

本文档适用于Android开发人员，可以将您现有的Android知识应用于使用Flutter构建移动应用程序。如果您了解Android框架的基础知识，那么您可以将此文档用作Flutter开发的一个开端。

使用Flutter构建应用时，您的Android知识和技能非常有价值，因为Flutter依赖移动操作系统提供众多功能和配置。Flutter是一种为移动设备构建用户界面的新方式，但它有一个插件系统可与Android（和iOS）进行非UI的任务通信。如果您是Android专家，则不必重新学习使用Flutter的所有内容。

可以将此文档作为cookbook，通过跳转并查找与您的需求最相关的问题。

- [Views](#)
 - [Flutter和Android中的View](#)
 - [如何更新widget](#)
 - [如何布局？XML layout 文件跑哪去了？](#)
 - [如何在布局中添加或删除组件](#)
 - [在Android中，可以通过View.animate\(\)对视图进行动画处理，那在Flutter中怎样才能对Widget进行处理](#)
 - [如何使用Canvas draw/paint](#)
 - [如何构建自定义 Widgets](#)
- [Intents](#)
 - [Intent在Flutter中等价于什么？](#)
 - [如何在Flutter中处理来自外部应用程序传入的Intents](#)
 - [startActivityForResult 在Flutter中等价于什么](#)
- [异步UI](#)
 - [runOnUiThread 在Flutter中等价于什么](#)
 - [AsyncTask和IntentService在Flutter中等价于什么](#)
 - [OkHttp在Flutter中等价于什么](#)
 - [如何在Flutter中显示进度指示器](#)

- [项目结构和资源](#)
 - [在哪里存储分辨率相关的图片文件? HDPI/XXHDPI](#)
 - [在哪里存储字符串? 如何存储不同的语言](#)
 - [Android Gradle vs Flutter pubspec.yaml](#)
- [Activities 和 Fragments](#)
 - [Activity和Fragment 在Flutter中等价于什么](#)
 - [如何监听Android Activity生命周期事件](#)
- [Layouts](#)
 - [LinearLayout在Flutter中相当于什么](#)
 - [RelativeLayout在Flutter中等价于什么](#)
 - [ScrollView在Flutter中等价于什么](#)
- [手势检测和触摸事件处理](#)
 - [如何将一个onClick监听器添加到Flutter中的widget](#)
 - [如何处理widget上的其他手势](#)
- [ListView & Adapter](#)
 - [ListView在Flutter中相当于什么](#)
 - [怎么知道哪个列表项被点击](#)
 - [如何动态更新ListView](#)
- [使用 Text](#)
 - [如何在 Text widget上设置自定义字体](#)
 - [如何在Text上定义样式](#)
- [表单输入](#)
 - [Input的“ hint” 在flutter中相当于什么](#)
 - [如何显示验证错误](#)

- [Flutter 插件](#)
 - [如何使用 GPS sensor](#)
 - [如何访问相机](#)
 - [如何使用Facebook登陆](#)
 - [如何构建自定义集成Native功能](#)
 - [如何在我的Flutter应用程序中使用NDK](#)
- [主题](#)
 - [如何构建Material主题风格的app](#)
- [数据库和本地存储](#)
 - [如何在Flutter中访问Shared Preferences ?](#)
 - [如何在Flutter中访问SQLite](#)
- [通知](#)
 - [如何设置推送通知](#)

Views

Flutter和Android中的View

在Android中，View是屏幕上显示的所有内容的基础，按钮、工具栏、输入框等一切都是View。在Flutter中，View相当于是Widget。然而，与View相比，Widget有一些不同之处。首先，Widget仅支持一帧，并且在每一帧上，Flutter的框架都会创建一个Widget实例树(译者语：相当于一次性绘制整个界面)。相比之下，在Android上View绘制结束后，就不会重绘，直到调用invalidate时才会重绘。

与Android的视图层次系统不同（在framework改变视图），而在Flutter中的widget是不可变的，这允许widget变得超级轻量。

如何更新widget

在Android中，您可以通过直接对view进行改变来更新视图。然而，在Flutter中Widget是不可变的，不会直接更新，而必须使用Widget的状态。

这是Stateful和Stateless widget的概念的来源。一个Stateless Widget就像它的名字，是一个没有状态信息的widget。

当您所需要的用户界面不依赖于对象配置信息以外的其他任何内容时，StatelessWidgets非常有用。

例如，在Android中，这与将logo图标放在ImageView中很相似。logo在运行时不会更改，因此您可以在Flutter中使用StatelessWidget。

如果您希望通过HTTP动态请求的数据更改用户界面，则必须使用StatefulWidget，并告诉Flutter框架该widget的状态已更新，以便可以更新该widget。

这里要注意的重要一点是无状态和有状态 widget的核心特性是相同的。每一帧它们都会重新构建，不同之处在于StatefulWidget有一个State对象，它可以跨帧存储状态数据并恢复它。

如果你有疑问，那么要记住这个规则：如果一个widget发生了变化（例如用户与它交互），它就是有状态的。但是，如果一个子widget对变化做出反应，而其父widget对变化没有反应，那么包含的父widget仍然可以是无状态的widget。

我们来看看你如何使用一个StatelessWidget。一个常见的StatelessWidget是Text。如果你看看Text Widget的实现，你会发现它是一个StatelessWidget的子类：

```
new Text(  
  'I like Flutter!',  
  style: new TextStyle(fontWeight: FontWeight.bold),  
);
```

正如你所看到的，Text 没有与之关联的状态信息，它呈现了构造函数中传递的内容，仅此而已。

但是，如果你想让“I Like Flutter”动态变化，例如点击一个FloatingActionButton？这可以通过将Text包装在StatefulWidget中并在点击按钮时更新它来实现，如：

