

# Flutter 与 Native 通信： PlatformChannel

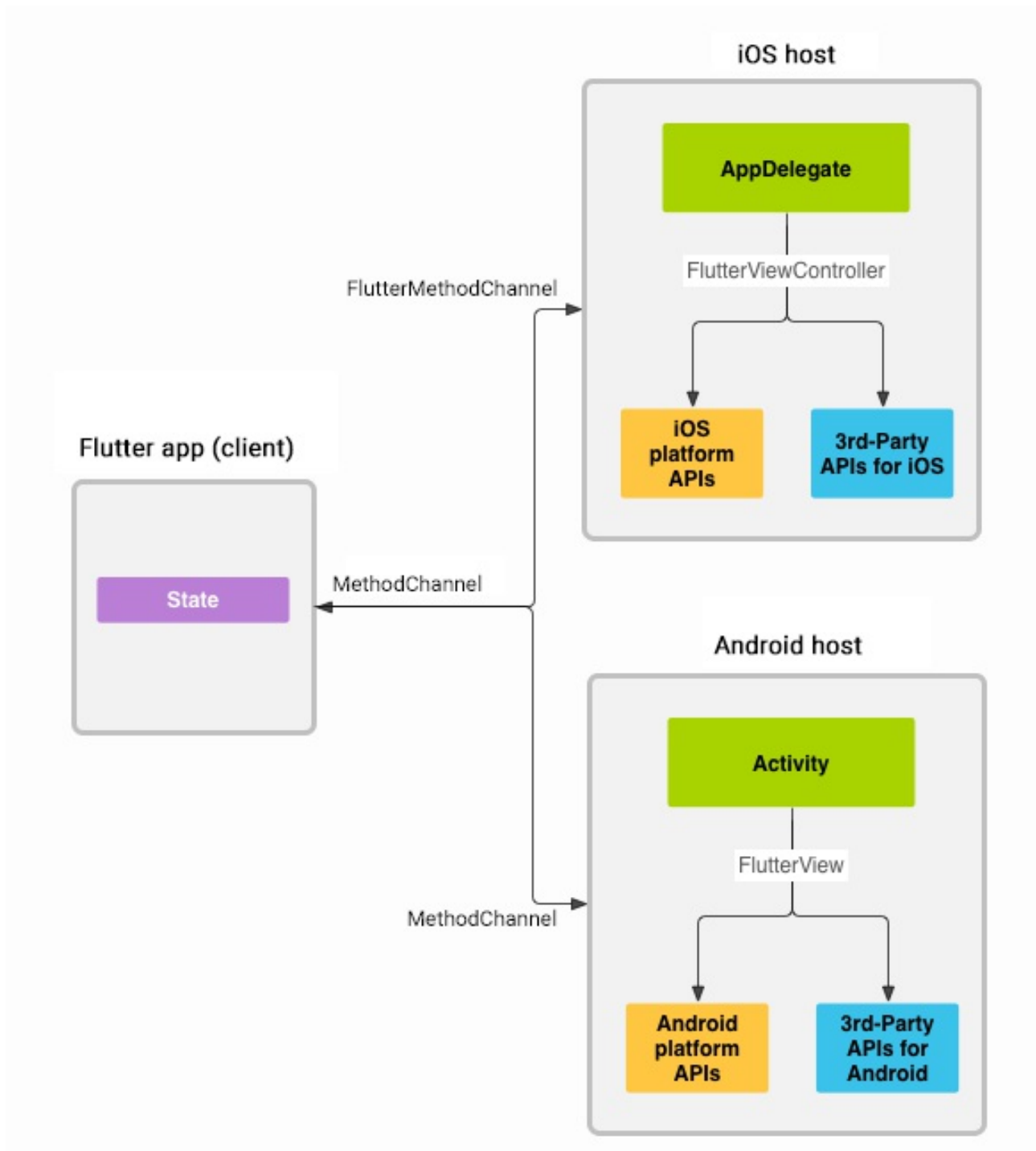
本节介绍 PlatformChannel 的使用。

## PlatformChannel 介绍

PlatformChannel 用于 Flutter 与 Native(Android、iOS) 之间的消息传递。

不仅可以方便的将 Native 的功能拓展给 Flutter 使用，而且也能将 Flutter 的功能扩展给 Native 使用。

## PlatformChannel 的架构图



上图是 PlatformChannel 的架构图，Flutter 与 Native 之间采用的是消息传递的模式，一个是 Client，一个是 Host：

- Client 通过 PlatformChannel 向 Host 发送消息。
- Host 监听 PlatformChannel，接受消息，然后将响应结果发送给 Client。这样就完成了 Flutter 与 Native 之间的消息传递。
- 而且 Flutter 与 Native 之间的消息传递都是异步的，所以不会

