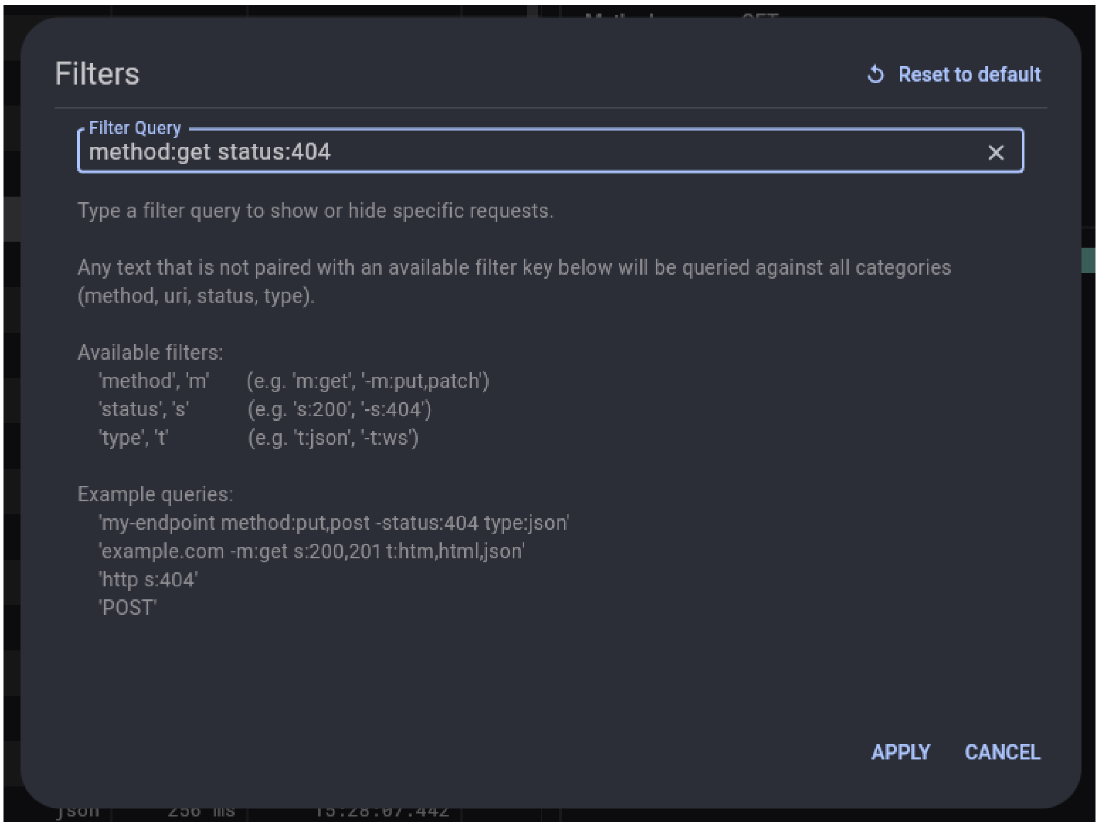
Showing 49 of 49 requests

x64 (64 bit) macos

Search



Filtering



Filters

Reset to default

Filter Query

```
method:get status:404
```

Type a filter query to show or hide specific requests.

Any text that is not paired with an available filter key below will be queried against all categories (method, uri, status, type).

Available filters:

```
    'method', 'm'     (e.g. 'm:get', '-m:put,patch')
    'status', 's'     (e.g. 's:200', '-s:404')
    'type', 't'       (e.g. 't:json', '-t:ws')
```

Example queries:

```
    'my-endpoint method:put,post -status:404 type:json'
    'example.com -m:get s:200,201 t:htm,html,json'
    'http s:404'
    'POST'
```

APPLY CANCEL

Logging view.

What is it?