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(new SampleApp());  
}  
  
class SampleApp extends StatelessWidget {  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return new MaterialApp(  
      title: 'Sample App',  
      theme: new ThemeData(  
        primarySwatch: Colors.blue,
```

```

    ),
    home: new SampleAppPage(),
  );
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  // Default placeholder text
  String textToShow = "I Like Flutter";

  void _updateText() {
    setState(() {
      // update the text
      textToShow = "Flutter is Awesome!";
    });
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new Center(child: new Text(textToShow)),
      floatingActionButton: new FloatingActionButton(
        onPressed: _updateText,
        tooltip: 'Update Text',
        child: new Icon(Icons.update),
      ),
    );
  }
}

```

如何布局？ XML layout 文件跑哪去了？

在Android中，您通过XML编写布局，但在Flutter中，您可以使用widget树来编写布局。

这里是一个例子，展示了如何在屏幕上显示一个简单的Widget并添加一些padding。

```

@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Sample App"),
    ),
    body: new Center(
      child: new MaterialButton(
        onPressed: () {},

```

```

        child: new Text('Hello'),
        padding: new EdgeInsets.only(left: 10.0, right: 10.0),
      ),
    ),
  );
}

```

您可以查看Flutter所提供的所有布局: [Flutter widget layout](#)

如何在布局中添加或删除组件

在Android中，您可以从父级控件调用addChild或removeChild以动态添加或删除View。在Flutter中，因为widget是不可变的，所以没有addChild。相反，您可以传入一个函数，该函数返回一个widget给父项，并通过布尔值控制该widget的创建。

例如，当你点击一个FloatingActionButton时，如何在两个widget之间切换：

```

import 'package:flutter/material.dart';

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  // Default value for toggle
  bool toggle = true;
  void _toggle() {
    setState(() {
      toggle = !toggle;
    });
  }

  _getToggleChild() {
    if (toggle) {

```

```

        return new Text('Toggle One');
      } else {
        return new MaterialButton(onPressed: () {}, child: new Text('Toggle Two'));
      }
    }
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new Center(
        child: _getToggleChild(),
      ),
      floatingActionButton: new FloatingActionButton(
        onPressed: _toggle,
        tooltip: 'Update Text',
        child: new Icon(Icons.update),
      ),
    );
  }
}

```

在Android中，可以通过View.animate()对视图进行动画处理，那在Flutter中怎样才能对Widget进行处理

在Flutter中，可以通过动画库给widget添加动画。

在Android中，您可以通过XML创建动画或在视图上调用.animate()。在Flutter中，您可以将widget包装到Animation中。

与Android相似，在Flutter中，您有一个AnimationController和一个Interpolator，它是Animation类的扩展，例如CurvedAnimation。您将控制器和动画传递到AnimationWidget中，并告诉控制器启动动画。

让我们来看看如何编写一个FadeTransition，当您按下时会淡入一个logo:

```

import 'package:flutter/material.dart';

void main() {
  runApp(new FadeAppTest());
}

class FadeAppTest extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Fade Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,

```

```

    ),
    home: new MyFadeTest(title: 'Fade Demo'),
  );
}

class MyFadeTest extends StatefulWidget {
  MyFadeTest({Key key, this.title}) : super(key: key);
  final String title;
  @override
  _MyFadeTest createState() => new _MyFadeTest();
}

class _MyFadeTest extends State<MyFadeTest> with TickerProviderStateMixin {
  AnimationController controller;
  CurvedAnimation curve;

  @override
  void initState() {
    controller = new AnimationController(duration: const Duration(milliseconds: 2000), vsync: this);
    curve = new CurvedAnimation(parent: controller, curve: Curves.easeIn);
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
      ),
      body: new Center(
        child: new Container(
          child: new FadeTransition(
            opacity: curve,
            child: new FlutterLogo(
              size: 100.0,
            )),
        floatingActionButton: new FloatingActionButton(
          tooltip: 'Fade',
          child: new Icon(Icons.brush),
          onPressed: () {
            controller.forward();
          },
        ),
      ),
    );
  }
}

```

See <https://flutter.io/widgets/animation/> and <https://flutter.io/tutorials/animation> for more specific details.

如何使用Canvas draw/paint

在Android中，您可以使用Canvas在屏幕上绘制自定义形状。

Flutter有两个类可以帮助您绘制画布，CustomPaint和CustomPainter，它们实现您的算法以绘制到画布。

在这个人气较高的StackOverFlow答案中，您可以看到签名painter是如何实现的。

请参阅<https://stackoverflow.com/questions/46241071/create-signature-area-for-mobile-app-in-dart-flutter>

```
import 'package:flutter/material.dart';
class SignaturePainter extends CustomPainter {
  SignaturePainter(this.points);
  final List<Offset> points;
  void paint(Canvas canvas, Size size) {
    var paint = new Paint()
      ..color = Colors.black
      ..strokeCap = StrokeCap.round
      ..strokeWidth = 5.0;
    for (int i = 0; i < points.length - 1; i++) {
      if (points[i] != null && points[i + 1] != null)
        canvas.drawLine(points[i], points[i + 1], paint);
    }
  }
  bool shouldRepaint(SignaturePainter other) => other.points != points;
}
class Signature extends StatefulWidget {
  SignatureState createState() => new SignatureState();
}
class SignatureState extends State<Signature> {
  List<Offset> _points = <Offset>[];
  Widget build(BuildContext context) {
    return new GestureDetector(
      onPanUpdate: (DragUpdateDetails details) {
        setState(() {
          RenderBox referenceBox = context.findRenderObject();
          Offset localPosition =
            referenceBox.globalToLocal(details.globalPosition);
          _points = new List.from(_points)..add(localPosition);
        });
      },
      onPanEnd: (DragEndDetails details) => _points.add(null),
      child: new CustomPaint(painter: new SignaturePainter(_points)),
    );
  }
}
class DemoApp extends StatelessWidget {
  Widget build(BuildContext context) => new Scaffold(body: new Signature());
}
void main() => runApp(new MaterialApp(home: new DemoApp()));
```

如何构建自定义 Widgets

在Android中，您通常会继承View或已经存在的某个控件，然后覆盖其绘制方法来实现自定义View。

在Flutter中，一个自定义widget通常是通过组合其它widget来实现的，而不是继承。我们来看看如何构建持有一个label的CustomButton。这是通过将Text与RaisedButton组合来实现的，而不是扩展RaisedButton并重写其绘制方法实现：