阻塞 UI。

而且由于 PlatformChannel 是双工的，所以 Flutter 和 Native 可以互相做 Client 和 Host。

## PlatformChannel 的种类

有三种 PlatformChannel 种类，适用于不同的场景：

### 1. MethodChannel

以方法的模式使用 PlatformChannel

### 2. EventChannel

以事件的模式使用 PlatformChannel

### 3. BasicMessageChannel

可以在 BasicMessageChannel 上方便的进行自定义扩展，主要用于个性化的扩展。

MethodChannel 和 EventChannel 这两个已经封装好了使用方式，是我们最常用的，我们一般很少使用 BasicMessageChannel。后面我们会着重讲一下 MethodChannel 和 EventChannel 的使用。

## codec

又因为同时要在 Flutter、Android、iOS 这三个不同的平台上传递消息，就需要可以在三个平台上都能处理的数据，所以需要对传递的消息进行编解码，而且只能支持一些基本数据类型的传递，传递过程如下：

- 首先将 Client 端平台的数据进行编码，然后通过

## PlatformChannel 发送

- Host 端接受到消息后，将消息解码成 Host 端平台的数据类型
- Host 端向 Client 端发送回应的消息，同样要先进行编码后，才能发送
- Client 端接收到消息后，将消息解码成 Client 端平台的数据类型

在这个过程中，编解码的方法就叫 codec。每一个 PlatformChannel 都有一个 codec。

下面是每种 PlatformChannel 使用的 codec：

MethodChannel	EventChannel	BasicMessageChannel
StandardMethodCodec	StandardMethodCodec	BasicMessageCodec

上面的四个 codec 都行

PlatformChannel 的 codec 主要分为两类：

### 1. MethodCodec

MethodCodec 是一个接口，它的实现类是如下的两个：

- StandardMethodCodec：可以编解码 PlatformChannel 支持的所有类型的数据
- JSONMethodCodec：只可以编解码 JSON 类型的数据

### 2. MessageCodec

MessageCodec 是一个接口，它的实现类是如下的四个：

- StandardMessageCodec：可以编解码 PlatformChannel 支持的所有类型的数据
- BinaryCodec：只可以编解码 byte 类型的数据

- StringCodec: 只可以编解码 String 类型的数据
- JSONMessageCodec: 只可以编解码 JSON 类型的数据

## PlatformChannel 支持的数据类型

下图是PlatformChannel支持的数据类型，以及在 Flutter、Android、iOS 三个平台上转化的对应关系：

Flutter	Android	iOS
null	null	nil (NSNull when nested)
bool	java.lang.Boolean	NSNumber numberWithBool:
int	java.lang.Integer	NSNumber numberWithInt:
int, if 32 bits not enough	java.lang.Long	NSNumber numberWithLong:
double	java.lang.Double	NSNumber numberWithDouble:
String	java.lang.String	NSString
Uint8List	byte[]	FlutterStandardTypedData typedDataWithBytes:
Int32List	int[]	FlutterStandardTypedData typedDataWithInt32:
Int64List	long[]	FlutterStandardTypedData typedDataWithInt64:
Float64List	double[]	FlutterStandardTypedData typedDataWithFloat64:
List	java.util.ArrayList	NSArray
Map	java.util.HashMap	NSDictionary

## PlatformChannel -- MethodChannel 的使用

本节讲一下 MethodChannel 的使用方法。MethodChannel 既可以让 Flutter 调用 Android，也可以让 Android 调用 Flutter。

这里写两个例子，一个是 Flutter 调用 Android 功能的例子，另一个是 Android 调用 Flutter 功能的例子。

PlatformChannel 涉及到编写 Native 代码，所以除了 VS Code，还需要使用 Android Studio 和 Xcode。

## Flutter 调用 Android 功能

使用 Flutter 弹一个 Android 的 Toast。

为了实现这一样功能，需要使用 Flutter 调用 Android 的功能，那么这里 Flutter 是 Client，Android 是 Host。

需要在 Android 端 上实现监听及功能，然后在 Flutter 端 调用。

**在 Android 端添加监听及实现功能：**

Android 端是 Host，在 Android 的代码里找到 MainActivity.java ,然后在 onCreate() 里使用 MethodChannel 添加监听：

```
@Override
protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    GeneratedPluginRegistrant.registerWith(this);

    new
MethodChannel(getFlutterView(), "samples.flutter.i
o/toast").setMethodCallHandler(new
MethodChannel.MethodCallHandler() {
    @Override
    public void onMethodCall(MethodCall
methodCall, MethodChannel.Result result) {

    }
});
}
```

这里 new MethodChannel 的实例，需要两个参数：

1. 第一个是 BinaryMessenger

这里用的是 FlutterView，因为 FlutterView 实现了 BinaryMessenger 的接口，在 MainActivity 里通过 getFlutterView() 获取

