🔗

Platform channels

허가받지 않은 복제(복사), 전송, 수정 및 배포를 금합니다.

Writing custom platform-specific code

플랫폼(OS) 자체에서 제공하는 **위치 정보, OS 버전, 접근 권한 여부** 등 정보를 어떻게 알 수 있을까요?

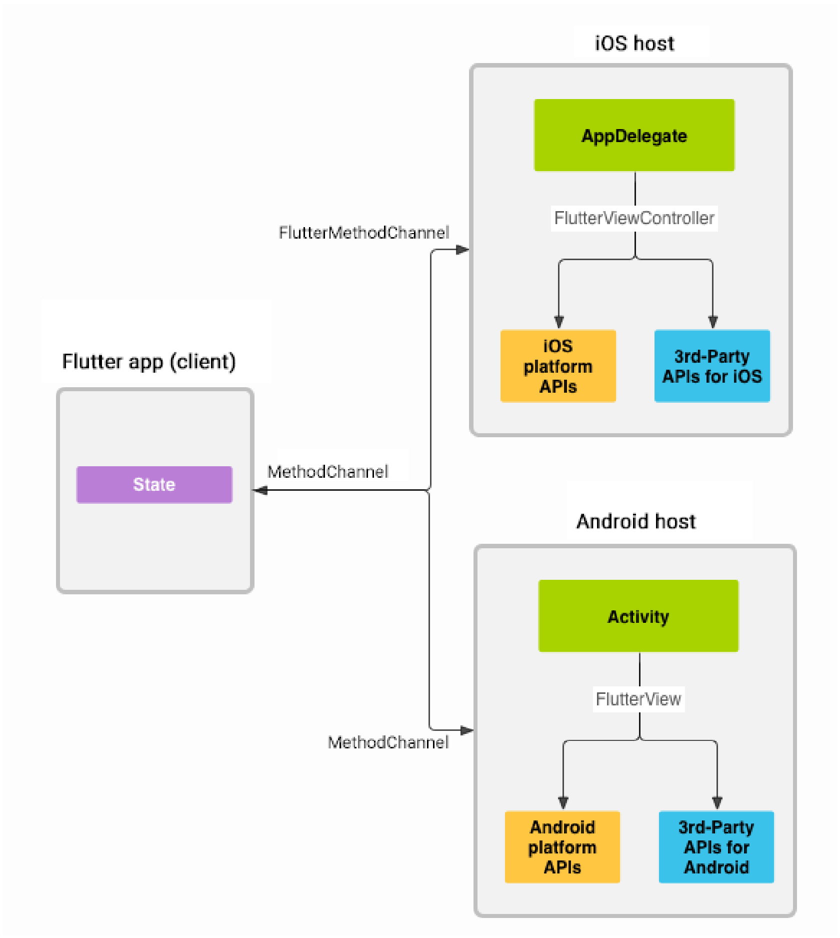
공유하기, 이미지 피커 등 플랫폼(OS) 자체 **시스템 기능**들을 어떻게 이용 할 수 있을까요?

👉 ***Platform channels***

platform-specific APIs

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

< 플랫폼 채널에서 사용하는 메시지 형태는 아래 다이어그램 >



요약

- **MethodChannel** 을 통해 플랫폼(네이티브)와 통신
  - result(응답값) 처리 필요
  - 응답값으로 지원하는 데이터 타입 전달 가능 (Error 도 가능)
- 메시지와 응답은 **비동기 처리**
- 각 네이티브 플랫폼은 **메인 스레드**에서 호출 해야함
- **양방향**으로 호출 가능
  - quick_action 예시

지원 데이터 타입

Aa Dart	Swift	Kotlin
null ⭐	nil	null
bool ⭐	NSNumber(value: Bool)	Boolean
int ⭐	NSNumber(value: Int32)	Int
int, if 32 bits not enough ⭐	NSNumber(value: Int)	Long

Aa Dart	Swift	Kotlin
double ⭐	NSNumber(value: Double)	Double
String ⭐	String	String
Uint8List ⭐	FlutterStandardTypedData(bytes: Data)	ByteArray
Int32List	FlutterStandardTypedData(int32: Data)	IntArray
Int64List	FlutterStandardTypedData(int64: Data)	LongArray
Float32List	FlutterStandardTypedData(float32: Data)	FloatArray
Float64List	FlutterStandardTypedData(float64: Data)	DoubleArray
List ⭐	Array	List
Map ⭐	Dictionary	HashMap