```
class CustomButton extends StatelessWidget {
  final String label;

  CustomButton(this.label);

  @override
  Widget build(BuildContext context) {
    return new RaisedButton(onPressed: () {}, child: new Text(label));
  }
}
```

Then you can use this CustomButton just like you would with any other Widget:

```
@override
Widget build(BuildContext context) {
  return new Center(
    child: new CustomButton("Hello"),
  );
}
```

Intents

Intent在Flutter中等价于什么？

在Android中，Intents主要有两种使用场景：在Activity之间切换，以及调用外部组件。Flutter不具有Intents的概念，但如果需要的话，Flutter可以通过Native整合来触发Intents。

要在Flutter中切换屏幕，您可以访问路由以绘制新的Widget。管理多个屏幕有两个核心概念和类：Route 和 Navigator。Route是应用程序的“屏幕”或“页面”的抽象（可以认为是Activity），Navigator是管理Route的Widget。Navigator可以通过push和pop route以实现页面切换。

和Android相似，您可以在AndroidManifest.xml中声明您的Activities，在Flutter中，您可以将具有指定Route的Map传递到顶层MaterialApp实例

```
void main() {
  runApp(new MaterialApp(
    home: new MyAppHome(), // becomes the route named '/'
    routes: <String, WidgetBuilder> {
      '/a': (BuildContext context) => new MyPage(title: 'page A'),
      '/b': (BuildContext context) => new MyPage(title: 'page B'),
      '/c': (BuildContext context) => new MyPage(title: 'page C'),
    },
  ));
}
```

然后，您可以通过Navigator来切换到命名路由的页面。

```
Navigator.of(context).pushNamed('/b');
```

Intents的另一个主要的用途是调用外部组件，如Camera或File picker。为此，您需要和native集成（或使用现有的库）

参阅 [Flutter Plugins] 了解如何与native集成.

如何在Flutter中处理来自外部应用程序传入的Intents

Flutter可以通过直接与Android层通信并请求共享的数据来处理来自Android的Intents
在这个例子中，我们注册文本共享intent，所以其他应用程序可以共享文本到我们的Flutter应用程序

这个应用程序的基本流程是我们首先处理Android端的共享文本数据，然后等待Flutter请求数据，然后通过MethodChannel发送。

首先在在AndroidManifest.xml中注册我们想要处理的intent

```
<activity
    android:name=".MainActivity"
    android:launchMode="singleTop"
    android:theme="@style/LaunchTheme"
    android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale|layoutDirection"
    android:hardwareAccelerated="true"
    android:windowSoftInputMode="adjustResize">
    <!-- This keeps the window background of the activity showing
         until Flutter renders its first frame. It can be removed if
         there is no splash screen (such as the default splash screen
         defined in @style/LaunchTheme). -->

    <meta-data
        android:name="io.flutter.app.android.SplashScreenUntilFirstFrame"
        android:value="true" />

    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

然后，在MainActivity中，您可以处理intent，一旦我们从intent中获得共享文本数据，我们就会持有它，直到Flutter在完成准备就绪时请求它。

```
package com.yourcompany.shared;
```

```
import android.content.Intent;
```

```
import android.os.Bundle;
```

```
import java.nio.ByteBuffer;
```

```
import io.flutter.app.FlutterActivity;
```

```

import io.flutter.plugin.common.ActivityLifecycleListener;
import io.flutter.plugin.common.MethodCall;
import io.flutter.plugin.common.MethodChannel;
import io.flutter.plugins.GeneratedPluginRegistrant;

public class MainActivity extends FlutterActivity {
    String sharedText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        GeneratedPluginRegistrant.registerWith(this);
        Intent intent = getIntent();
        String action = intent.getAction();
        String type = intent.getType();

        if (Intent.ACTION_SEND.equals(action) && type != null) {
            if ("text/plain".equals(type)) {
                handleSendText(intent); // Handle text being sent
            }
        }

        new MethodChannel(getFlutterView(), "app.channel.shared.data").setMethodCallHandler(new
MethodChannel.MethodCallHandler() {
            @Override
            public void onMethodCall(MethodCall methodCall, MethodChannel.Result result) {
                if (methodCall.method.equals("getSharedText")) {
                    result.success(sharedText);
                    sharedText = null;
                }
            }
        });
    }

    void handleSendText(Intent intent) {
        sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    }
}

```

最后，在Flutter中，您可以在渲染Flutter视图时请求数据。

```

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

void main() {
    runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
    // This widget is the root of your application.
    @override
    Widget build(BuildContext context) {
        return new MaterialApp(
            title: 'Sample Shared App Handler',
            theme: new ThemeData(

```

```

        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  static const platform = const MethodChannel('app.channel.shared.data');
  String dataShared = "No data";

  @override
  void initState() {
    super.initState();
    getSharedText();
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(body: new Center(child: new Text(dataShared)));
  }

  getSharedText() async {
    var sharedData = await platform.invokeMethod("getSharedText");
    if (sharedData != null) {
      setState(() {
        dataShared = sharedData;
      });
    }
  }
}

