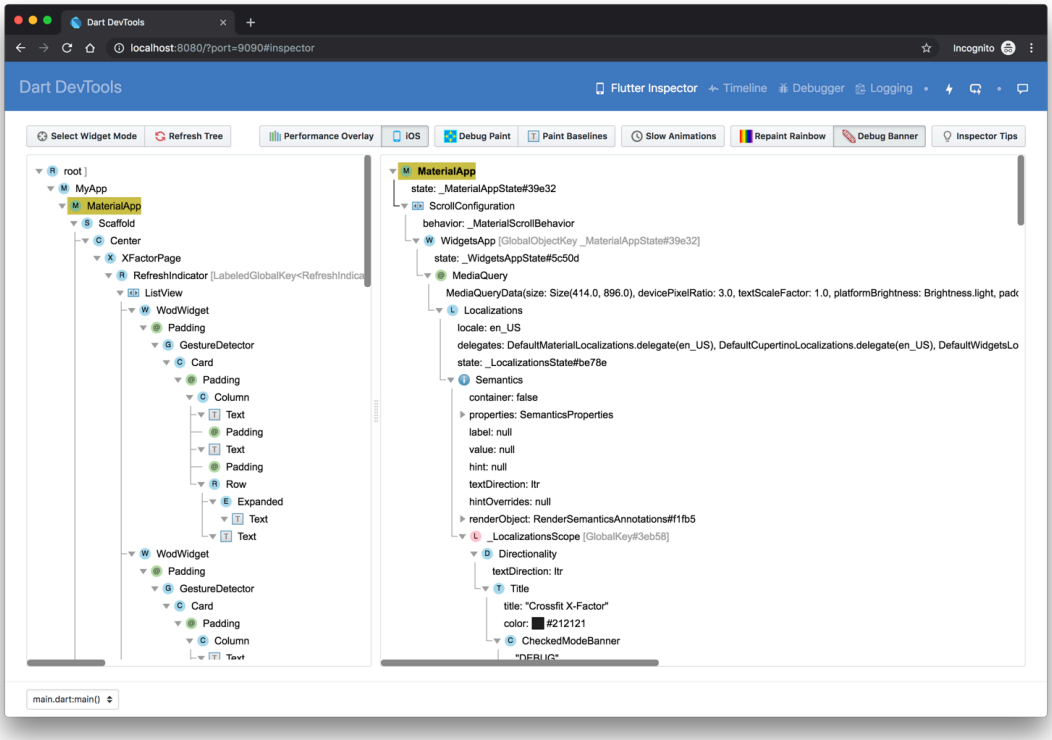
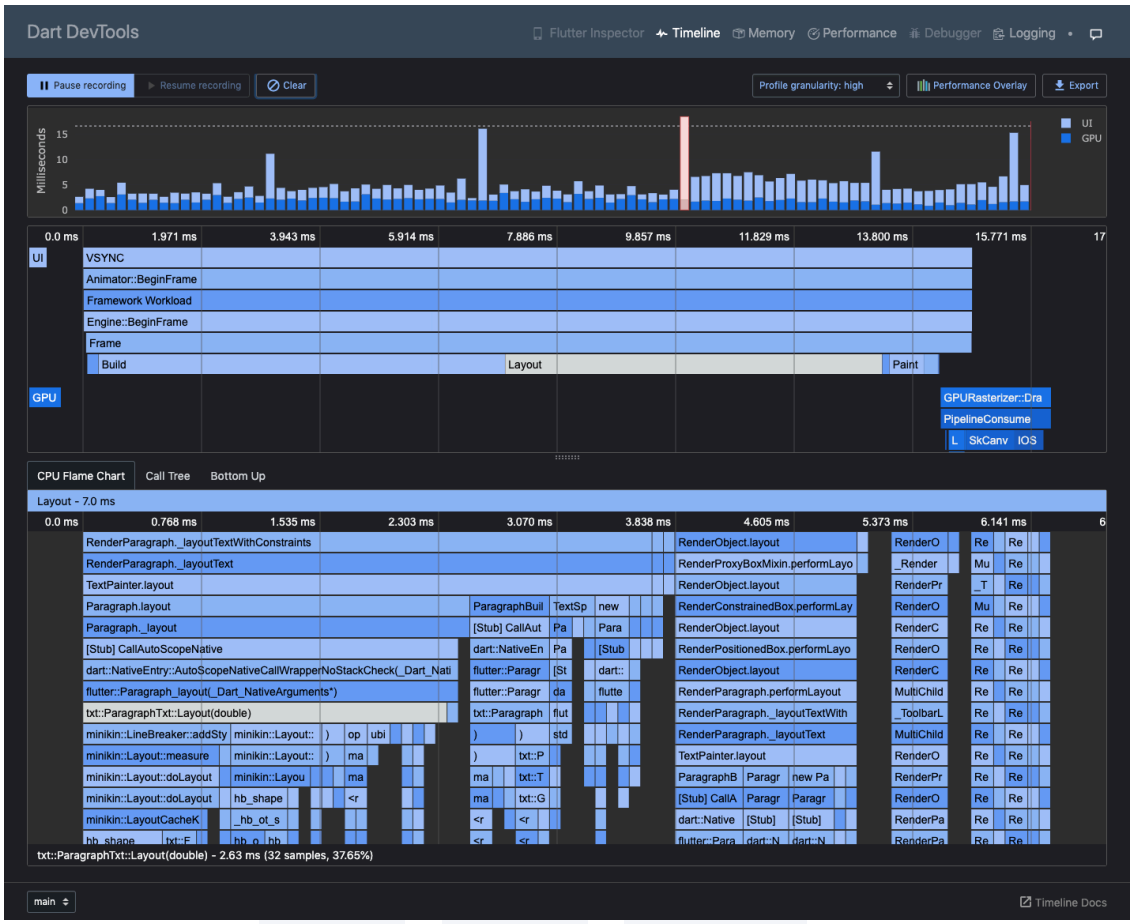
Flutter调试工具devTools是如何工作的