2. 第二个是 String

是 Channel Name，用于标识不同的 MethodChannel，这个值可以随便取，但每个 MethodChannel 都必须是唯一的，一般取值是：包名+"/" + 功能，这个值在 Flutter 中也要用，Flutter 正是通过这个来区分不同的 MethodChannel 的。在 demo 里我取的值是 samples.flutter.io/toast。

创建完 `MethodChannel` 的实例后，还需要添加监听，是通过 `MethodChannel` 的 `setMethodCallHandler` 来添加监听的。

监听监听完后，在看回调里的两个参数：

## 1. `MethodCall`

从 Flutter 端传送来的数据。

`MethodCall` 里有如下的几个方法，用来获取从 Flutter 端传送来的数据：

- `methodCall.method`:

`methodCall` 的属性，获取方法名，用于标识 Flutter 想调用 Android 的哪个方法，这里 `method` 只能是 `String` 类型，这就要求 Flutter 只能传 `String`

- `methodCall.arguments`

`methodCall` 的属性，获取具体的参数，这里 `arguments` 只能是 `Map` 类型或者 `JSONObject` 类型，这就要求 Flutter 只能传这两种类型的数据

- `methodCall.argument(String key)`

`methodCall` 的方法，获取 `Key` 为特定值的参数的值

## 2. `MethodChannel.Result`

Android 向 Flutter 端发送数据

Android 端处理完从 Flutter 端传来的数据后，就需要向 Flutter 端发送数据，使用如下的方法：

- `result.success(Object result)`



如果运行成功，使用这个方法，同时将 result 返回给 Flutter,请注意，result 只能是 PlatformChannel 支持的数据类型

- result.error(String errorCode,String errorMsg,Object errorDetails)

如果运行失败，使用这个方法,带入 errorCode、errorMsg 还有 errorDetails 的参数，errorDetails 只能是 PlatformChannel 支持的数据类型

- result.notImplemented()

如果 Flutter 想调用的方法,Android 没有实现，使用这个方法。会将 notImplemented 返回给 Flutter。

下面是 MainActivity.java 里的代码：

```
package com.kk.flutter_app;

import android.os.Bundle;
import android.widget.Toast;

import io.flutter.app.FlutterActivity;
import io.flutter.plugin.common.MethodCall;
import
io.flutter.plugin.common.MethodChannel;
import
io.flutter.plugins.GeneratedPluginRegistrant;

public class MainActivity extends
FlutterActivity {
    @Override
    protected void onCreate(Bundle
```

```
savedInstanceState) {
    super.onCreate(savedInstanceState);

    GeneratedPluginRegistrant.registerWith(this);

    new
MethodChannel(getFlutterView(), "samples.flutter.i
o/toast").setMethodCallHandler(new
MethodChannel.MethodCallHandler() {
    @Override
    public void onMethodCall(MethodCall
methodCall, MethodChannel.Result result) {
        if("toast".equals(methodCall.method))
{

if(methodCall.hasArgument("content")){
            Toast.makeText(getBaseContext(),
methodCall.argument("content"), Toast.LENGTH_SHORT
).show();

            result.success("success");
        }else {
            result.error("-1", "toast
fail", "content is null");
        }
    }else {
        result.notImplemented();
    }
}
});
}
```

使用 `methodCall.method` 的值来区分调用哪个方法。

如果调用的方法名是 `toast`，则弹 `Toast`。

**在 Flutter 端发送消息：**

Flutter 端是 Client，首先也要创建一个 `MethodChannel` 的实例：

```
static const platformChannel = const  
MethodChannel('samples.flutter.io/toast');
```

创建 `MethodChannel` 实例的时候，要传入 `ChannelName`，`ChannelName` 必须要和 Android 的 `ChannelName` 保持一致。`ChannelName` 用于区分调用哪个方法。

然后调用接口：

```

void showToast(String content) async {
    var arguments = Map();
    arguments['content'] = content;

    try {
        String result = await
platformChannel.invokeMethod('toast', arguments);
        //success
        print('showToast ' + result);
    } on PlatformException catch (e) {
        //error
        print('showToast ' + e.code + e.message
+ e.details);
    } on MissingPluginException catch (e){
        //notImplemented
        print('showToast ' + e.message);
    }
}

```

- `platformChannel.invokeMethod('toast', arguments)` 是调用Android的方法,
- `toast` 是要调用的方法名, 为String类型;
- `arguments` 是参数, 只能是Map或者JSON类型, 这里是Map类型。

