Flutter官方推荐插件开发辅助工具-Pigeon

xmb LV.3
2021年09月14日 17:54 · 阅读 10924
关注

工具插件：pigeon

推荐必看的官方例子：pigeon_plugin_example

其他pigeon写法的例子：① pigeon/example ② pigeons

> Pigeon is a code generator tool to make communication between Flutter and the host platform type-safe, easier and faster. 一个代码生成工具，让 Flutter 和宿主平台更安全、更简单、更快地通信。

通过 Dart 入口，生成两端通用的模板代码，原生则只需重写模板内的接口，无需管理 Method Channel 的实现。参数可以通过模板来同步生成。

目前的 pigeon 只支持生成 OC 和 Java 代码。

1、命令创建 Plugin

```
$ flutter create --template=plugin --platforms=android,ios -i swift -a kotlin flutter_pigeon_plugin
```

2、 flutter 项目中 pubspec.yaml 的 dev_dependencies 中，添加 pigeon 插件依赖

yaml 复制代码

```yaml
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^1.0.0

  pigeon:
```

3、在 Flutter 项目的 lib 目录外创建一个 pigeons 文件夹，在 pigeons 文件夹中创建 all_types_pigeon.dart

java 复制代码

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

class Everything {
  bool? aBool;
```

```
    int? anInt;
    double? aDouble;
    String? aString;
    Uint8List? aByteArray;
    Int32List? a4ByteArray;
    Int64List? a8ByteArray;
    Float64List? aFloatArray;
    // ignore: always_specify_types
    List? aList;
    // ignore: always_specify_types
    Map? aMap;
    List<List<bool?>?>? nestedList;
    Map<String?, String?>? mapWithAnnotations;
  }

  /// Flutter调用原生的方法
  @HostApi()
  abstract class HostEverything {
    Everything giveMeEverything();
    Everything echo(Everything everything);
  }

  /// 原生调用Flutter的方法
  @FlutterApi()
  abstract class FlutterEverything {
    Everything giveMeEverythingFlutter();
    Everything echoFlutter(Everything everything);
  }
```

4、执行命令

首先创建存放生成文件的文件夹： 在 `android/src/mian` 下创建 `java/com/example/flutter_pigeon_plugin/` 文件夹，存放生成的 `Java` 文件。

在项目目录 `~/flutter_pigeon_plugin` 下，执行以下命令： `$ flutter pub run pigeon --input pigeons/all_types_pigeon.dart --dart_out lib/all_types_pigeon.dart --objc_header_out ios/Classes/AllTypesPigeon.h --objc_source_out ios/Classes/AllTypesPigeon.m --java_out android/src/main/java/com/example/flutter_pigeon_plugin/AllTypesPigeon.java --java_package "com.example.flutter_pigeon_plugin"`