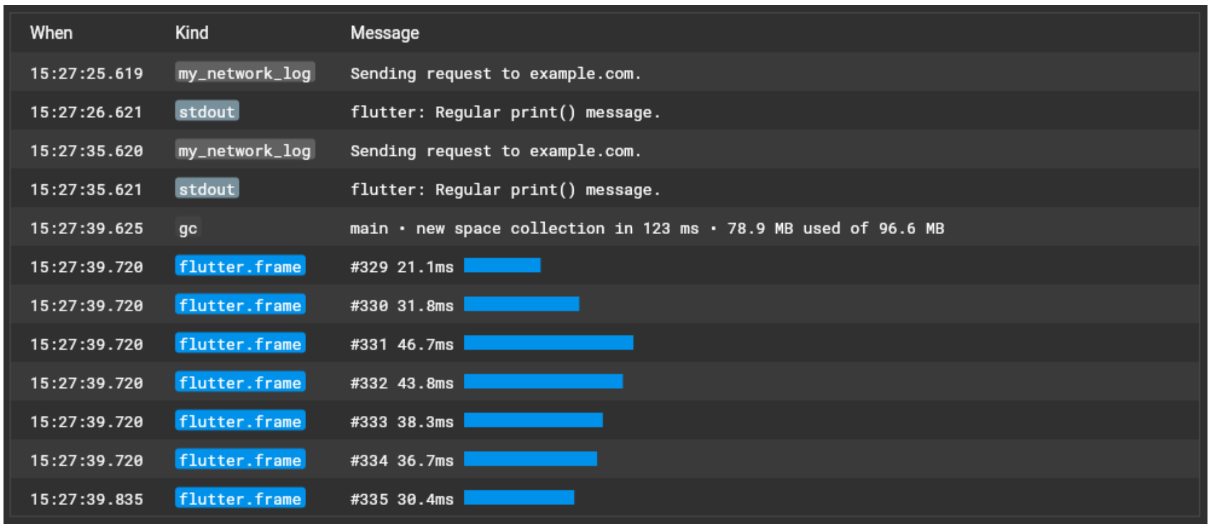
displays **events** from the Dart runtime, application frameworks, application-level logging events.

Standard logging events

- **Garbage collection** events
- **events**, like frame creation events (flutter.frame …)
- `stdout` and `stderr` from applications
- **Custom logging** events from applications

실제 활용

- search
- filters - ex) **kind:gc**
- clear



```
When          Kind              Message
15:27:25.619  my_network_log    Sending request to example.com.
15:27:26.621  stdout            flutter: Regular print() message.
15:27:35.620  my_network_log    Sending request to example.com.
15:27:35.621  stdout            flutter: Regular print() message.
15:27:39.625  gc                main • new space collection in 123 ms • 78.9 MB used of 96.6 MB
15:27:39.720  flutter.frame     #329 21.1ms
15:27:39.720  flutter.frame     #330 31.8ms
15:27:39.720  flutter.frame     #331 46.7ms
15:27:39.720  flutter.frame     #332 43.8ms
15:27:39.720  flutter.frame     #333 38.3ms
15:27:39.720  flutter.frame     #334 36.7ms
15:27:39.835  flutter.frame     #335 30.4ms
```

Logging from your application.

To implement logging in your code,
see the Logging section in the Debugging Flutter apps programmatically page.

App size tool.

What is it?.

analyze the total size of app.

- Analysis tab : size information
- Diff tab : compare two different snapshots

What is “size information”?

size data for Dart code, native code, and non-code elements(assets and fonts)

Dart size information.

- **profile or release mode** only—the AOT compiler
- optimize by removing pieces of code that are **unused or unreachable**

- summarized as the collection of packages, libraries, classes, and functions that exist in the binary output, along with their size in bytes