为了不阻塞 UI, `PlatformChannel` 使用 `async` 和 `await` 来修饰, 而且必须要用 `try catch`。前面知道有三种返回结果: `success`、`error`、`notImplemented`, 第一个 `catch` 的 `PlatformException` 就是 `error` 的结果, 第二个 `catch` 的 `MissingPluginException` 就是 `notImplemented` 的结果。

Flutter 的代码是:

```

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

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

class MyApp extends StatelessWidget {
  static const platformChannel =
    const
MethodChannel('samples.flutter.io/toast');

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: "Flutter Demo",
      theme: ThemeData(
        primaryColor: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(title:
Text("Platform Channel")),
        body: RaisedButton(
          child: Text("Show Toast"),
          onPressed: () {
            showToast("Flutter Toast");
          })),
    );
  }

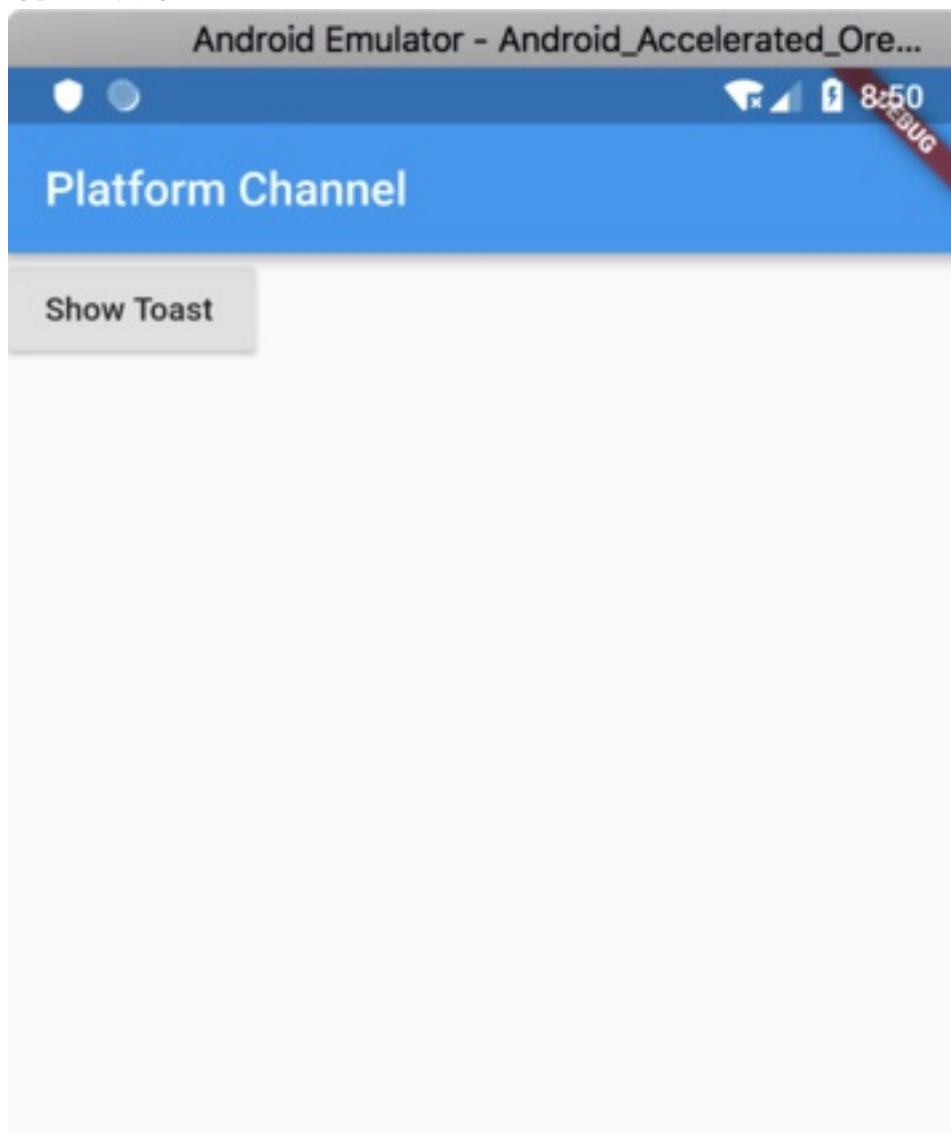
  void showToast(String content) async {
    var arguments = Map();
    arguments['content'] = content;

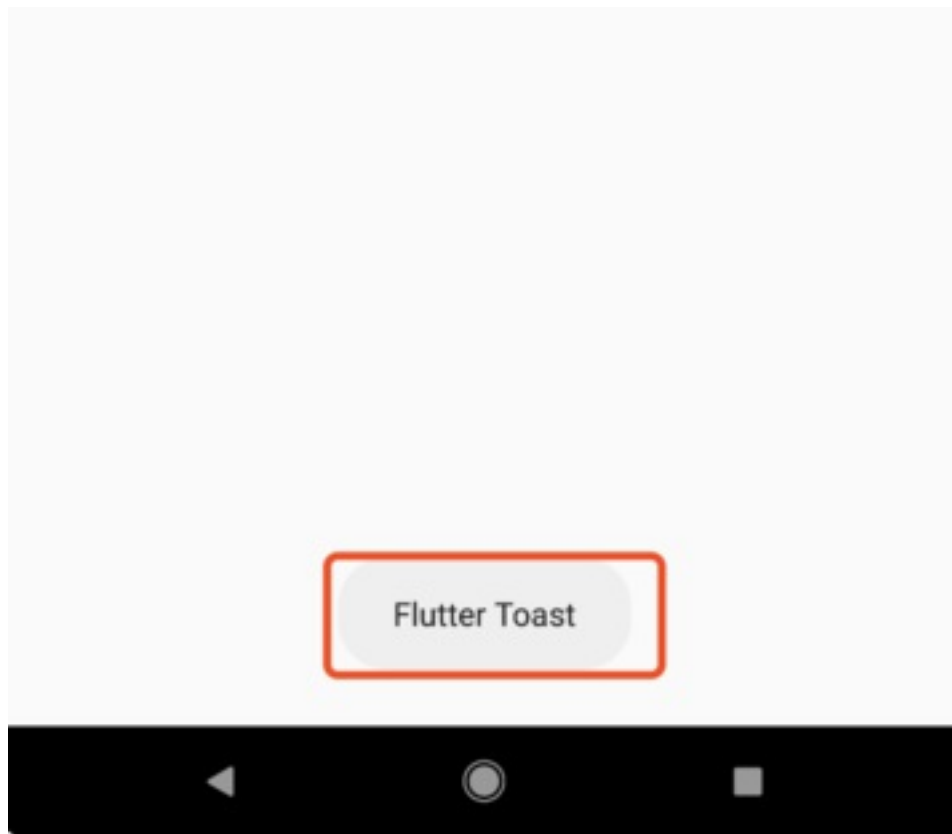
    try {