常用的功能就有性能调优，布局查看，函数调用栈等。

安装这个工具可以直接在命令行下执行，用命令行安装是一个比较好的习惯：

```
flutter pub global activate devtools
```

然后，这不，你就会安装一下这些依赖库，如是，就可以对这个devtools的原理进行一个初步的分析。

代码语言：javascript

```
Package devtools is currently active at version 0.1.14.
Resolving dependencies...
+ args 1.5.2
+ async 2.4.0
```

```
+ browser_launcher 0.1.5
+ charcode 1.1.3
+ collection 1.14.12
+ convert 2.1.1
+ crypto 2.1.4
+ devtools 0.1.15
+ devtools_server 0.1.14
+ devtools_shared 0.2.0
+ http 0.12.0+4
+ http_multi_server 2.2.0
+ http_parser 3.1.3
+ intl 0.16.1
+ logging 0.11.4
+ meta 1.1.8
+ mime 0.9.6+3
+ path 1.6.4
+ pedantic 1.9.0
+ shelf 0.7.5
+ shelf_static 0.2.8
+ source_span 1.6.0
+ sse 3.1.2
+ stack_trace 1.9.3
+ stream_channel 2.0.0
+ string_scanner 1.0.5
+ term_glyph 1.1.0
+ typed_data 1.1.6
+ usage 3.4.1
+ uuid 2.0.4
+ vm_service 2.3.1
```

```
+ webkit_inspection_protocol 0.5.0
Downloading devtools 0.1.15...
```

从这些依赖库中，我们发现有以下三个库，也是最值得我们关注的。

devtools 0.1.15

devtools_server 0.1.14

devtools_shared 0.2.0

本文的主要目的是了解清楚devtools是如何从app中拿到数据并且将数据展示给用户的。

下载源码，自己动手编译，把devTools跑起来

要了解这个工具的原理，最好的办法就是下载他的源码，调试它：

```
git clone https://github.com/flutter/devtools
cd devtools/packages/devtools_app
flutter pub get
```