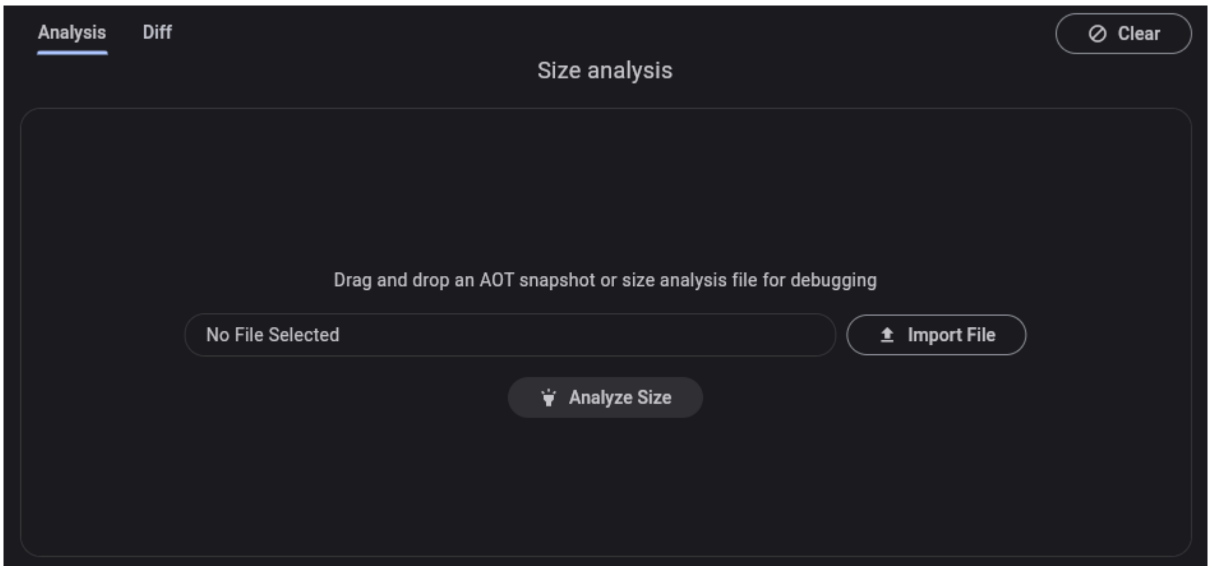
Generating size files.

```
flutter build <your target platform> --analyze-size
```

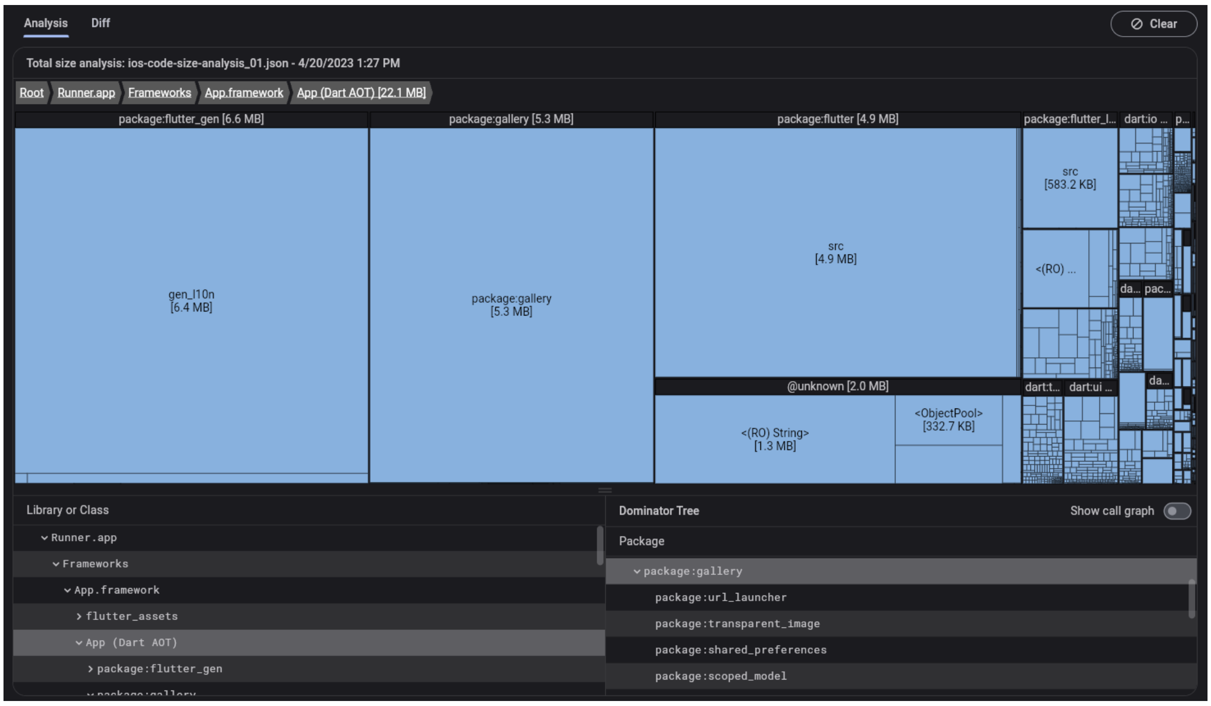
```
flutter build apk --analyze-size --target-platform=android-arm64

...
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
app-release.apk (total compressed)                               6 MB
...
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
A summary of your APK analysis can be found at: build/apk-code-size-analysis_01.json
```