```

```
String result = await
platformChannel.invokeMethod('toast', arguments);
print('showToast ' + result);
} on PlatformException catch (e) {
    print('showToast ' + e.code + e.message
+ e.details);
} on MissingPluginException catch (e){
    print('showToast ' + e.message);
}
}
```

下面是运行的效果：





## Android 调用 Flutter 功能

Android 发送一段文字到 Flutter，并显示在 Flutter 的 UI 上：原本Flutter UI上显示的是Flutter Message，然后将APP退后台，在切到前台，Android向Flutter发送数据，将UI的显示变为Android Message。

为了实现这一功能，就需要 Android 调用 Flutter 的功能，这时 Android 是 Client，而 Flutter 是 Host。

所以要在 Flutter 上实现监听和功能，然后在 Android 中使用。

在Flutter端添加监听及实现功能：

```
class MyAppState extends State<MyApp> {  
  static const platformChannel =  
    const  
MethodChannel('samples.flutter.io/message');
```

```
String textContent = 'Flutter Message';

@override
void initState() {
    // TODO: implement initState
    super.initState();

platformChannel.setMethodCallHandler((methodCall)
    async {
        switch (methodCall.method) {
            case 'showText':
                String content = await
methodCall.arguments['content'];
                if (content !=null &&
content.isNotEmpty) {
                    setState(() {
                        textContent = content;
                    });
                    //返回成功的结果
                    return 'success';
                } else {
                    //返回失败的结果
                    throw PlatformException(
                        code: '-1',
                        message: 'showText fail',
                        details: 'content is null');
                }
                break;
            default:
                //返回方法未实现的结果
                throw MissingPluginException();
        }
    }
}
```



```
    });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    // TODO: implement build  
    return ...  
  }  
}
```

首先创建 MethodChannel 的实例：

```
static const platformChannel = const  
MethodChannel('samples.flutter.io/message');
```

根据前面讲的 StatefulWidget 的生命周期，我们在 initState 里添加 MethodChannel 的监听，就是通过 setMethodCallHandler，为 MethodChannel 添加回调，回调的方式添加了 async，因此不会阻塞 UI：

```
platformChannel.setMethodCallHandler((methodCall)
  async {
    switch (methodCall.method) {
      case 'showText':
        String content = await
methodCall.arguments['content'];
        if (content !=null &&
content.isNotEmpty) {
          ...
          return 'success';
        } else {
          throw PlatformException(
            code: '-1',
            message: 'showText fail',
            details: 'content is null');
        }
        break;
      default:
        throw MissingPluginException();
    }
  });
```