실제 예제 코드

디바이스의 남은 배터리 퍼센트 조회

dart

```
class BatteryLevel {
  // MethodChannel 인자로 id 값이 들어가요!
  static const _channel = const MethodChannel('samples.flutter.dev/battery');

  // 배터리 정보를 조회하는 메서드명 "getBatteryLevel"
  Future<int> getBatteryLevel() => _channel.invokeMethod('getBatteryLevel');

  BatteryLevel._internal() {
    _channel.setMethodCallHandler(_handleMethod);
  }

  Future<dynamic> _handleMethod(MethodCall call) async {
    switch (call.method) {
        case 'onLaunch':
          print('BatteryLevel - onLaunch');
          break;
        default:
          throw UnsupportedError('Unsupported error - method call handler');
    }
  }
}
```

iOS (swift)

```
public class BatteryLevelPlugin: NSObject, FlutterPlugin {
    public static func register(with registrar: FlutterPluginRegistrar) {
        let channel = FlutterMethodChannel(name: "samples.flutter.dev/battery", binaryMessenger: registrar.messenger())
        let instance = BatteryLevelPlugin()
        registrar.addMethodCallDelegate(instance, channel: channel!)

        // iOS => Flutter
        channel!.invokeMethod("onLaunch", arguments: nil)
    }

    public func handle(_ call: FlutterMethodCall, result: @escaping FlutterResult) {
        if call.method == "getBatteryLevel" {
            result(100)
        } else {
            result(FlutterMethodNotImplemented)
        }
    }
}
```

Android (kotlin)

```
class BatteryLevelPlugin: FlutterPlugin, MethodCallHandler {
  private var channel: MethodChannel? = null

  override fun onAttachedToEngine(@NonNull flutterPluginBinding: FlutterPlugin.FlutterPluginBinding) {
    channel = MethodChannel(flutterPluginBinding.binaryMessenger, "samples.flutter.dev/battery")
    channel?.setMethodCallHandler(this)

    // Android => Flutter
    channel?.invokeMethod("onLaunch", null)
  }

  override fun onMethodCall(@NonNull call: MethodCall, @NonNull result: Result) {
    when (call.method) {
      "getBatteryLevel" -> {
        result.success(100)
      }
      else -> {
        result.notImplemented()
      }
    }
  }

  override fun onDetachedFromEngine(@NonNull binding: FlutterPlugin.FlutterPluginBinding) {
    channel?.setMethodCallHandler(null)
    channel = null
  }
}
```

Typesafe platform channels using Pigeon.

문제점

- **not typesafe**
- 메시지를 주고 받을 때, 동일한 arguments and datatypes 보장 어려움.

해결

- typesafe 하도록 Pigeon 패키지 활용하여 `MethodChannel` 코드 **generate**
- 메시지의 names and datatypes 등 신경 안써도 됩니다.
  네이티브 플랫폼, Flutter 각각 동일한 인터페이스 생성 보장.

현재 지원 언어 : Objective-C, Swift, Java, Kotlin

+ C++ code for Windows

Pigeon example.

Pigeon file: messages.dart

```
import 'package:pigeon/pigeon.dart';

@ConfigurePigeon(PigeonOptions(
  dartOut: 'lib/src/messages.g.dart',
  cppOptions: CppOptions(namespace: 'pigeon_example'),
  cppHeaderOut: 'windows/runner/messages.g.h',
  cppSourceOut: 'windows/runner/messages.g.cpp',
  kotlinOut:
      'android/app/src/main/kotlin/dev/flutter/pigeon_example_app/Messages.g.kt',
  // This file is also used by the macOS project.
  swiftOut: 'ios/Runner/Messages.g.swift',
  copyrightHeader: 'pigeons/copyright.txt',
))
@HostApi()
abstract class ExampleHostApi {
  String getHostLanguage();
}
```