Analysis tab.



Drag and drop an AOT snapshot or size analysis file for debugging



Total size analysis: ios-code-size-analysis_01.json - 4/20/2023 1:27 PM

Root > Runner.app > Frameworks > App.framework > App (Dart AOT) [22.1 MB]

Library or Class

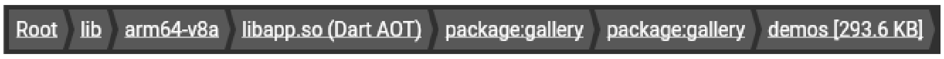
```
⌄ Runner.app
  ⌄ Frameworks
    ⌄ App.framework
      › flutter_assets
      ⌄ App (Dart AOT)
        › package:flutter_gen
        ⌄ package:gallery
```

Dominator Tree　Show call graph

Package

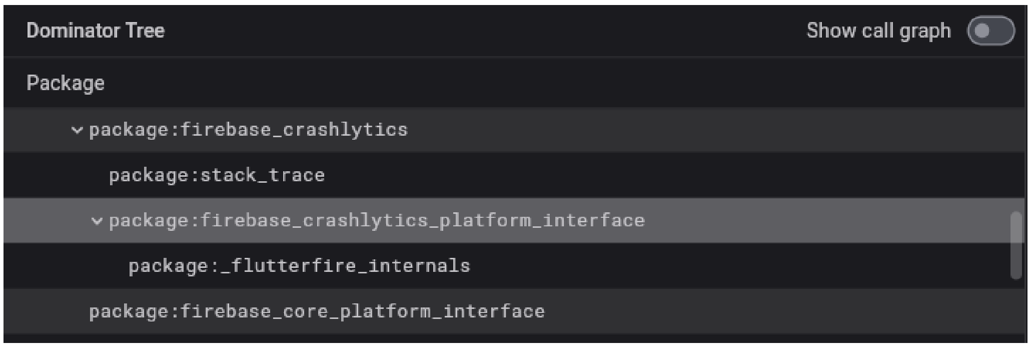
```
⌄ package:gallery
    package:url_launcher
    package:transparent_image
    package:shared_preferences
    package:scoped_model
```

the treemap



Root > lib > arm64-v8a > libapp.so (Dart AOT) > package:gallery > package:gallery > demos [293.6 KB]

dominator tree

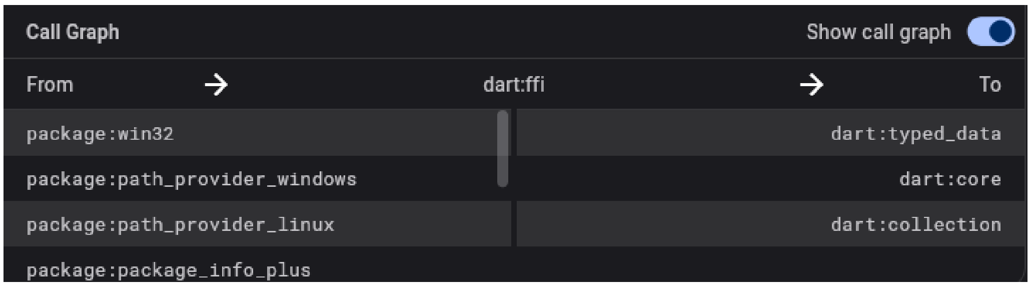


Dominator Tree　Show call graph

Package

```
⌄ package:firebase_crashlytics
    package:stack_trace
  ⌄ package:firebase_crashlytics_platform_interface
      package:_flutterfire_internals
  package:firebase_core_platform_interface
```