```

startActivityForResult 在Flutter中等价于什么

处理Flutter中所有路由的Navigator类可用于从已经push到栈的路由中获取结果。这可以通过等待push返回的Future来完成。例如，如果您要启动让用户选择其位置的路线的路由，则可以执行以下操作：

```
Map coordinates = await Navigator.of(context).pushNamed('/location');
```

然后在你的位置路由中，一旦用户选择了他们的位置，你可以将结果”pop”出栈

```
Navigator.of(context).pop({"lat":43.821757,"long":-79.226392});
```

异步UI

runOnUiThread 在Flutter中等价于什么

Dart是单线程执行模型，支持Isolates（在另一个线程上运行Dart代码的方式）、事件循环和异步编程。除非您启动一个Isolate，否则您的Dart代码将在主UI线程中运行，并由事件循环驱动（译者语：和JavaScript一样）。

例如，您可以在UI线程上运行网络请求代码而不会导致UI挂起(译者语：因为网络请求是异步的)：

```
loadData() async {  
  String dataURL = "https://jsonplaceholder.typicode.com/posts";  
  http.Response response = await http.get(dataURL);  
  setState(() {  
    widgets = JSON.decode(response.body);  
  });  
}
```

要更新UI，您可以调用setState，这会触发build方法再次运行并更新数据。

以下是异步加载数据并将其显示在ListView中的完整示例：

```
import 'dart:convert';  
  
import 'package:flutter/material.dart';  
import 'package:http/http.dart' as http;  
  
void main() {  
  runApp(new SampleApp());  
}  
  
class SampleApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return new MaterialApp(  
      title: 'Sample App',  
      theme: new ThemeData(  
        primarySwatch: Colors.blue,  
      ),  
      home: new SampleAppPage(),  
    );  
  }  
}  
  
class SampleAppPage extends StatefulWidget {  
  SampleAppPage({Key key}) : super(key: key);  
  
  @override  
  _SampleAppPageState createState() => new _SampleAppPageState();  
}  
  
class _SampleAppPageState extends State<SampleAppPage> {  
  List widgets = [];  
  
  @override  
  void initState() {  
    super.initState();  
  }  
}
```

```

    loadData();
}

@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Sample App"),
    ),
    body: new ListView.builder(
      itemCount: widgets.length,
      itemBuilder: (BuildContext context, int position) {
        return getRow(position);
      }
    ));
}

Widget getRow(int i) {
  return new Padding(
    padding: new EdgeInsets.all(10.0),
    child: new Text("Row ${widgets[i]["title"]}"),
  );
}

loadData() async {
  String dataURL = "https://jsonplaceholder.typicode.com/posts";
  http.Response response = await http.get(dataURL);
  setState(() {
    widgets = JSON.decode(response.body);
  });
}
}

```

AsyncTask和IntentService在Flutter中等价于什么

在Android中，当你想访问一个网络资源时，你通常会创建一个AsyncTask，它将在UI线程之外运行代码来防止你的UI被阻塞。AsyncTask有一个线程池，可以为你管理线程。

由于Flutter是单线程的，运行一个事件循环（如Node.js），所以您不必担心线程管理或者使用AsyncTasks、IntentServices。

要异步运行代码，可以将函数声明为异步函数，并在该函数中等待这个耗时任务

```

loadData() async {
  String dataURL = "https://jsonplaceholder.typicode.com/posts";
  http.Response response = await http.get(dataURL);
  setState(() {
    widgets = JSON.decode(response.body);
  });
}

```

这就是典型的进行网络或数据库调用的方式

在Android上，当您继承AsyncTask时，通常会覆盖3个方法，OnPreExecute、doInBackground和onPostExecute。在Flutter中没有这种模式的等价物，因为您只需

等待一个长时间运行的函数，而Dart的事件循环将负责其余的事情。

但是，有时您可能需要处理大量数据，导致UI可能会挂起。

在这种情况下，与AsyncTask一样，在Flutter中，可以利用多个CPU内核来执行耗时或计算密集型任务。这是通过使用Isolates来完成的。

是一个独立的执行线程，它运行时不会与主线程共享任何内存。这意味着你不能从该线程访问变量或通过调用setState来更新你的UI。

我们来看一个简单的Isolate的例子，以及如何与主线程通信和共享数据以更新UI：

```
loadData() async {
  ReceivePort receivePort = new ReceivePort();
  await Isolate.spawn(dataLoader, receivePort.sendPort);

  // The 'echo' isolate sends it's SendPort as the first message
  SendPort sendPort = await receivePort.first;

  List msg = await sendReceive(sendPort, "https://jsonplaceholder.typicode.com/posts");

  setState(() {
    widgets = msg;
  });
}

// the entry point for the isolate
static dataLoader(SendPort sendPort) async {
  // Open the ReceivePort for incoming messages.
  ReceivePort port = new ReceivePort();

  // Notify any other isolates what port this isolate listens to.
  sendPort.send(port.sendPort);

  await for (var msg in port) {
    String data = msg[0];
    SendPort replyTo = msg[1];

    String dataURL = data;
    http.Response response = await http.get(dataURL);
    // Lots of JSON to parse
    replyTo.send(JSON.decode(response.body));
  }
}

Future sendReceive(SendPort port, msg) {
  ReceivePort response = new ReceivePort();
  port.send([msg, response.sendPort]);
  return response.first;
}
```