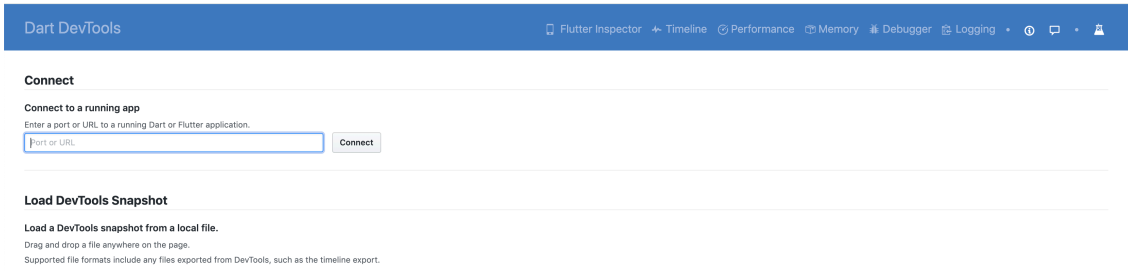
以上源码就把源码下载好，而且相关库都准备好了，应该可以可以开车了。

1、随便找一个flutter的项目，把他跑起来，用做我们debug的数据源，都说这个调试工具要采集数据的，那数据当然是从一个flutter项目来啊。

2、运行这个项目

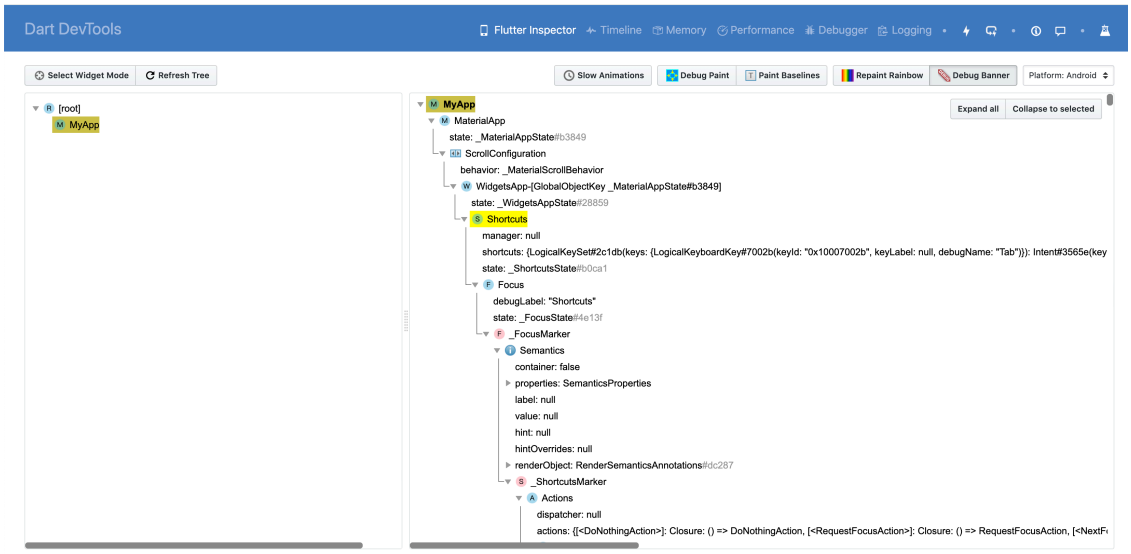
```
cd devtools/packages/devtools_app
alias build_runner=&quot;flutter pub run build_runner&quot;
build_runner serve web
```

3、你就能够看到这个界面了



需要我们输入一个url，其实就是`http://127.0.0.1:49288/GG5v10t9kKQ=`类似这样的一个鬼东西，莫要惊慌失措，这个会在你跑你flutter项目的时候在日志中给出，一定会有，没有你找我。

把url填入进去，连接，就可以看到这个界面了：



从何处来，到何处去

既然已经跑起来了，那么，入口在哪里，很显然，我们发现devtools既然是一个用dart写的项目，那么或许会有一个main.dart，果不其然，在devtools_app/lib下面就找到了main.dart，翻到最后，我们发现了这个。