Generate CLI:

```
dart run pigeon --input pigeons/messages.dart
```

swift: Messages.g.swift

```
/// Generated protocol from Pigeon that represents a handler of messages from Flutter.
protocol ExampleHostApi {
  func getHostLanguage() throws -> String
}

/// Generated setup class from Pigeon to handle messages through the `binaryMessenger`.
class ExampleHostApiSetup {
  /// The codec used by ExampleHostApi.
  /// Sets up an instance of `ExampleHostApi` to handle messages through the `binaryMessenger`.
  static func setUp(binaryMessenger: FlutterBinaryMessenger, api: ExampleHostApi?) {
    let getHostLanguageChannel = FlutterBasicMessageChannel(
      name: "dev.flutter.pigeon.ExampleHostApi.getHostLanguage", binaryMessenger: binaryMessenger)
    if let api = api {
      getHostLanguageChannel.setMessageHandler { _, reply in
        do {
          let result = try api.getHostLanguage()
          reply(wrapResult(result))
        } catch {
          reply(wrapError(error))
        }
      }
    } else {
      getHostLanguageChannel.setMessageHandler(nil)
    }
  }
}
```

swift: AppDelegate.swift

```
import Flutter
import UIKit

/// 실제 구현체
private class PigeonApiImplementation: ExampleHostApi {
  func getHostLanguage() throws -> String {
    return "Swift"
  }
}

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

    let controller = window?.rootViewController as! FlutterViewController

    /// 등록해주는 코드
    let api = PigeonApiImplementation()
    ExampleHostApiSetup.setUp(binaryMessenger: controller.binaryMessenger, api: api)

    return super.application(application, didFinishLaunchingWithOptions: launchOptions)

  }
}
```

Kotlin: Message.g.kt

```
/** Generated interface from Pigeon that represents a handler of messages from Flutter. */
interface ExampleHostApi {
  fun getHostLanguage(): String

  companion object {
    /** The codec used by ExampleHostApi. */
    val codec: MessageCodec<Any?> by lazy {
      StandardMessageCodec()
    }
    /** Sets up an instance of `ExampleHostApi` to handle messages through the `binaryMessenger`. */
    @Suppress("UNCHECKED_CAST")
    fun setUp(binaryMessenger: BinaryMessenger, api: ExampleHostApi?) {
      run {
        val channel = BasicMessageChannel<Any?>(binaryMessenger, "dev.flutter.pigeon.ExampleHostApi.getHostLanguage", codec)
        if (api != null) {
          channel.setMessageHandler { _, reply ->
            var wrapped: List<Any?>
            try {
              wrapped = listOf<Any?>(api.getHostLanguage())
            } catch (exception: Throwable) {
              wrapped = wrapError(exception)
            }
            reply.reply(wrapped)
          }
        } else {
          channel.setMessageHandler(null)
        }
      }
    }
  }
}
```

Kotlin: MainActivity.kt

```
package dev.flutter.pigeon_example_app

import ExampleHostApi
import androidx.annotation.NonNull
import io.flutter.embedding.android.FlutterActivity
import io.flutter.embedding.engine.FlutterEngine

private class PigeonApiImplementation: ExampleHostApi {
    override fun getHostLanguage(): String {
        return "Kotlin"
    }
}

class MainActivity: FlutterActivity() {
    override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
        super.configureFlutterEngine(flutterEngine)

        val api = PigeonApiImplementation()
        ExampleHostApi.setUp(flutterEngine.dartExecutor.binaryMessenger, api);
    }
}
```