“dataLoader”是在它自己的独立执行线程中运行的隔离区，您可以在其中执行CPU密集型任务，例如解析大于1万的JSON或执行计算密集型数学计算。

下面是一个可以运行的完整示例：


```

import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:async';
import 'dart:isolate';

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  List widgets = [];

  @override
  void initState() {
    super.initState();
    loadData();
  }

  showLoadingDialog() {
    if (widgets.length == 0) {
      return true;
    }

    return false;
  }

  getBody() {
    if (showLoadingDialog()) {
      return getProgressDialog();
    } else {
      return getListView();
    }
  }
}

```

```

    }

    getProgressDialog() {
        return new Center(child: new CircularProgressIndicator());
    }

    @override
    Widget build(BuildContext context) {
        return new Scaffold(
            appBar: new AppBar(
                title: new Text("Sample App"),
            ),
            body: getBody());
    }

    ListView getListView() => new ListView.builder(
        itemCount: widgets.length,
        itemBuilder: (BuildContext context, int position) {
            return getRow(position);
        });

    Widget getRow(int i) {
        return new Padding(padding: new EdgeInsets.all(10.0), child: new Text("Row ${widgets[i]}["title"]"));
    }

    loadData() async {
        ReceivePort receivePort = new ReceivePort();
        await Isolate.spawn(dataLoader, receivePort.sendPort);

        // The 'echo' isolate sends it's SendPort as the first message
        SendPort sendPort = await receivePort.first;

        List msg = await sendReceive(sendPort, "https://jsonplaceholder.typicode.com/posts");

        setState(() {
            widgets = msg;
        });
    }

    // the entry point for the isolate
    static dataLoader(SendPort sendPort) async {
        // Open the ReceivePort for incoming messages.
        ReceivePort port = new ReceivePort();

        // Notify any other isolates what port this isolate listens to.
        sendPort.send(port.sendPort);

        await for (var msg in port) {
            String data = msg[0];
            SendPort replyTo = msg[1];

            String dataURL = data;
            http.Response response = await http.get(dataURL);

```

```
// Lots of JSON to parse
replyTo.send(JSON.decode(response.body));
}
}

Future sendReceive(SendPort port, msg) {
  ReceivePort response = new ReceivePort();
  port.send([msg, response.sendPort]);
  return response.first;
}
}
```

OkHttp在Flutter中等价于什么

当使用受欢迎的“http”package时，Flutter进行网络信非常简单。

虽然“http” package 没有实现OkHttp的所有功能，但“http” package 抽象出了许多常用的API，可以简单有效的发起网络请求。

<https://pub.dartlang.org/packages/http>

您可以通过在pubspec.yaml中添加依赖项来使用它

dependencies:

```
...
http: '>=0.11.3+12'
```

然后就可以进行网络调用，例如请求GitHub上的这个JSON GIST：

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
[...]

loadData() async {
  String dataURL = "https://jsonplaceholder.typicode.com/posts";
  http.Response response = await http.get(dataURL);
  setState(() {
    widgets = JSON.decode(response.body);
  });
}
```

一旦获得结果后，您可以通过调用setState来告诉Flutter更新其状态，setState将使用网络调用的结果更新您的UI。

如何在Flutter中显示进度指示器

在Android中，当您执行耗时任务时，通常会显示进度指示器。

在Flutter中，这可以通过渲染Progress Indicator widget来实现。您可以通过编程方式显示Progress Indicator，通过布尔值通知Flutter在耗时任务发起之前更新其状态。

在下面的例子中，我们将build函数分解为三个不同的函数。如果showLoadingDialog为true（当widgets.length == 0时），那么我们展示ProgressIndicator，否则我们将展

示包含所有数据的ListView。

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  List widgets = [];

  @override
  void initState() {
    super.initState();
    loadData();
  }

  showLoadingDialog() {
    if (widgets.length == 0) {
      return true;
    }

    return false;
  }

  getBody() {
    if (showLoadingDialog()) {
      return getProgressDialog();
    } else {
      return getListView();
    }
  }
}
```

```

getProgressDialog() {
  return new Center(child: new CircularProgressIndicator());
}

@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Sample App"),
    ),
    body: getBody();
  );
}

ListView getListView() => new ListView.builder(
  itemCount: widgets.length,
  itemBuilder: (BuildContext context, int position) {
    return getRow(position);
  });

Widget getRow(int i) {
  return new Padding(padding: new EdgeInsets.all(10.0), child: new Text("Row ${widgets[i]}["title"]"));
}

loadData() async {
  String dataURL = "https://jsonplaceholder.typicode.com/posts";
  http.Response response = await http.get(dataURL);
  setState(() {
    widgets = JSON.decode(response.body);
  });
}

```

项目结构和资源

在哪里存储分辨率相关的图片文件？ HDPI/XXHDPI

Flutter遵循像iOS这样简单的3种分辨率格式：1x, 2x, and 3x.

创建一个名为images的文件夹，并为每个图像文件生成一个@2x和@3x文件，并将它们放置在如下这样的文件夹中

- .../my_icon.png
- .../2.0x/my_icon.png
- .../3.0x/my_icon.png

然后，您需要在pubspec.yaml文件中声明这些图片

```

assets:
- images/a_dot_burr.jpeg
- images/a_dot_ham.jpeg

```

然后您可以使用AssetImage访问您的图像

```
return new AssetImage("images/a_dot_burr.jpeg");
```

在哪里存储字符串？ 如何存储不同的语言

目前，最好的做法是创建一个名为Strings的类

```
class Strings{  
  static String welcomeMessage = "Welcome To Flutter";  
}
```