代码语言：javascript

```
// Now run the app.
  runApp(
```

```
    DevToolsApp(),
  );
```

继续跟踪，还是顶一个目标呢？要不，我们就看看Flutter Inspector是如何把我们 flutter app的树结构显示到devTools上的把，随着深挖下去，我们在app.dart中找到这样一段代码

代码语言：javascript

```
/// The routes that the app exposes.
  final Map&lt;String, UrlParametersBuilder&gt;
_routes = {
    &#39;/&#39;: (_, params) =&gt; Initializer(
          url: params[&#39;uri&#39;],
          builder: (_) =&gt; DevToolsScaffold(
            tabs: const [
              InspectorScreen(),
              TimelineScreen(),
              MemoryScreen(),
              PerformanceScreen(),
              // TODO(https://github.com/flutter/
flutter/issues/43783): Put back
              // the debugger screen.
              if (showNetworkPage)
                NetworkScreen(),
              LoggingScreen(),
              InfoScreen(),
            ],
```

很显然这个排布就够就是我们devtools上面的tab

因此，我们毫不犹豫的点进InspectorScreen这个类中去一看究竟。我

们看到他的initState方法中有一个`_handleConnectionStart`，从名字上看应该是开始连接之后干些啥事，不急，我们先猜一下，我猜，会启动一个service来收集数据，然后启动一个client来查看数据，然后看看代码。

代码语言：javascript

```
setState(() {
      inspectorController?.dispose();
      summaryTreeController =
InspectorTreeControllerFlutter();
      detailsTreeController =
InspectorTreeControllerFlutter();
      inspectorController = InspectorController(
        inspectorTree: summaryTreeController,
        detailsTree: detailsTreeController,
        inspectorService: inspectorService,
        treeType: FlutterTreeType.widget,
        onExpandCollapseSupported:
_onExpandCollapseSupported,
        onLayoutExplorerSupported:
_onLayoutExplorerSupported,
      );
```

我们这里只看到了一个inspectorService，不急，跟到InspectorController里面瞄一瞄。结果我们发现这货其实就是实现了`InspectorServiceClient`

代码语言：javascript

```
class InspectorController extends
DisposableController
```

```
    with AutoDisposeControllerMixin
    implements InspectorServiceClient
```

所以，很显然，这种就是cs架构无疑了。然后，我们深入看一看这个InspectorService，这货肯定就是采集数据的了。然后他是如何创建的，以下是创建它的方法

代码语言：javascript

```
  static Future&lt;InspectorService&gt;
create(VmService vmService) async {
    assert(_inspectorDependenciesLoaded);
    assert(serviceManager.hasConnection);
    assert(serviceManager.service != null);
    final inspectorLibrary = EvalOnDartLibrary(
      inspectorLibraryUriCandidates,
      vmService,
    );

    final libraryRef = await
inspectorLibrary.libraryRef.catchError(
      (_) =&gt; throw
FlutterInspectorLibraryNotFound(),
      test: (e) =&gt; e is LibraryNotFound,
    );
    final libraryFuture =
inspectorLibrary.getLibrary(libraryRef, null);
    final library = await libraryFuture;
    Future&lt;Set&lt;String&gt;&gt;
lookupFunctionNames() async {
      for (ClassRef classRef in library.classes) {
```

```
        if (&#39;WidgetInspectorService&#39; ==
classRef.name) {
          final classObj = await
inspectorLibrary.getClass(classRef, null);
          final functionNames = &lt;String&gt;{};
          for (FuncRef funcRef in
classObj.functions) {
            functionNames.add(funcRef.name);
          }
          return functionNames;
        }
      }
      // WidgetInspectorService is not available.
Either this is not a Flutter
      // application or it is running in profile
mode.
      return null;
    }

    final supportedServiceMethods = await
lookupFunctionNames();
    if (supportedServiceMethods == null) return
null;
    return InspectorService(
      vmService,
      inspectorLibrary,
      supportedServiceMethods,
    );
  }
```

这里，接收一个vm参数，这个参数是哪里来的呢，他是来自一个全局的servermanger，叫做`ServiceConnectionManager`。但是它最终触发他创建的地方在这里：