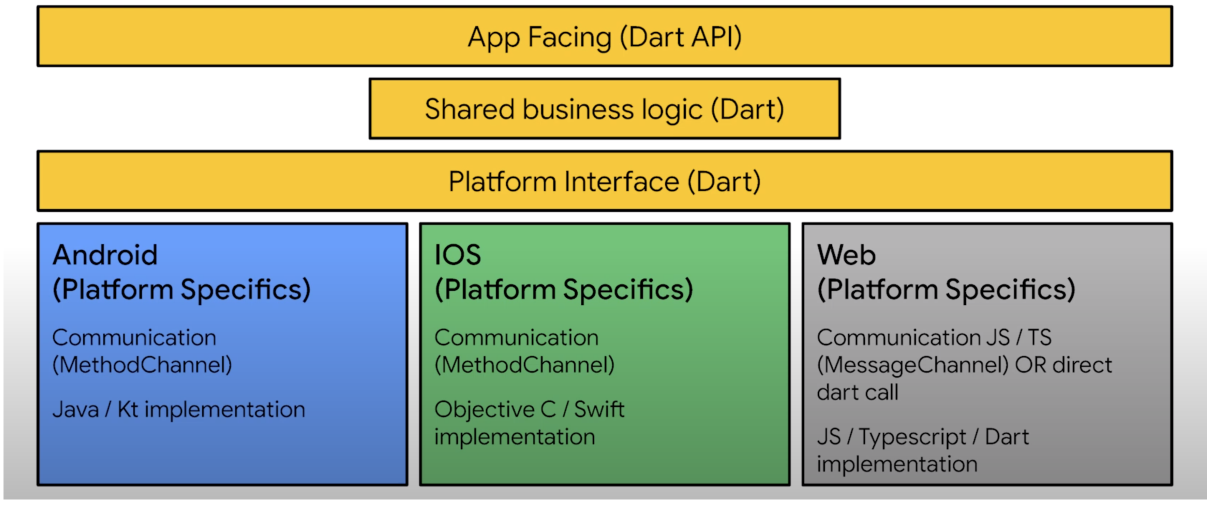
Separate platform-specific code from UI code.

Federated plugins

서로 다른 플랫폼에 대한 지원을 별도의 패키지로 분할
iOS용 패키지, Android용 패키지, 웹용 패키지, 그리고 자동차용 패키지(IoT 장치의 예)…

< Google IO 2022 - Flutter lessons for federated plugin development >

Plugin Diagram


App Facing (Dart API)
Shared business logic (Dart)
Platform Interface (Dart)
Android (Platform Specifics)
Communication (MethodChannel)
Java / Kt implementation
IOS (Platform Specifics)
Communication (MethodChannel)
Objective C / Swift implementation
Web (Platform Specifics)
Communication JS / TS (MessageChannel) OR direct dart call
JS / Typescript / Dart implementation

app-facing package

플러그인 사용자가 Flutter app 에서 직접 사용하는 패키지

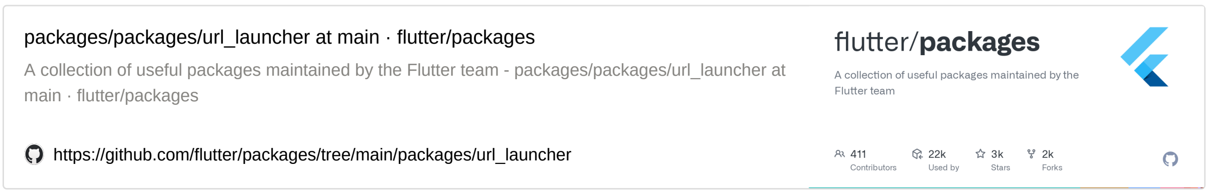
platform package(s)

플랫폼(OS) 별 구현 코드가 포함된 패키지로 **app-facing package** 에 의해 호출되어서 사용됨.
단, 특정 플랫폼 기능이 아니라면 사용자가 직접 이 패키지를 접근해서 쓰지 않음.

platform interface package

플랫폼별 통일된 인터페이스 패키지

Federated plugins 예시 - url_launcher



packages/packages/url_launcher at main · flutter/packages

A collection of useful packages maintained by the Flutter team - packages/packages/url_launcher at main · flutter/packages

https://github.com/flutter/packages/tree/main/packages/url_launcher

flutter/**packages**

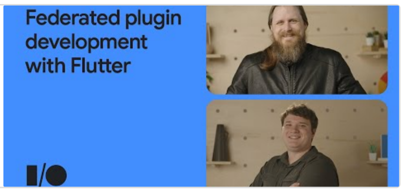
A collection of useful packages maintained by the Flutter team

411 Contributors 22k Used by 3k Stars 2k Forks

Google IO 2022 - Flutter lessons for federated plugin development

Flutter lessons for federated plugin development

https://youtu.be/GAnSNplNpCA



Custom channels and codecs.

MethodChannel 말고도 `BasicMessageChannel` 활용하면 커스텀 메시지 코덱 사용하여 비동기 처리

> cloud_firestore 에서 커스텀 코덱 예시 확인 가능

Package? Plugin?

- 패키지 : **다트**로만 구성되어 있는 패키지
- 플러그인 : **멀티 플랫폼 지원**을 위한 코드가 포함된 패키지

Channels and platform threading.

