LeanCode

Programming Mobile Applications in Flutter

Communication with Native

How does Flutter deal with platform-specific APIs?

How does Flutter deal with platform-specific APIs?

- Flutter is hosted by an ambient Android or iOS app
- **Platform channels** provide a simple mechanism for communicating between your Dart code and the platform-specific code of your host app
- **Plugins** make it possible to create a Dart API backed by an Android implementation and an iOS - and package that up as a Flutter/Android/iOS triple glued together using platform channels

When do I need to use platform-specific APIs

- Notifications, app lifecycle, deep links
- Sensors, camera, battery, geolocation, sound, connectivity
- Sharing information with other apps, launching other apps
- Persisted preferences, special folders, device information
- Intensive, high performance requiring tasks (e.g. Bitmap processing)
- …

Which language can I use?

Flutter uses a flexible system that allows you to call platform-specific APIs in a language that works directly with those APIs:

- Kotlin or Java on Android
- Swift or Objective-C on iOS
- C++ on Windows
- Swift or Objective-C on macOS
- C on Linux

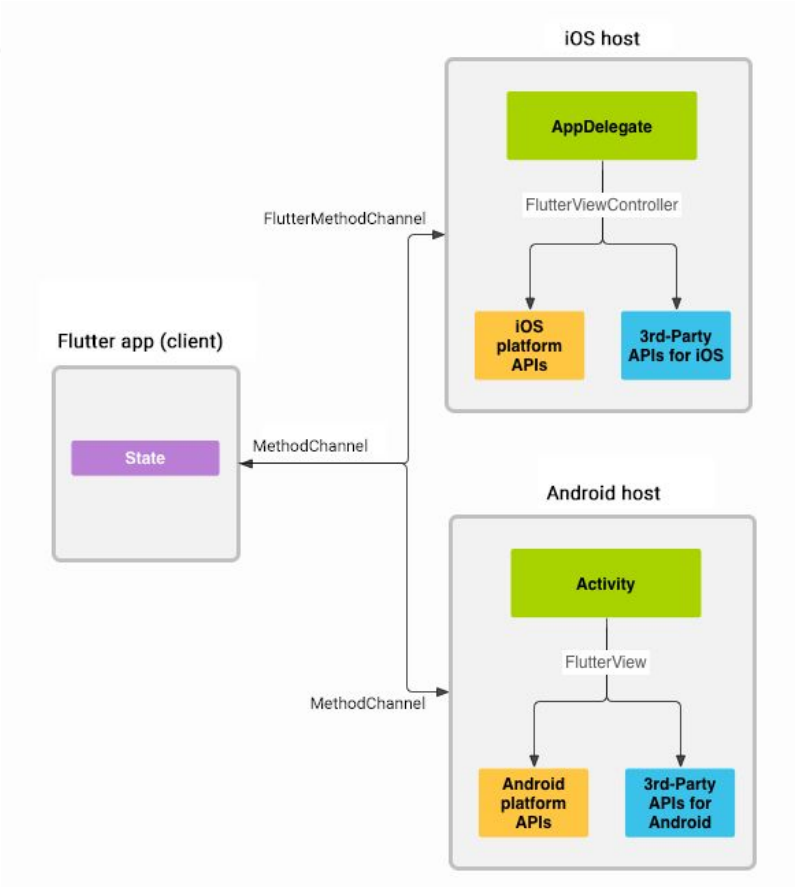
Method channels

A named channel for communicating with platform plugins using asynchronous method calls. Method calls are encoded into binary before being sent, and binary results received are decoded into Dart values.

When invoking channels on the platform side destined for Flutter, they need to be invoked on the platform’s main thread. When invoking channels in Flutter destined for the platform side, they need to be invoked on the root Isolate.

What is a main/UI Thread?

It’s the main thread of execution for your application which is responsible for managing the UI.



Source: https://docs.flutter.dev/development/platform-integration/platform-channels

```dart
class _MyHomePageState extends State<MyHomePage> {
  static const platform = MethodChannel('samples.flutter.dev/battery');

  String _batteryLevel = 'Unknown battery level.';

  Future<void> _getBatteryLevel() async {
    String batteryLevel;
    try {
      final int result = await platform.invokeMethod('getBatteryLevel');
      batteryLevel = 'Battery level at $result % .';
    } on PlatformException catch (e) {
      batteryLevel = "Failed to get battery level: '${e.message}'.";
    }

    setState(() {
      _batteryLevel = batteryLevel;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Material(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            ElevatedButton(
              child: const Text('Get Battery Level'),
              onPressed: _getBatteryLevel,
            ),  // ElevatedButton
            Text(_batteryLevel),
          ],
        ),  // Column
      ),  // Center
    );  // Material
  }
}
```

```kotlin
class MainActivity: FlutterActivity() {
    private val CHANNEL = "samples.flutter.dev/battery"

    override fun configureFlutterEngine(flutterEngine: FlutterEngine) {
        super.configureFlutterEngine(flutterEngine)
        MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler {
                call, result ->
            // Note: this method is invoked on the main thread.
            // TODO
        }
    }
}
```

```swift
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