回调里有个参数 `methodCall`，和 Android 里的 `methodCall` 有相同的属性：

- `methodCall.method`：获取方法名，String 类型
- `methodCall.arguments`：获取具体的参数，arguments 只能是 Map 类型或者 JSONObject 类型

在返回结果的时候：

- 如果运行成功，则直接 `return Object`
- 如果允许失败，则 `throw` 一个 `PlatformException`，表示运行

失败

- 如果方法没有实现，则 throw 一个 `MissingPluginException`

Flutter部分的全部代码为：

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

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

class MyApp extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return MyAppState();
  }
}

class MyAppState extends State<MyApp> {
  static const platformChannel =
    const
MethodChannel('samples.flutter.io/message');

  String textContent = 'Flutter Message';

  @override
  void initState() {
    // TODO: implement initState
    super.initState();

    platformChannel.setMethodCallHandler((methodCall)
    async {
      switch (methodCall.method) {
```

```
        case 'showText':
            String content = await
methodCall.arguments['content'];
            if (content !=null &&
content.isNotEmpty) {
                setState(() {
                    textContent = content;
                });
                return 'success';
            } else {
                throw PlatformException(
                    code: '-1',
                    message: 'showText fail',
                    details: 'content is null');
            }
            break;
        default:
            throw MissingPluginException();
    }
});
}
```

```
@override
Widget build(BuildContext context) {
    // TODO: implement build
    return MaterialApp(
        title: "Flutter Demo",
        theme: ThemeData(
            primaryColor: Colors.blue,
        ),
        home: Scaffold(
            appBar: AppBar(
                title: Text('Platform Channel'),
            ),
        ),
    );
}
```

```
        ),  
        body: Text(textContent),  
      ),  
    );  
  }  
}
```

并且实现了在 Flutter 端更新 UI 的功能。

## 在 Android 端发送数据

发送数据我写在 onResume 里面，当 APP 退后台，在切到前台时就会触发。

Android 端的代码还是在 MainActivity.java 里：

```
package com.kk.flutter_app;  
  
import android.os.Bundle;  
import android.util.Log;  
  
import java.util.HashMap;  
import java.util.Map;  
  
import io.flutter.app.FlutterActivity;  
import io.flutter.plugin.common.MethodChannel;  
import  
io.flutter.plugins.GeneratedPluginRegistrant;  
  
public class MainActivity extends FlutterActivity  
{
```

```
MethodChannel methodChannel;
@Override
protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    GeneratedPluginRegistrant.registerWith(this);

    methodChannel = new
MethodChannel(getFlutterView(), "samples.flutter.i
o/message");

}

@Override
protected void onResume() {
    super.onResume();
    Map map = new HashMap();
    map.put("content", "Android Message");
    methodChannel.invokeMethod("showText", map,
new MethodChannel.Result() {
        @Override
        public void success(Object o) {
            Log.d("MainActivity", (String)o);
        }

        @Override
        public void error(String errorCode, String
errorMsg, Object errorDetail) {

Log.d("MainActivity", "errorCode:" + errorCode + "
errorMsg:" + errorMsg + " errorDetail:" +
(String)errorDetail);
        }
    }
}
```

```
        @Override
        public void notImplemented() {
            Log.d("MainActivity", "notImplemented");
        }
    });
}
```

首先也是先创建 MethodChannel 的实例：

```
methodChannel = new
MethodChannel(getFlutterView(), "samples.flutter.i
o/message");
```

然后向 Flutter 发送消息，告诉 Flutter 要调用哪个方法：

```
methodChannel.invokeMethod("showText", map, new
MethodChannel.Result() {
    @Override
    public void success(Object o) {
        Log.d("MainActivity", (String)o);
    }

    @Override
    public void error(String errorCode, String
errorMsg, Object errorDetail) {

Log.d("MainActivity", "errorCode:"+errorCode+"
errorMsg:"+errorMsg+" errorDetail:"+
(String)errorDetail);
    }

    @Override
    public void notImplemented() {
        Log.d("MainActivity", "notImplemented");
    }
});
}
```

在 MethodChannel.Result() 回调里有三个方法：

- success：表示调用成功
- error：表示调用失败
- notImplemented：表示没有此方法

刚运行 APP 的效果：

退后台，又切到前台的效果：



Android Emulator - Android\_Accelerated\_Ore...



10:01

2018 AUG

## Platform Channel

Android Message



# PlatformChannel -- EventChannel 的使用

本节讲一下 EventChannel 使用的使用方法。EventChannel 只能从 Android 向 Flutter 发送数据，它的使用方式很像我们熟悉的 Event 的使用方式。

这里使用 EventChannel 实现一个和上节相同的例子：

Android 发送一段文字到 Flutter，并显示在 Flutter 的 UI 上：原本Flutter UI上显示的是Flutter Message，然后将APP退后台，在切到前台，Android向Flutter发送数据，将UI的显示变为Android Message。

## 在 Flutter 端添加监听并实现功能

首先是创建 EventChannel：