call graph

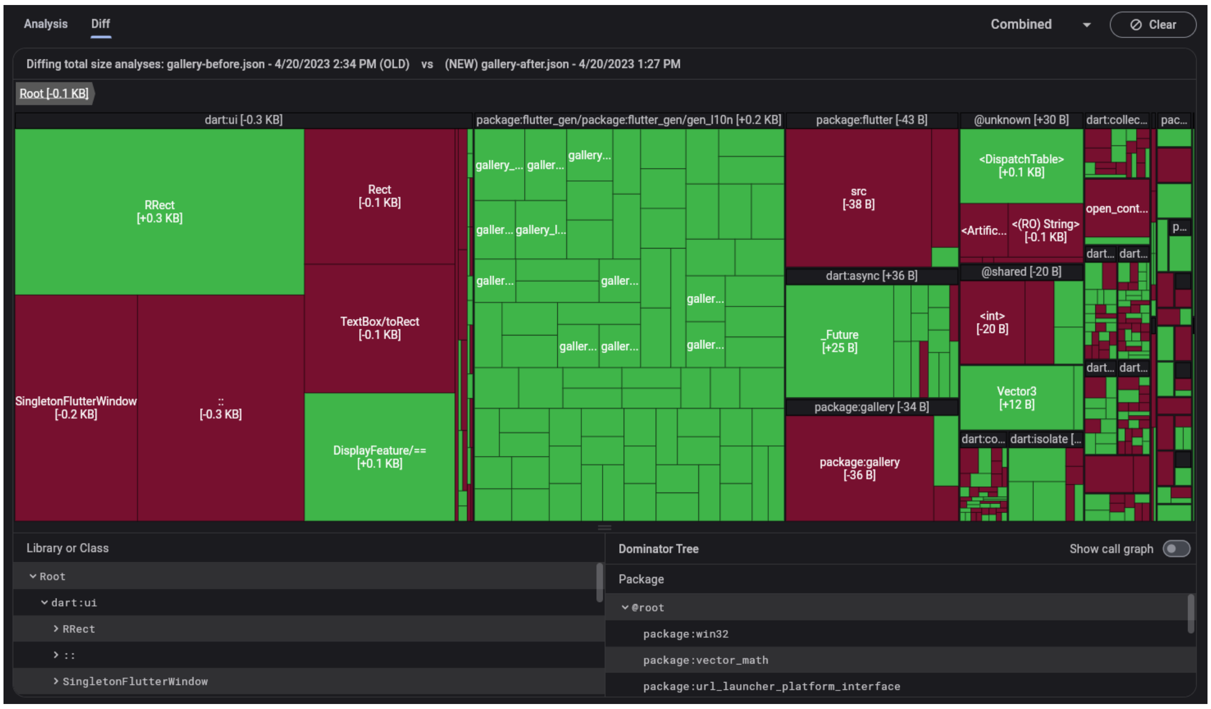


Call Graph — Show call graph

From → dart:ffi → To

From	To
package:win32	dart:typed_data
package:path_provider_windows	dart:core
package:path_provider_linux	dart:collection
package:package_info_plus	

Diff tab.



Diffing total size analyses: gallery-before.json - 4/20/2023 2:34 PM (OLD) vs (NEW) gallery-after.json - 4/20/2023 1:27 PM

Library or Class

```
⌄ Root
  ⌄ dart:ui
    › RRect
    › ::
    › SingletonFlutterWindow
```

Dominator Tree — Show call graph

Package

```
⌄ @root
    package:win32
    package:vector_math
    package:url_launcher_platform_interface
```

참고자료

DevTools

How to use the DevTools with Flutter.

https://docs.flutter.dev/tools/devtools/overview