You can invoke the platform side handlers asynchronously and **on any thread**.

- root `Isolate`
- registered as a background `Isolate` for a root `Isolate`.

Dart 기본 원리

- Dart 는 싱글 쓰레드 환경
- 비동기 지원
  - async, await
  - 비동기 처리를 지원하더라도 **무거운 작업과 UI 업데이트 같이** 진행되면 버벅거림

isolate

- 싱글 스레드
- 별도 이벤트 루프
- 개별 메모리
    - isolate 간의 메모리 공유하지 않음
    - 싱글턴 패턴으로 만든 클래스 hash code 확인해보면 다른걸 알 수 있음
- 기본적으로 앱 실행되면 **main isolate** 에서 실행됨.
- `Isolate.spawn`
- `Isolate.run()` ⇒ `compute`

how to register a background `Isolate`

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

void _isolateMain(RootIsolateToken rootIsolateToken) async {
  BackgroundIsolateBinaryMessenger.ensureInitialized(rootIsolateToken);
  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
  print(sharedPreferences.getBool('isDebug'));
}

void main() {
  RootIsolateToken rootIsolateToken = RootIsolateToken.instance!;
  Isolate.spawn(_isolateMain, rootIsolateToken);
}
```

Executing channel handlers on background threads.

Task Queue API

In Swift:

```
public static func register(with registrar: FlutterPluginRegistrar) {
  let taskQueue = registrar.messenger.makeBackgroundTaskQueue()
  let channel = FlutterMethodChannel(name: "com.example.foo",
                                     binaryMessenger: registrar.messenger(),
                                     codec: FlutterStandardMethodCodec.sharedInstance,
                                     taskQueue: taskQueue)
  let instance = MyPlugin()
  registrar.addMethodCallDelegate(instance, channel: channel)
}
```

In Kotlin:

```
override fun onAttachedToEngine(@NonNull flutterPluginBinding: FlutterPlugin.FlutterPluginBinding) {
  val taskQueue =
      flutterPluginBinding.binaryMessenger.makeBackgroundTaskQueue()
  channel = MethodChannel(flutterPluginBinding.binaryMessenger,
                          "com.example.foo",
                          StandardMethodCodec.INSTANCE,
                          taskQueue)
  channel.setMethodCallHandler(this)
}
```

UI thread(Android)