代码语言：javascript

```
 Future&lt;void&gt; _attemptUrlConnection() async {
    final uri = normalizeVmServiceUri(widget.url);
    final connected = await
FrameworkCore.initVmService(
      &#39;&#39;,
      explicitUri: uri,
      errorReporter: (message, error) =&gt;

Notifications.of(context).push(&#39;$message,
$error&#39;),
    );
    if (!connected) {
      _navigateToConnectPage();
    }
  }
```

你应该还记得你填入的那个进入调试页面，然后填了一个url，回车，没错，就是在这个时候`initVmService`的。

service创建好了，不是用来放着供着的，那是要干活的。我们主要到controller中有这样一个方法：

代码语言：javascript

```
Future&lt;void&gt; maybeLoadUI() async {
    if (!visibleToUser || !isActive) {
      return;
    }
```

```
    if (flutterAppFrameReady) {
      // We need to start by querying the inspector
service to find out the
      // current state of the UI.
      await
inspectorService.inferPubRootDirectoryIfNeeded();
      await updateSelectionFromService(firstFrame:
true);
    } else {
      final ready = await
inspectorService.isWidgetTreeReady();
      flutterAppFrameReady = ready;
      if (isActive && ready) {
        await maybeLoadUI();
      }
    }
  }
```

而这个方法的调用时机就是在 `onFlutterFrame`就是第一帧绘制好的时候，代码就不细贴了，然后，我们注意到有这样一个调用

`await inspectorService.isWidgetTreeReady();`

那么这个Service肯定是去问我们那个app是否应准备好了，那么，他的源码在哪里呢？我看到前面InspectorService创建的时候时候，有一个参数是`inspectorLibraryUriCandidates`,而这个东西实际是：

代码语言：javascript

```
// TODO(jacobr): remove flutter_web entry once
flutter_web and flutter are
// unified.
```

```
const inspectorLibraryUriCandidates = [
  &#39;package:flutter/src/widgets/
widget_inspector.dart&#39;,
  &#39;package:flutter_web/src/widgets/
widget_inspector.dart&#39;,
];
```

稍微追踪一下代码，就能够发现isWidgetTreeReady，就是去问`package:flutter/src/widgets/widget_inspector.dart`这个类中的方法。然后我们看一看isWidgetTreeReady的实现：

代码语言：javascript

```
 /// If the widget tree is not ready, the
application should wait for the next
  /// Flutter.Frame event before attempting to
display the widget tree. If the
  /// application is ready, the next Flutter.Frame
event may never come as no
  /// new frames will be triggered to draw unless
something changes in the UI.
  Future&lt;bool&gt; isWidgetTreeReady() {
    return
invokeBoolServiceMethodNoArgs(&#39;isWidgetTreeRead
y&#39;);
  }
```

很加单，就是一个方法调用，这应该就是调用flutter框架中的方法了。

代码语言：javascript

```
@protected
  bool isWidgetTreeReady([ String groupName ]) {
    return WidgetsBinding.instance != null &&
```

```
WidgetsBinding.instance.debugDidSendFirstFrameEvent
;
  }
```

实际上还不是直接调这个方法，还经过了一个映射，最后映射到这个方法上来了。

所以，我们要去第一帧的数据的化，那么，我们就要去看看maybeLoadUI这个方法中这个调用了。`inspectorService.inferPubRootDirectoryIfNeeded()`，

