

# Flutter/Dart における FFI

**今日話すこと**

**FFI の実装が進んでいる理由は？**

## 【結論】

**主にパフォーマンスの観点から、  
Native Extension より FFI が適してい  
るため**

**FFI ?**

# Foreign **f**unction **i**nterface

今回は C 呼び出しの話

(C -> Dart の話は省きます)

# ひとまず自己紹介

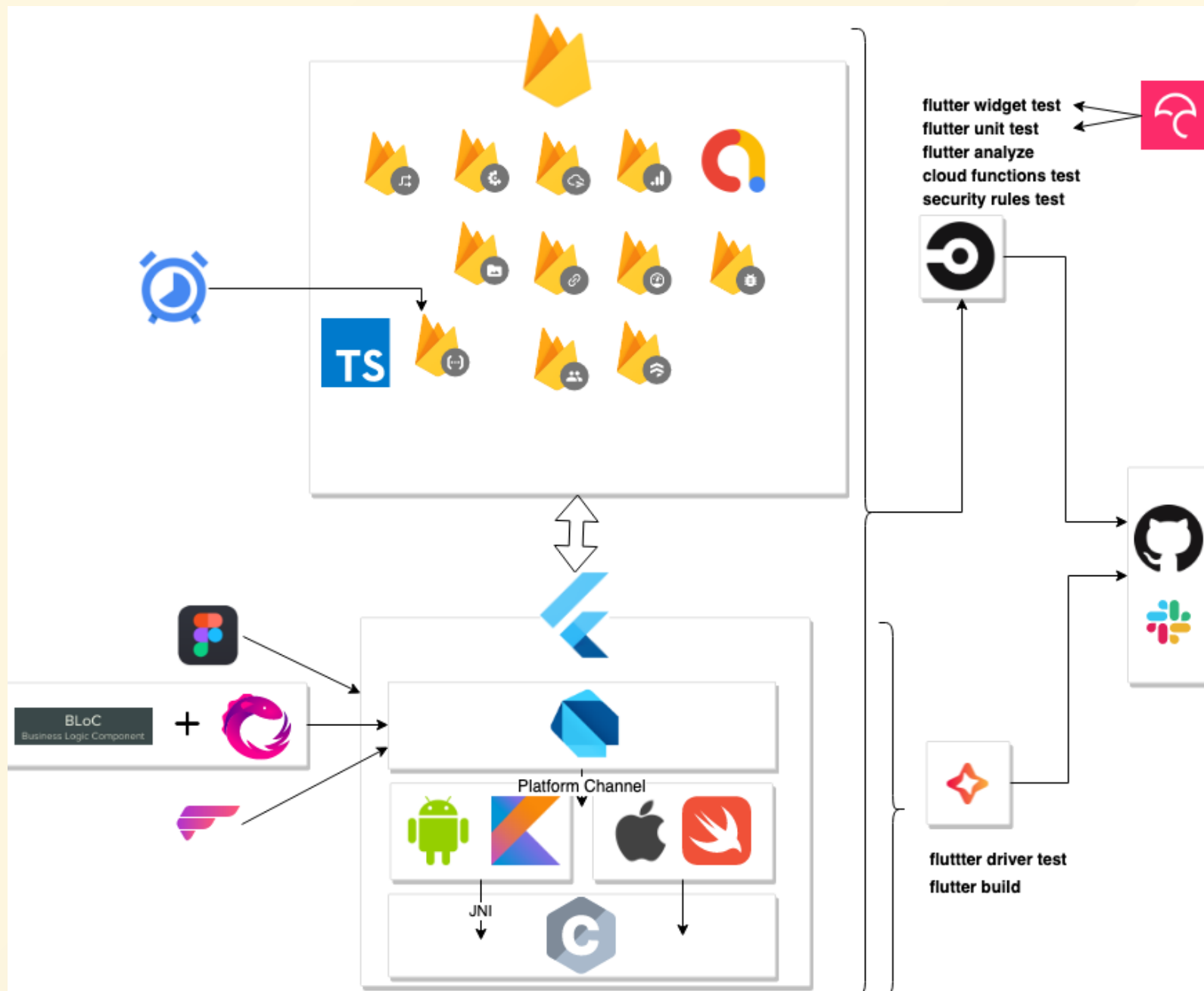
**しみず なおき**



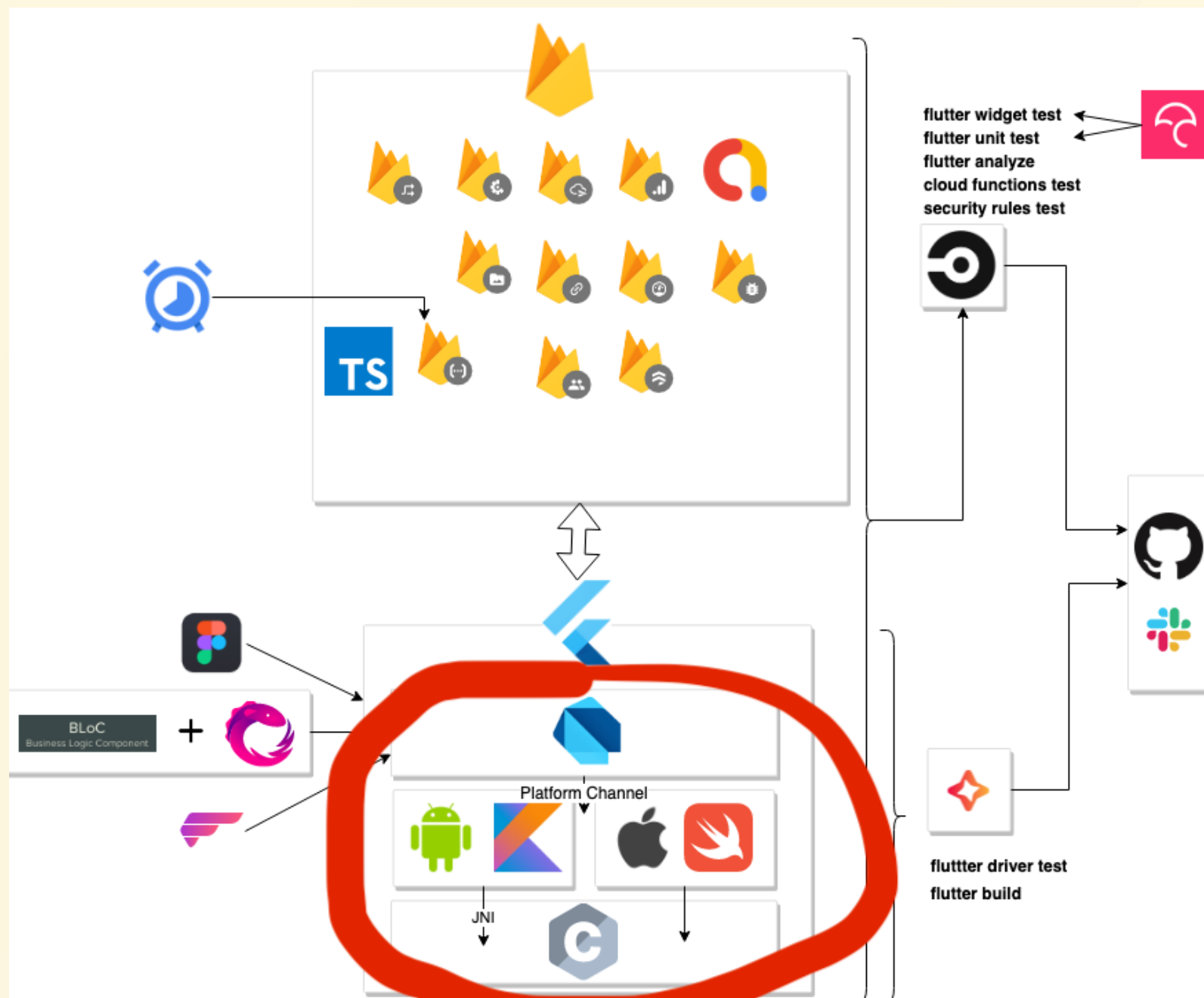




# お家で作ってるモノ



**オセロには常に C が必要**



# 各言語の C 呼び出し

## 代表的なもの

言語	実装方法
Java	<a href="#">JNI</a> や <a href="#">JNA</a> , <a href="#">SWIG</a> を使う
Go	<a href="#">cgo</a> を使う
Python	<a href="#">ctypes</a> や <a href="#">cffi</a> を使う
Rust	<a href="#">extern キーワード</a> で容易に呼べる
Ruby	<a href="#">Ruby-FFI</a> を使う
Javascript	<a href="#">WebAssembly</a> を使う
Swift	<a href="#">そのままいける</a> し、 <a href="#">カスタム</a> も可能

# Dart は？

# Dart から C を呼ぶ方法 (これまで)



# Native Extension

## Dart 側

```
library sample_hello;  
import 'dart-ext:sample_hello';  
void hello() native "Hello";
```

参考: [dart-lang sample extension](#)

## C++ 側 (一部省略)

```
DART_EXPORT Dart_Handle sample_hello_Init(Dart_Handle parent_library) {
    if (Dart_IsError(parent_library)) return parent_library;
    Dart_Handle result_code = Dart_SetNativeResolver(parent_library, ResolveName, NULL);
    if (Dart_IsError(result_code)) return result_code;
    return Dart_Null();
}

void hello(Dart_NativeArguments arguments) {
    Dart_EnterScope();
    printf("Hello\n");
    Dart_ExitScope();