然后在你的代码中，你可以像访问你的字符串一样：

```
new Text(Strings.welcomeMessage)
```

Flutter对Android的可访问性提供了基本的支持，虽然这个功能正在进行中。

鼓励Flutter开发者使用[intl package](#) 进行国际化和本地化

Android Gradle vs Flutter pubspec.yaml

在Android中，您可以在Gradle文件来添加依赖项。

在Flutter中，虽然在Flutter项目中的Android文件夹下有Gradle文件，但只有在添加平台相关所需的依赖关系时才使用这些文件。 否则，应该使用pubspec.yaml声明用于Flutter的外部依赖项。

发现好的flutter packages的一个好地方 [Pub](#)

Activities 和 Fragments

Activity和Fragment 在Flutter中等价于什么

在Android中，Activity代表用户可以完成的一项重点工作。Fragment代表了一种模块化代码的方式，可以为大屏幕设备构建更复杂的用户界面，可以在小屏幕和大屏幕之间自动调整UI。 在Flutter中，这两个概念都等同于Widget。

如何监听Android Activity生命周期事件

在Android中，您可以覆盖Activity的方法来捕获Activity的生命周期回调。

在Flutter中您可以通过挂接到WidgetsBinding观察并监听

didChangeAppLifecycleState更改事件来监听生命周期事件

您可以监听到的生命周期事件是

- resumed - 应用程序可见并响应用户输入。这是来自Android的onResume
- inactive - 应用程序处于非活动状态，并且未接收用户输入。此事件在Android上未使用，仅适用于iOS
- paused - 应用程序当前对用户不可见，不响应用户输入，并在后台运行。这是来自Android的暂停
- suspending - 该应用程序将暂时中止。这在iOS上未使用

```

import 'package:flutter/widgets.dart';

class LifecycleWatcher extends StatefulWidget {
  @override
  _LifecycleWatcherState createState() => new _LifecycleWatcherState();
}

class _LifecycleWatcherState extends State<LifecycleWatcher> with WidgetsBindingObserver {
  AppLifecycleState _lastLifecycleState;

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
  }

  @override
  void dispose() {
    WidgetsBinding.instance.removeObserver(this);
    super.dispose();
  }

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    setState(() {
      _lastLifecycleState = state;
    });
  }

  @override
  Widget build(BuildContext context) {
    if (_lastLifecycleState == null)
      return new Text('This widget has not observed any lifecycle changes.', textDirection:
TextDirection.ltr);
    return new Text('The most recent lifecycle state this widget observed was: $_lastLifecycleState.',
      textDirection: TextDirection.ltr);
  }
}

void main() {
  runApp(new Center(child: new LifecycleWatcher()));
}

```

Layouts

LinearLayout在Flutter中相当于什么

在Android中，使用LinearLayout来使您的控件呈水平或垂直排列。在Flutter中，您可以使用Row或Column来实现相同的结果。

```

@override
Widget build(BuildContext context) {
  return new Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[

```

```

        new Text(' Row One'),
        new Text(' Row Two'),
        new Text(' Row Three'),
        new Text(' Row Four'),
    ],
);
}

@override
Widget build(BuildContext context) {
    return new Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            new Text(' Column One'),
            new Text(' Column Two'),
            new Text(' Column Three'),
            new Text(' Column Four'),
        ],
    );
}

```

RelativeLayout在Flutter中等价于什么

RelativeLayout用于使widget相对于彼此位置排列。在Flutter中，有几种方法可以实现相同的结果

您可以通过使用Column、Row和Stack的组合来实现RelativeLayout的效果。您可以为widget构造函数指定相对于父组件的布局规则。

一个在Flutter中构建RelativeLayout的好例子，请参考在StackOverflow上

<https://stackoverflow.com/questions/44396075/equivalent-of-relativelayout-in-flutter>

ScrollView在Flutter中等价于什么

在Android中，ScrollView允许您包含一个子控件，以便在用户设备的屏幕比控件内容小的情况下，使它们可以滚动。

在Flutter中，最简单的方法是使用ListView。但在Flutter中，一个ListView既是一个ScrollView，也是一个Android ListView。

```

@override
Widget build(BuildContext context) {
    return new ListView(
        children: <Widget>[
            new Text(' Row One'),
            new Text(' Row Two'),
            new Text(' Row Three'),
            new Text(' Row Four'),
        ],
    );
}

```

手势检测和触摸事件处理

如何将一个onClick监听器添加到Flutter中的widget

在Android中，您可以通过调用方法`setOnClickListener`将OnClick绑定到按钮等view上。
在Flutter中，添加触摸监听器有两种方法：

1. 如果Widget支持事件监听，则可以将一个函数传递给它并进行处理。例如，`RaisedButton`有一个`onPressed`参数

```
@override
Widget build(BuildContext context) {
  return new RaisedButton(
    onPressed: () {
      print("click");
    },
    child: new Text("Button"));
}
```

1. 如果Widget不支持事件监听，则可以将该Widget包装到`GestureDetector`中，并将处理函数传递给`onTap`参数

```
class SampleApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      body: new Center(
        child: new GestureDetector(
          child: new FlutterLogo(
            size: 200.0,
          ),
          onTap: () {
            print("tap");
          },
        ),
      ),
    );
  }
}
```

如何处理widget上的其他手势

使用`GestureDetector`，可以监听多种手势，例如

- Tap
 - `onTapDown`
 - `onTapUp`
 - `onTap`
 - `onTapCancel`
- Double tap

- `onDoubleTap` 用户快速连续两次在同一位置轻敲屏幕.
- 长按
 - `onLongPress`
- 垂直拖动
 - `onVerticalDragStart`
 - `onVerticalDragUpdate`
 - `onVerticalDragEnd`
- 水平拖拽
 - `onHorizontalDragStart`
 - `onHorizontalDragUpdate`
 - `onHorizontalDragEnd`