```
Handler(Looper.getMainLooper()).post {
  // Call the desired channel message here.
}
```

main thread(iOS)

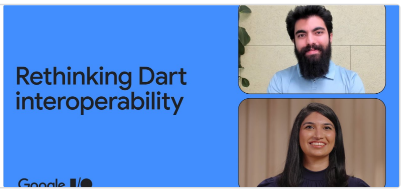
```
DispatchQueue.main.async {
  // Call the desired channel message here.
}
```

Google IO 2023 - Rethinking Dart interoperability with Android.

Rethinking Dart interoperability with Android

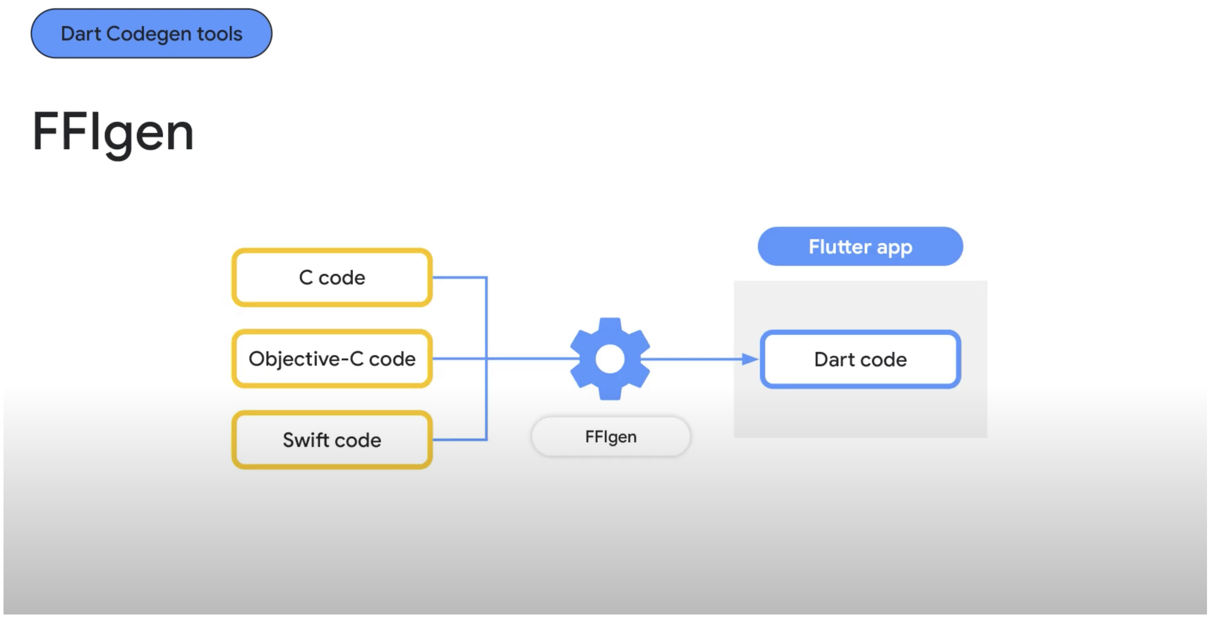
In the past, Flutter only supported integration with Android libraries through a message-based approach called platform channels. With a new command using JNI to bridge to Android system APIs, Flutter developers can easily access platform APIs without needing to use a platform channel

https://youtu.be/ZWp2FJ2TuJs



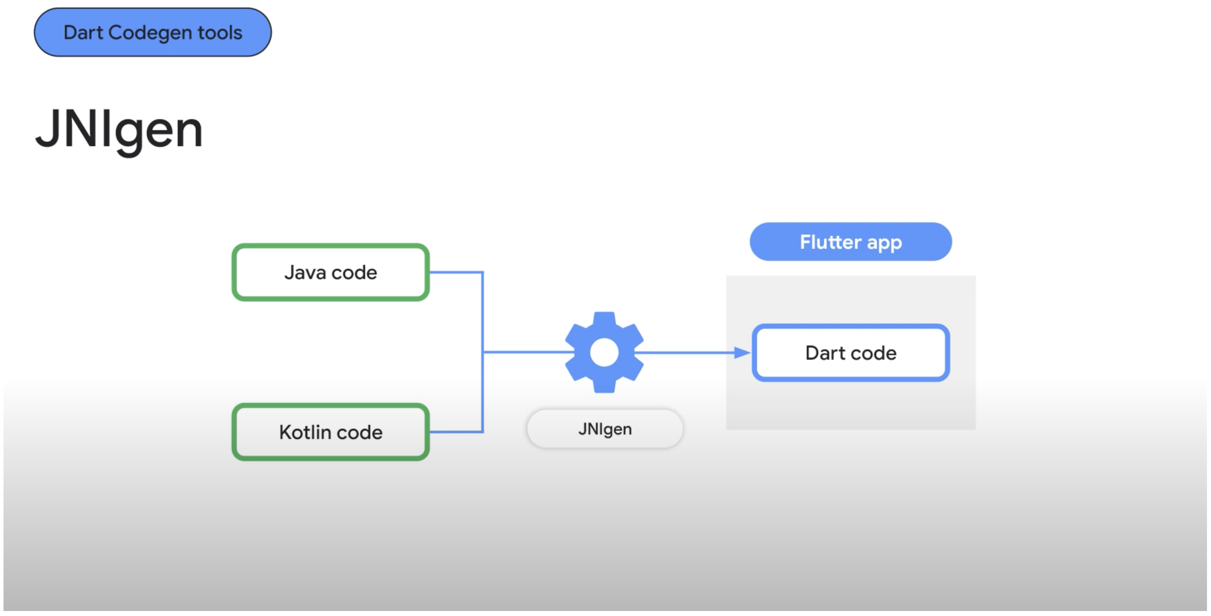
> This feature is **experimental.**
> https://github.com/dart-lang/sdk/issues/49673 https://github.com/dart-lang/sdk/issues/49674

FFI (Foreign Function Interface)



- `dart:ffi` 다트 코드와 C APIs 상호 작용 가능하도록 해줌
- Objective-C 는 compatible with C 하여 APIs 상호 작용 가능 (Android 와 차이점)
- Swift APIs 를 위해 Objc 헤더 파일 생성

JNI (Java Native Interface)



- 다트 코드와 Java 언어를 JNI 통해 상호 작용

- 사용하기 위해서는 생성해줘야 하는 보일러 플레이트 코드가 있음.
  👉 생성해주는 도구가 JNIgen

example.

Example.java:

```
package dev.dart;

public class Example {
  public static int sum(int a, int b) {
    return a + b;
  }
}
```

jnigen.yaml:

```
output:
  c:
    library_name: example
    path: src/example/
  dart:
    path: lib/example.dart
    structure: single_file

source_path:
  - 'java'
classes:
  - 'dev.dart.Example'
```

To generate bindings, run:

```
dart run jnigen --config jnigen.yaml
```

example.c:

```
// ... 생략

jmethodID _m_Example__sum = NULL;
FFI_PLUGIN_EXPORT
JniResult Example__sum(int32_t a, int32_t b) {
  load_env();
  load_class_gr(&_c_Example, "dev/dart/Example");
  if (_c_Example == NULL)
    return (JniResult){.result = {.j = 0}, .exception = check_exception()};
  load_static_method(_c_Example, &_m_Example__sum, "sum", "(II)I");
  if (_m_Example__sum == NULL)
    return (JniResult){.result = {.j = 0}, .exception = check_exception()};
  int32_t _result =
      (*jniEnv)->CallStaticIntMethod(jniEnv, _c_Example, _m_Example__sum, a, b);
  return (JniResult){.result = {.i = _result}, .exception = check_exception()};
}
```

example.dart:

```
import "dart:ffi" as ffi;
import "package:jni/internal_helpers_for_jnigen.dart";
import "package:jni/jni.dart" as jni;

// Auto-generated initialization code.

final ffi.Pointer<T> Function<T extends ffi.NativeType>(String sym) jniLookup =
    ProtectedJniExtensions.initGeneratedLibrary("example");

/// from: dev.dart.Example
class Example extends jni.JObject {
  late final jni.JObjType? _$type;
  @override
  jni.JObjType get $type => _$type ??= type;

  Example.fromRef(
    jni.JObjectPtr ref,
  ) : super.fromRef(ref);

  /// The type which includes information such as the signature of this class.
  static const type = $ExampleType();

  static final _ctor =
      jniLookup<ffi.NativeFunction<jni.JniResult Function()>>("Example__ctor")
          .asFunction<jni.JniResult Function()>();

  /// from: public void <init>()
  Example() : super.fromRef(_ctor().object);

  static final _sum = jniLookup<
              ffi.NativeFunction<jni.JniResult Function(ffi.Int32, ffi.Int32)>>(
          "Example__sum")
      .asFunction<jni.JniResult Function(int, int)>();

  /// from: static public int sum(int a, int b)
  static int sum(int a, int b) => _sum(a, b).integer;
}

/// ... 생략
```

실제 사용 sum.dart:

```
// Prerequisites:
// Run `dart run jni:setup -p jni -s src/example`
// Run `javac java/dev/dart/Example.java`

/// ... 생략
print(Example.sum(a, b)); // prints a + b
```

Why?

- 그럼 **Pigeon 과 FFIgen, JNIgen 차이점**이 뭘까?
- Pigeon 이 있는데 **FFIgen, JNIgen** 왜 만든걸까?

👉 purely Dart app 개발 할 때 사용될 수 있음

👉 네이티브 플랫폼 OS 기능을 활용하기 쉬워지고 유지보수도 편해질 수 있음

pedometer example.

만보기 앱

참고 자료.

Writing custom platform-specific code

This guide describes how to write custom platform-specific code. Some platform-specific functionality is available through existing packages; see using packages. Note: The information in this page is valid for most platforms, but platform-specific code for the web generally uses JS

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

Flutter

Developing packages & plugins

The plugin API supports federated plugins that enable separation of different platform implementations. You can also now indicate which platforms a plugin supports, for example web and macOS. Eventually, the old plugin APIs will be deprecated. In the short term, you will see a

https://docs.flutter.dev/development/packages-and-plugins/developing-packages#federated-plug
ins

Flutter

C interop using dart:ffi

To use C code in your Dart program, use the dart:ffi library.

https://dart.dev/guides/libraries/c-interop

Dart