代码语言：javascript

```
Future&lt;String&gt;
inferPubRootDirectoryIfNeeded() async {
    final group =
createObjectGroup(&#39;temp&#39;);
    final root = await
group.getRoot(FlutterTreeType.widget);

    if (root == null) {
      // No need to do anything as there isn&#39;t
a valid tree (yet?).
      await group.dispose();
      return null;
    }
    final children = await root.children;
    if (children?.isNotEmpty == true) {
      // There are already widgets identified as
being from the summary tree so
      // no need to guess the pub root directory.
```

```
      return null;
    }

    final List&lt;RemoteDiagnosticsNode&gt;
allChildren =
        await
group.getChildren(root.dartDiagnosticRef, false,
null);
    final path =
allChildren.first.creationLocation?.path;
    if (path == null) {
      await group.dispose();
      return null;
    }

    // this directory rather than guessing based on
url structure.
    final parts = path.split(&#39;/&#39;);
    String pubRootDirectory;
    for (int i = parts.length - 1; i &gt;= 0; i--)
{
      String part;
      if (part == &#39;lib&#39; || part ==
&#39;web&#39;) {
        pubRootDirectory = parts.sublist(0,
i).join(&#39;/&#39;);
        break;
      }
```

```
      if (part == &#39;packages&#39;) {
        pubRootDirectory = parts.sublist(0, i +
1).join(&#39;/&#39;);
        break;
      }
    }
    pubRootDirectory ??=
(parts..removeLast()).join(&#39;/&#39;);

    await
setPubRootDirectories([pubRootDirectory]);
    await group.dispose();
    return pubRootDirectory;
  }
```

所以，至此就拿到了flutter页渲染的那个树，返回的信息是一个string，其实是存放那个树对应的List<RemoteDiagnosticsNode>的地址，目次是可以恢复的，就没有必要往下追踪了。

之间使用什么数据互通

通过具体的方法，我们可以看到：

代码语言：javascript

```
 /// Returns a JSON representation of the subtree
rooted at the
  /// [DiagnosticsNode] object that
`diagnosticsNodeId` references providing
  /// information needed for the details subtree
view.
  ///
  /// The number of levels of the subtree that
```

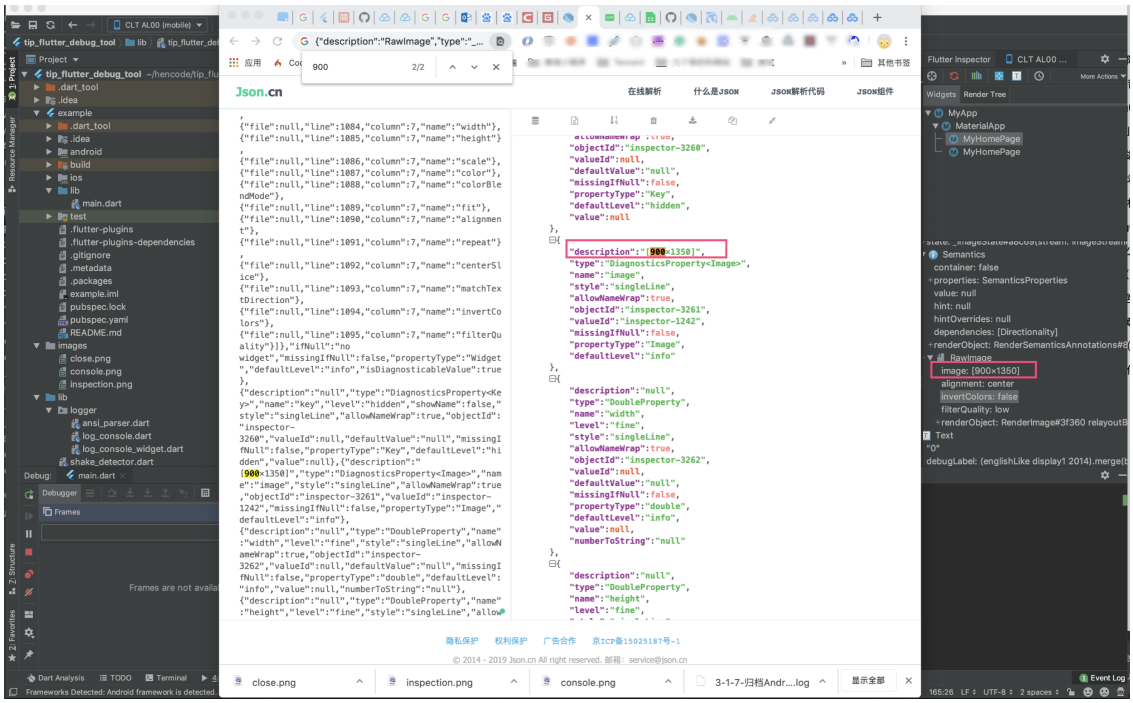
```
should be returned is specified
  /// by the [subtreeDepth] parameter. This value
defaults to 2 for backwards
  /// compatibility.
  ///
  /// See also:
  ///
  ///  * [getChildrenDetailsSubtree], a method to
get children of a node
  ///    in the details subtree.
  String getDetailsSubtree(
    String id,
    String groupName, {
    int subtreeDepth = 2,
  }) {
    return _safeJsonEncode(_getDetailsSubtree( id,
groupName, subtreeDepth));
  }
```