For example here is a `GestureDetector` for double tap on the FlutterLogo that will make it rotate 例如，这里是一个`GestureDetector`，用于监听FlutterLogo的双击事件，双击时使其旋转

```
AnimationController controller;
CurvedAnimation curve;
```

```
@override
void initState() {
  controller = new AnimationController(duration: const Duration(milliseconds: 2000), vsync: this);
  curve = new CurvedAnimation(parent: controller, curve: Curves.easeIn);
}
```

```
class SampleApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      body: new Center(
        child: new GestureDetector(
          child: new RotationTransition(
            turns: curve,
            child: new FlutterLogo(
              size: 200.0,
            )),
          onTap: () {
            if (controller.isCompleted) {
              controller.reverse();
            } else {
              controller.forward();
            }
          }
        )
      )
    );
  }
}
```

```

    },
  ),
));
}
}
}

```

Listview & Adapter

ListView在Flutter中相当于什么

在Flutter中，ListView就是一个ListView！

在Android ListView中，您可以创建一个适配器，然后您可以将它传递给ListView，该适配器将使用适配器返回的内容来展示每一行。然而，你必须确保在合适的时机回收行，否则，你会得到各种疯狂的视觉和内存问题。

在Flutter中，由于Flutter的不可变的widget模型，将一个Widgets列表传递给ListView，而Flutter将负责确保它们快速平滑地滚动。

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

```
void main() {
  runApp(new SampleApp());
}
```

```
class SampleApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}
```

```
class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}
```

```
class _SampleAppPageState extends State<SampleAppPage> {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
    ),
  }
}
```

```

        body: new ListView(children: _getListData()),
      );
    }

    _getListData() {
      List<Widget> widgets = [];
      for (int i = 0; i < 100; i++) {
        widgets.add(new Padding(padding: new EdgeInsets.all(10.0), child: new Text("Row $i")));
      }
      return widgets;
    }
  }
}

```

怎么知道哪个列表项被点击

在Android中，ListView有一个方法'onItemClickListener'来确定哪个列表项被点击。

Flutter中可以更轻松地通过您传入的处理回调来进行操作。

```

import 'package:flutter/material.dart';

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new ListView(children: _getListData()),
    );
  }
}

```

```

_getListData() {
  List<Widget> widgets = [];
  for (int i = 0; i < 100; i++) {
    widgets.add(new GestureDetector(
      child: new Padding(
        padding: new EdgeInsets.all(10.0),
        child: new Text("Row $i")),
      onTap: () {
        print('row tapped');
      },
    ));
  }
  return widgets;
}
}

```

如何动态更新ListView

需要更新适配器并调用notifyDataSetChanged。在Flutter中，如果setState()中更新widget列表，您会发现没有变化，这是因为当setState被调用时，Flutter渲染引擎会遍历所有的widget以查看它们是否已经改变。当遍历到你的ListView时，它会做一个==运算，以查看两个ListView是否相同，因为没有任何改变，因此没有更新数据。要更新您的ListView，然后在setState中创建一个新的List()并将所有旧数据复制到新列表中。这是实现更新的简单方法（译者语：此时状态改变，ListView被重新构建）

```

import 'package:flutter/material.dart';

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Sample App',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new SampleAppPage(),
    );
  }
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

```

```

}

class _SampleAppPageState extends State<SampleAppPage> {
  List widgets = [];

  @override
  void initState() {
    super.initState();
    for (int i = 0; i < 100; i++) {
      widgets.add(getRow(i));
    }
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new ListView(children: widgets),
    );
  }

  Widget getRow(int i) {
    return new GestureDetector(
      child: new Padding(
        padding: new EdgeInsets.all(10.0),
        child: new Text("Row $i"),
      ),
      onTap: () {
        setState(() {
          widgets = new List.from(widgets);
          widgets.add(getRow(widgets.length + 1));
          print('row $i');
        });
      },
    );
  }
}

```

然而，推荐的方法是使用`ListView.Builder`。当您拥有动态列表或包含大量数据的列表时，此方法非常有用。这实际上相当于在Android上使用`RecyclerView`，它会自动为您回收列表元素：

```

import 'package:flutter/material.dart';

void main() {
  runApp(new SampleApp());
}

class SampleApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(

```

```

        title: 'Sample App',
        theme: new ThemeData(
          primarySwatch: Colors.blue,
        ),
        home: new SampleAppPage(),
      );
    }
  }

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  List widgets = [];

  @override
  void initState() {
    super.initState();
    for (int i = 0; i < 100; i++) {
      widgets.add(getRow(i));
    }
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new ListView.builder(
        itemCount: widgets.length,
        itemBuilder: (BuildContext context, int position) {
          return getRow(position);
        }
      ));
  }

  Widget getRow(int i) {
    return new GestureDetector(
      child: new Padding(
        padding: new EdgeInsets.all(10.0),
        child: new Text("Row $i"),
      ),
      onTap: () {
        setState(() {
          widgets.add(getRow(widgets.length + 1));
          print('row $i');
        });
      },
    );
  }
}

```

我们不是创建一个“新的ListView”，而是创建一个新的ListView.builder，它接受两个参数，即列表的初始长度和一个ItemBuilder函数。

ItemBuilder函数非常类似于Android适配器中的getView函数，它需要一个位置并返回要为该位置渲染的行。

最后，但最重要的是，如果您注意到onTap函数，在里面，我们不会再重新创建列表，而只是添加新元素到列表。

使用 Text

如何在 Text widget上设置自定义字体

在Android SDK（从Android O开始）中，创建一个Font资源文件并将其传递到TextView的FontFamily参数中。

在Flutter中，首先你需要把你的字体文件放在项目文件夹中（最好的做法是创建一个名为assets的文件夹）

接下来在pubspec.yaml文件中，声明字体：

```
fonts:
  - family: MyCustomFont
    fonts:
      - asset: fonts/MyCustomFont.ttf
      - style: italic
```