```
static const eventChannel =  
    const  
    EventChannel('samples.flutter.io/event');
```

创建方式和 MethodChannel 是一样的，都需要传入一个字符串。

然后在 initState() 里使用 eventChannel 的 listen 方法添加监听：

```
eventChannel.receiveBroadcastStream().listen(_onL  
isten,onError: _onError,onDone:  
_onDone,cancelOnError: false);
```

listen 方法定义为：

```
StreamSubscription<T> listen(void onData(T
event),
    {Function onError, void onDone(), bool
cancelOnError});
```

可以看到 listen 方法有如下几个参数：

参数名字	参数类型	意义	必选 or 可选
onData	void onData(T event)	EventChannel 正常收到数据的事件	必选
onError	Function	EventChannel 收到错误的事件	可选
onDone	void onDone()	EventChannel 收到结束监听的事件	可选
cancelOnError	bool	当出错的时候是不是自动取消 默认为false	可选

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

main() {
  runApp(new MyApp());
}

class MyApp extends StatefulWidget{
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return MyAppState();
  }
}
```

```
}  
  
class MyAppState extends State<MyApp> {  
  static const eventChannel =  
    const  
EventChannel('samples.flutter.io/event');  
  
  String textContent = 'Flutter Message';  
  
  @override  
  void initState() {  
    // TODO: implement initState  
    super.initState();  
  
eventChannel.receiveBroadcastStream().listen(_onL  
isten,onError: _onError,onDone:  
_onDone,cancelOnError: false);  
  }  
  
  void _onListen(dynamic data){  
    setState(() {  
      textContent = data;  
    });  
  }  
  
  void _onError(){  
    setState(() {  
      textContent = 'EventChannel error';  
    });  
  }  
  
  void _onDone(){  
    setState(() {
```

```

        textContent = 'EventChannel done';
    });
}

@override
Widget build(BuildContext context) {
    // TODO: implement build
    return MaterialApp(
        title: "Flutter Demo",
        theme: ThemeData(
            primaryColor: Colors.blue,
        ),
        home: Scaffold(
            appBar: AppBar(
                title: Text('Platform Channel --
EventChannel'),
            ),
            body: Text(textContent),
        ),
    );
}
}

```

- 如果运行正常，就会在 `_onListen` 收到正常数据的事件
- 如果运行出现 `error`，就会在 `_onError` 收到错误的事件
- 如果运行结束，就会在 `_onDone` 收到结束监听的事件

## 在 Android 端发送数据

```

private EventChannel.EventSink mEventSink;
@Override
protected void onCreate(Bundle
savedInstanceState) {

```

```

        super.onCreate(savedInstanceState);
        GeneratedPluginRegistrant.registerWith(this);

        new
EventChannel(getFlutterView(), "samples.flutter.io
/event").setStreamHandler(new
EventChannel.StreamHandler() {
    @Override
    public void onListen(Object o,
EventChannel.EventSink eventSink) {
        mEventSink = eventSink;
    }

    @Override
    public void onCancel(Object o) {
        mEventSink = null;
    }
});

}

@Override
protected void onResume() {
    super.onResume();
    if(mEventSink != null){
        mEventSink.success("Android Message");
    }
}

```

Android 端首先是创建 EventChannel 实例，要保证 EventChannel 的 EventName 和 Flutter 的保持一致，同时实现 setStreamHandler() 方法，让 Android 和 Flutter 建立连接，而且

要保存 onListen() 方法里传过来的 EventSink 对象，需要用 EventSink 对象来发送数据

发送数据的代码就是：

- 发送正常数据的事件

```
if(mEventSink != null){  
    mEventSink.success("Android Message");  
}
```

- 发送 error 事件

```
if(mEventSink != null){  
    mEventSink.error("-1", "EventChannel  
fail", "content is null");  
}
```

- 发送 结束监听的事件

```
if(mEventSink != null){  
    mEventSink.endOfStream();  
}
```