最终方法的调用将会回调会一个json数据，举个例子，大概是：

```
⊟{
    "description":"RichText",
    "type":"_ElementDiagnosticableTreeNode",
    "style":"dense",
    "allowWrap":false,
    "objectId":"inspector-1782",
    "valueId":"inspector-31",
    "locationId":14,
    "creationLocation":⊟{
        "file":"file:///Users/xx/xx/flutter/packages/flutter/lib/src/widgets/text.dart",
        "line":425,
        "column":21,
        "parameterLocations":⊟[
            ⊟{
                "file":null,
                "line":426,
                "column":7,
                "name":"textAlign"
            },
            ⊟{
                "file":null,
                "line":427,
                "column":7,
                "name":"textDirection"
            },
            ⊟{
                "file":null,
                "line":428
```


然后更具这些信息，devTools上呈现出树状接口的ui，然后devTools其实还可以反过来控制app上显示debug标志等其他操作，其实这都是通过service发送触发那边的方法调用。

下图是我验证了一下，这些数据是否和工具展示的对得上，验证结果是可以对上的：


tip_flutter_debug_tool
Project
.dart_tool
.idea
example
.dart_tool
.idea
android
build
ios
lib
main.dart
test
.flutter-plugins
.flutter-plugins-dependencies
.gitignore
.metadata
.packages
example.iml
pubspec.lock
pubspec.yaml
README.md
images
close.png
console.png
inspection.png
lib
logger
ansi_parser.dart
log_console.dart
log_console_widget.dart
shake_detector.dart
Debug: main.dart
Debugger
Frames
Frames are not availa
Dart Analysis
TODO
Terminal
Frameworks Detected: Android framework is detected
{"description":"RawImage","type":"...
900
2/2
Json.cn
在线解析
什么是JSON
JSON解析代码
JSON组件
"objectId":"inspector-3260",
"valueId":null,
"defaultValue":"null",
"missingIfNull":false,
"propertyType":"Key",
"defaultLevel":"hidden",
"value":null
"description":"[900×1350]",
"type":"DiagnosticsProperty<Image>",
"name":"image",
"style":"singleLine",
"allowNameWrap":true,
"objectId":"inspector-3261",
"valueId":"inspector-1242",
"missingIfNull":false,
"propertyType":"Image",
"defaultLevel":"info"
"description":"null",
"type":"DoubleProperty",
"name":"width",
"level":"fine",
"style":"singleLine",
"allowNameWrap":true,
"objectId":"inspector-3262",
"valueId":null,
"defaultValue":"null",
"missingIfNull":false,
"propertyType":"double",
"defaultLevel":"info",
"value":null,
"numberToString":"null"
"description":"null",
"type":"DoubleProperty",
"name":"height",
"level":"fine",
隐私保护
权利保护
广告合作
© 2014 - 2019 Json.cn All right reserved.
close.png
inspection.png
console.png
显示全部
Flutter Inspector
Widgets
Render Tree
MyApp
MaterialApp
MyHomePage
MyHomePage
Semantics
container: false
properties: SemanticsProperties
value: null
hint: null
hintOverrides: null
dependencies: [Directionality]
renderObject: RenderSemanticsAnnotations#
RawImage
image: [900×1350]
alignment: center
invertColors: false
filterQuality: low
renderObject: RenderImage#3f360
Text
"0"
debugLabel: (englishLike display1 2014).merge(
Event Log
165:26
LF
UTF-8
2 spaces