```kotlin
override fun configureFlutterEngine(flutterEngine: FlutterEngine) {
    super.configureFlutterEngine(flutterEngine)
    MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler {
        // Note: this method is invoked on the main thread.
            call, result ->
        if (call.method == "getBatteryLevel") {
            val batteryLevel = getBatteryLevel()

            if (batteryLevel != -1) {
                result.success(batteryLevel)
            } else {
                result.error( errorCode: "UNAVAILABLE",  errorMessage: "Battery level not available.",  errorDetails: null)
            }
        } else {
            result.notImplemented()
        }
    }
}

private fun getBatteryLevel(): Int {
    return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        val batteryManager = getSystemService(Context.BATTERY_SERVICE) as BatteryManager
        batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
    } else {
        val intent = ContextWrapper(applicationContext).registerReceiver(
             receiver: null, IntentFilter(
                Intent.ACTION_BATTERY_CHANGED
            )
        )
        intent!!.getIntExtra(BatteryManager.EXTRA_LEVEL,  defaultValue: -1) * 100 / intent.getIntExtra(
            BatteryManager.EXTRA_SCALE,
             defaultValue: -1
        )
    }
}
```

```dart
final int result = await platform.invokeMethod(
  'getBatteryLevel',
  {
    "currentBatteryLevel": _batteryLevel,
  },
);
```

```kotlin
if (call.method == "getBatteryLevel") {
    val currentBatteryLevel = call.argument<String>( key: "currentBatteryLevel")
```

Dart values to the platform values and vice versa

Dart	Kotlin
null	null
bool	Boolean
int	Int
int, if 32 bits not enough	Long
double	Double
String	String
Uint8List	ByteArray
Int32List	IntArray
Int64List	LongArray
Float32List	FloatArray
Float64List	DoubleArray
List	List
Map	HashMap

What if I need to react to some platform events?

Event channels

A named channel for communicating with platform plugins using event streams!

```
static const eventChannel = EventChannel("sample.flutter/randomValueEventChannel");

@override
void initState() {
  eventChannel.receiveBroadcastStream().distinct().listen((event) {
    setState(() {
      _randomValue = event;
    });
  });
  super.initState();
}
```

```kotlin
private fun setupEventChannel(flutterEngine: FlutterEngine) {
    EventChannel(
        flutterEngine.dartExecutor.binaryMessenger,
        name: "sample.flutter/randomValueEventChannel"
    )
        .setStreamHandler(object : EventChannel.StreamHandler {
            override fun onListen(arguments: Any?, events: EventChannel.EventSink) {
                thread {
                    while (true) {
                        runOnUiThread { events.success(Random.nextInt()) }
                        Thread.sleep( millis: 2000)
                    }
                }
            }

            override fun onCancel(arguments: Any?) = Unit
        })
}
```

Can I use native views?

Platform View

- **AndroidView** - Embeds an Android view in the Widget hierarchy
- **UiKitView** - Embeds an iOS view in the Widget hierarchy
- **HtmlElementView** - Embeds an HTML element in the Widget hierarchy in Flutter Web
- …

```
internal class NativeView(context: Context, id: Int, creationParams: Map<String?, Any?>?) : PlatformView {
    private val textView by lazy {
        TextView(context).apply {  this: TextView
            textSize = 24f
            text = "Rendered on a native Android view (id: $id)"
        }
    }

    override fun getView(): View {
        return textView
    }

    override fun dispose() = Unit
}

class NativeViewFactory : PlatformViewFactory(StandardMessageCodec.INSTANCE) {
    override fun create(context: Context, viewId: Int, args: Any?): PlatformView {
        val creationParams = args as Map<String?, Any?>?
        return NativeView(context, viewId, creationParams)
    }
}
```

```
flutterEngine
    .platformViewsController
    .registry
    .registerViewFactory( viewTypeId: "androidNativeViewType", NativeViewFactory())
```

```
AndroidView(
  viewType: "androidNativeViewType",
),  // AndroidView
```

Contracts for MethodChannels

Piegeon

```
class Value {
  int? number;
}

@HostApi()
abstract class Api2Host {
  @async
  Value calculate(Value value);
}
```

```swift
// Swift

/** Generated interface from Pigeon that represents a handler of messages from Flutter.*/
protocol Api2Host {
  func calculate(value: Value, completion: @escaping (Value) -> Void)
}
```

```kotlin
// Kotlin

/** Generated interface from Pigeon that represents a handler of messages from Flutter.*/
interface Api2Host {
   fun calculate(value: Value, callback: (Value) -> Unit)
}
```

Pigeon isn’t perfect watch out for it’s limitations

Should I know Native if want to be a Flutter Developer?

Yes*

Should I know Native?

- Flutter is a good starting point for learning Native
- Most of the time you won’t have to touch native code due to big number of available plugins
- Native platforms knowledge gives you more possibilities and flexibility as you can easily integrate every Android/iOS library and do certain optimizations

Can I embed Flutter in an existing Native App?

Of course! You can even embed Flutter in React Native app

Questions?