Android 的 MainActivity.java 的代码：

```
package com.kk.flutter_app;  
  
import android.os.Bundle;  
import io.flutter.app.FlutterActivity;  
import io.flutter.plugin.common.EventChannel;  
import  
io.flutter.plugins.GeneratedPluginRegistrant;  
  
public class MainActivity extends FlutterActivity
```

```
{

    private EventChannel.EventSink mEventSink;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        GeneratedPluginRegistrant.registerWith(this);

        new
EventChannel(getFlutterView(), "samples.flutter.io
/event").setStreamHandler(new
EventChannel.StreamHandler() {
            @Override
            public void onListen(Object o,
EventChannel.EventSink eventSink) {
                mEventSink = eventSink;
            }

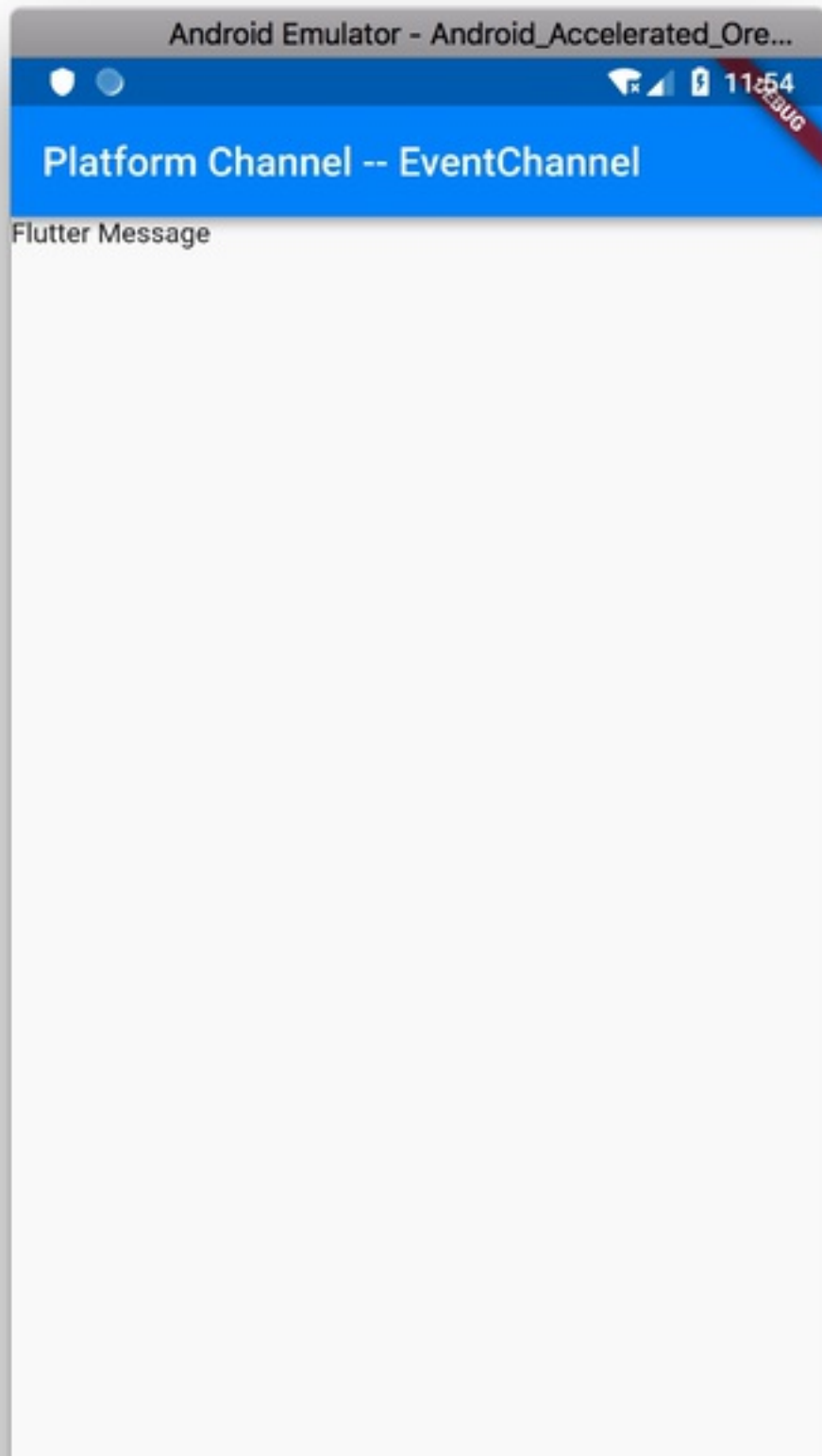
            @Override
            public void onCancel(Object o) {
                mEventSink = null;
            }
        });
    }

    @Override
    protected void onResume() {
        super.onResume();
        if(mEventSink != null){
            mEventSink.success("Android Message");
        }
    }
}
```



```
}  
  }  
}
```

刚运行 APP 的效果：





退后台，又切到前台的效果：

## 总结

你会发现，`MethodChannel` 和 `EventChannel` 都可以实现 Flutter 与 Native 的双向通信，所以 `MethodChannel` 和 `EventChannel` 在使用过程中是可以互相替换的。

在官方文档里，更多的是对 `MethodChannel` 的介绍，`EventChannel` 的几乎没有，但你可以根据自己的需要和习惯来选择使用，因为 `MethodChannel` 和 `EventChannel` 的底层实现是一样的。