最后，将字体应用到Text widget:

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text("Sample App"),
    ),
    body: new Center(
      child: new Text(
        'This is a custom font text',
        style: new TextStyle(fontFamily: 'MyCustomFont'),
      ),
    ),
  );
}
```

如何在Text上定义样式

除了自定义字体，您还可在Text上自定义很多不同的样式

Text的样式参数需要一个TextStyle对象，您可以在其中自定义许多参数，如

- color
- decoration
- decorationColor

- decorationStyle
- fontFamily
- fontSize
- fontStyle
- fontWeight
- hashCode
- height
- inherit
- letterSpacing
- textBaseline
- wordSpacing

表单输入

Input的” hint” 在flutter中相当于什么

在Flutter中，您可以通过向Text Widget的装饰构造函数参数添加InputDecoration对象，轻松地输入框显示占位符文本

```
body: new Center(  
  child: new TextField(  
    decoration: new InputDecoration(hintText: "This is a hint"),  
  )  
)
```

如何显示验证错误

就像您如何使用“hint”一样，您可以将InputDecoration对象传递给Text的装饰构造函数。

但是，您不希望首先显示错误，并且通常会在用户输入一些无效数据时显示该错误。这可以通过更新状态并传递一个新的InputDecoration对象来完成

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(new SampleApp());  
}  
  
class SampleApp extends StatelessWidget {  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return new MaterialApp(  
      title: 'Sample App',  
      theme: new ThemeData(  
        primarySwatch: Colors.blue,
```

```

    ),
    home: new SampleAppPage(),
  );
}

class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key key}) : super(key: key);

  @override
  _SampleAppPageState createState() => new _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  String _errorText;

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Sample App"),
      ),
      body: new Center(
        child: new TextField(
          onSubmitted: (String text) {
            setState(() {
              if (!isEmail(text)) {
                _errorText = 'Error: This is not an email';
              } else {
                _errorText = null;
              }
            });
          },
          decoration: new InputDecoration(hintText: "This is a hint", errorText: _getErrorText()),
        ),
      ),
    );
  }

  _getErrorText() {
    return _errorText;
  }

  bool isEmail(String em) {
    String emailRegexp =
      r'^((^[<>() []\.\,;:\s@\"']+(\.[^<>() []\.\,;:\s@\"']+)*|(\\".+\"))@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\)|((\b[a-zA-Z-0-9]+\.)+[a-zA-Z]{2,}))$';

    RegExp regExp = new RegExp(p);

    return regExp.hasMatch(em);
  }
}

```

Flutter 插件

如何使用 GPS sensor

要访问GPS传感器，您可以使用社区插件 <https://pub.dartlang.org/packages/location>

如何访问相机

访问相机的流行社区插件是 https://pub.dartlang.org/packages/image_picker

如何使用Facebook登陆

要访问Facebook Connect功能，您可以使

用 https://pub.dartlang.org/packages/flutter_facebook_connect .

如何构建自定义集成Native功能

如果有Flutter或其社区插件缺失的平台特定功能，那么您可以自己按照以下教程构建 [开发packages](#) .

简而言之，Flutter的插件架构就像在Android中使用Event bus一样：您可以发出消息并让接收者进行处理并将结果返回给您，在这种情况下，接收者将是iOS或Android。

如何在我的Flutter应用程序中使用NDK

自定义插件首先会与Android应用程序通信，您可以在其中调用native标记的函数。一旦Native完成了相应操作，就可以将响应消息发回给Flutter并呈现结果。

主题

如何构建Material主题风格的app

Flutter很好的实现了一个美丽的Material Design，它会满足很多样式和主题的需求。

与Android中使用XML声明主题不同，在Flutter中，您可以通过顶层widget声明主题。

MaterialApp是一个方便的widget，它包装了许多Material Design应用通常需要的widget，它通过添加Material特定功能构建在WidgetsApp上。

如果你不想使用Material Components，那么你可以声明一个顶级widget-

WidgetsApp，它是一个便利的类，它包装了许多应用程序通常需要的widget

要自定义Material Components的颜色和样式，您可以将ThemeData对象传递到

MaterialApp widget中，例如在下面的代码中，您可以看到主色板设置为蓝色，并且所有选择区域的文本颜色都应为红色。

```
class SampleApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return new MaterialApp(  
      title: 'Sample App',  
      theme: new ThemeData(  
        primarySwatch: Colors.blue,
```

```

        textSelectionColor: Colors.red
      ),
      home: new SampleAppPage(),
    );
  }
}

```

数据库和本地存储

如何在Flutter中访问Shared Preferences ?

在Android中，您可以使用SharedPreferences API存储一些键值对

在Flutter中，您可以通过使用插件[Shared_Preferences](#)来访问此功能

这个插件包装了Shared Preferences和NSUserDefaults（与iOS相同）的功能

```

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

void main() {
  runApp(
    new MaterialApp(
      home: new Scaffold(
        body: new Center(
          child: new RaisedButton(
            onPressed: _incrementCounter,
            child: new Text('Increment Counter'),
          ),
        ),
      ),
    ),
  );
}

_incrementCounter() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  int counter = (prefs.getInt('counter') ?? 0) + 1;
  print('Pressed $counter times. ');
  prefs.setInt('counter', counter);
}

```

如何在Flutter中访问SQLite

在Android中，您可以使用SQLite存储，通过SQL查询的结构化数据。

在Flutter中，您可以使用[SQFlite](#)插件来访问SQFlite此功能