```
rb 复制代码
命令拆解：
①` flutter pub run pigeon`
生成代码的命令

②` --input pigeons/all_types_pigeon.dart `
指定生成代码的输入`dart`文件
```

③ `--dart_out lib/all_types_pigeon.dart `
指定输出生成`dart`文件的目录文件

④ `--objc_header_out ios/Classes/AllTypesPigeon.h  `
指定要生成的`iOS`的`.h`文件路径

⑤ `--objc_source_out ios/Classes/AllTypesPigeon.m  `
指定要生成的`iOS`的`.m`文件路径

⑥ `--java_out android/src/main/java/com/example/flutter_pigeon_plugin/AllTypesPigeon.java `
指定要生成的`Android`的`.java`文件路径

⑦ `--java_package "com.example.flutter_pigeon_plugin`
指定`Android`的包名，在`android/src/main/`下的`AndroidManifest.xml`里的`package`

⑧ `--objc_prefix FLT` （可选）指定生成OC文件的前缀为FLT，前缀自己定义为自己的。

可以参考官方的例子里的做法：① 项目目录下创建一个 `run_pigeon.sh` 文件 ② 每次有改动，执行命令：`. ./run_pigeon.sh` 即可 ③ `run_pigeon.sh` 文件内容如下：

shell 复制代码
```shell
flutter pub run pigeon \
  --input pigeons/all_types_pigeon.dart \
  --dart_out lib/all_types_pigeon.dart \
  --objc_header_out ios/Classes/AllTypesPigeon.h \
  --objc_source_out ios/Classes/AllTypesPigeon.m \
  --objc_prefix FLT \
  --java_out android/src/main/java/com/example/flutter_pigeon_plugin/AllTypesPigeon.java \
  --java_package "com.example.flutter_pigeon_plugin"
```

命令执行完成，会自动在命令中指定的几个位置生成响应的文件。

5、 iOS 实现 Flutter 调用原生的方法

① 删掉项目中之前的获取版本的原生的和 `Flutter` 侧的相关 `channel` 代码 ② 在 `AllTypesPigeon.m` 中自动生成了一个方法 `HostEverythingSetup`

less 复制代码
```less
void HostEverythingSetup(id<FlutterBinaryMessenger> binaryMessenger, NSObject<HostEverythin
  {
    FlutterBasicMessageChannel *channel =
      [FlutterBasicMessageChannel        messageChannelWithName:@"dev.flutter.pigeon.HostEv
    if (api) {
      NSCAssert([api respondsToSelector:@selector(giveMeEverythingWithError:)], @"HostEvery
      [channel setMessageHandler:^(id _Nullable message, FlutterReply callback) {        Fl
        callback(wrapResult(output, error));
      }];
    }
    else {
      [channel setMessageHandler:nil];
    }
  }
  {
    FlutterBasicMessageChannel *channel =
      [FlutterBasicMessageChannel        messageChannelWithName:@"dev.flutter.pigeon.HostEv
    if (api) {
      NSCAssert([api respondsToSelector:@selector(echoEverything:error:)], @"HostEverything
      [channel setMessageHandler:^(id _Nullable message, FlutterReply callback) {        NS
        FlutterError *error;
        Everything *output = [api echoEverything:arg_everything error:&error];
        callback(wrapResult(output, error));
      }];
    }
    else {
      [channel setMessageHandler:nil];
    }
  }
}
```

③ 在 `SwiftFlutterPigeonPlugin.swift` 的注册方法里，调用这个 `setup` 方法进行初始化和设置

swift 复制代码

```swift
    public static func register(with registrar: FlutterPluginRegistrar) {
        let messenger: FlutterBinaryMessenger = registrar.messenger()
        let api: HostEverything & NSObjectProtocol = SwiftFlutterPigeonPlugin.init()
        HostEverythingSetup(messenger, api)
    }
```

④ `SwiftFlutterPigeonPlugin` 遵循 `HostEverything` 协议，实现 `Flutter` 调原生的方法

swift 复制代码

```swift
import Flutter
import UIKit

/// 遵循HostEverything协议，实现Flutter调原生的方法
public class SwiftFlutterPigeonPlugin: NSObject, FlutterPlugin, HostEverything {
    public static func register(with registrar: FlutterPluginRegistrar) {
        let messenger: FlutterBinaryMessenger = registrar.messenger()
        let api: HostEverything & NSObjectProtocol = SwiftFlutterPigeonPlugin.init()
        HostEverythingSetup(messenger, api)
    }

    // MARK: HostEverything

    public func giveMeEverythingWithError(_ error: AutoreleasingUnsafeMutablePointer<Flutte
        let everyThing = Everything()
        everyThing.aString = "原生返给Flutter的字符串"
        everyThing.aBool = false
        everyThing.anInt = 11
        return everyThing
    }

    /// 遵循HostEverything协议，实现Flutter调原生的方法
    public func echo(_ everything: Everything, error: AutoreleasingUnsafeMutablePointer<Flut
        everything.aString = "原生交换的给Flutter的字符串"
        everything.aBool = false
        everything.anInt = 12
        return everything
    }
}
```

⑤ `iOS/Classes` 目录下，创建 `flutter_pigeon_plugin.h` 文件，导入头文件，此文件在 `iOS` 自动生成的 `<flutter_pigeon_plugin/flutter_pigeon_plugin-Swift.h>` 文件中会自动引用。

oc 复制代码

```
//
//  flutter_pigeon_plugin.h
//  Pods
//
//  Created by yuanzhiying on 2021/9/13.
//

#ifndef flutter_pigeon_plugin_h
#define flutter_pigeon_plugin_h

#import "AllTypesPigeon.h"

#endif /* flutter_pigeon_plugin_h */
```

6、 `flutter_pigeon_plugin.dart` 中实现插件调原生方法

dart 复制代码

```
import 'dart:async';

import 'package:flutter_pigeon_plugin/all_types_pigeon.dart';

class FlutterPigeonPlugin {
  static final HostEverything _hostEverything = HostEverything();

  /// Flutter 调用原生方法
  static Future<Everything> getEverything() async {
    return await _hostEverything.giveMeEverything();
  }

  /// Flutter 调用原生方法
  static Future<Everything> echoEveryThing(Everything everything) async {
    return await _hostEverything.echo(everything);
  }
}
```

7、使用插件的方法

dart 复制代码

```dart
Future<void> getHostData() async {
  /// 通过插件调用原生方法
  Everything everything = await FlutterPigeonPlugin.getEverything();
  debugPrint('everything.aString: ${everything.aString}');
  debugPrint('everything.aBool: ${everything.aBool}');
  debugPrint('everything.anInt: ${everything.anInt}');
}

void echoHostData() async {
  Everything echoEveryThing = Everything();
  echoEveryThing.aString = '我要跟原生交换的字符串';
  echoEveryThing.aBool = true;
  echoEveryThing.anInt = 10;

  /// 通过插件调用原生方法
  Everything everything = await FlutterPigeonPlugin.echoEveryThing(echoEveryThing);
  debugPrint('everything.aString: ${everything.aString}');
  debugPrint('everything.aBool: ${everything.aBool}');
  debugPrint('everything.anInt: ${everything.anInt}');
}
```

至此， `flutter` 调用原生 `iOS` 方法完成。

8、 `flutter` 调用安卓原生的实现

① 删除原有的获取版本号的 `channel` 的代码 ② `FlutterPigeonPlugin.kt` 中继承 `AllTypesPigeon.HostEverything` ，并实现对应的方法

kotlin 复制代码

```kotlin
override fun giveMeEverything(): AllTypesPigeon.Everything {
  var everything: AllTypesPigeon.Everything = AllTypesPigeon.Everything()
  everything.aString = "原生返给Flutter的字符串"
  everything.aBool = false
  everything.anInt = 11
  return everything
}
```

```
  override fun echo(everything: AllTypesPigeon.Everything?): AllTypesPigeon.Everything? {
    everything?.aString = "原生交换的给Flutter的字符串"
    everything?.aBool = false
    everything?.anInt = 12
    return everything
  }
```

③ 通过自动生成的setup方法，进行初始化和设置

kotlin 复制代码

```
  override fun onAttachedToEngine(@NonNull flutterPluginBinding: FlutterPlugin.FlutterPlugi
    AllTypesPigeon.HostEverything.setup(flutterPluginBinding.binaryMessenger, this)
  }

  override fun onDetachedFromEngine(@NonNull binding: FlutterPlugin.FlutterPluginBinding) {
    AllTypesPigeon.HostEverything.setup(binding.binaryMessenger, null)
  }
```

最终如下：

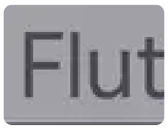
kotlin 复制代码

```
package com.example.flutter_pigeon_plugin

import androidx.annotation.NonNull

import io.flutter.embedding.engine.plugins.FlutterPlugin
import io.flutter.plugin.common.MethodCall
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugin.common.MethodChannel.MethodCallHandler
import io.flutter.plugin.common.MethodChannel.Result

/** FlutterPigeonPlugin */
class FlutterPigeonPlugin: FlutterPlugin, MethodCallHandler, AllTypesPigeon.HostEverything 

  override fun onAttachedToEngine(@NonNull flutterPluginBinding: FlutterPlugin.FlutterPlugi
    AllTypesPigeon.HostEverything.setup(flutterPluginBinding.binaryMessenger, this)
  }

  override fun onMethodCall(@NonNull call: MethodCall, @NonNull result: Result) {
    result.notImplemented()
  }

  override fun onDetachedFromEngine(@NonNull binding: FlutterPlugin.FlutterPluginBinding) {
    AllTypesPigeon.HostEverything.setup(binding.binaryMessenger, null)
  }

  override fun giveMeEverything(): AllTypesPigeon.Everything {
    var everything: AllTypesPigeon.Everything = AllTypesPigeon.Everything()
    everything.aString = "原生返给Flutter的字符串"
    everything.aBool = false
    everything.anInt = 11
    return everything
  }

  override fun echo(everything: AllTypesPigeon.Everything?): AllTypesPigeon.Everything? {
    everything?.aString = "原生交换的给Flutter的字符串"
    everything?.aBool = false
    everything?.anInt = 12
    return everything
  }
}
```

项目代码见：flutter_pigeon_plugin

分类： 前端 标签： Flutter 前端

文章被收录于专栏：



Flutter进阶

Flutter知识点总结

关注专栏