}

Dart_NativeFunction ResolveName(Dart_Handle name, int argc, bool* auto_setup_scope) {
    if (!Dart_IsString(name) || auto_setup_scope == NULL) return NULL;
    Dart_EnterScope();
    const char *cname;
    Dart_StringToCString(name, &cname);
    Dart_NativeFunction result = NULL;
    if (strcmp(cname, "hello") == 0) result = hello;
    Dart_ExitScope();
    return result;
}
```

👉 深いレベルで拡張可能

## わかりやすく例をもう一個

```
void isEven(Dart_NativeArguments arguments) {  
  Dart_EnterScope();  
  Dart_Handle arg1 = Dart_GetNativeArgument(arguments, 0);  
  int64_t input;  
  if (Dart_IsError(Dart_IntegerToInt64(arg1, &input)))  
  {  
    Dart_ThrowException(Dart_NewStringFromCString("Error だよ"));  
  }  
  Dart_SetReturnValue(arguments, Dart_NewBoolean(input % 2 == 0));  
  Dart_ExitScope();  
}
```

👉 引数と返り値の型情報が静的に定義されていない

**さて、Flutter では？**

**現状、Swift/Objective-C, Kotlin/Java  
を経由する必要がある**

# Support integrating with C/C++ in plugin framework #7053



jtrunick opened this issue on 29 Nov 2016 · 141 comments



jtrunick commented on 29 Nov 2016 • edited by mit-mit ▾



It would be nice to have an example of calling C/C++ code, or at least how to build native code along with a Flutter app. This may purely a Gradle question, but its not clear to someone that's not an expert on Gradle (for example, me), how to pull this off.

Admin comment: Please see [dart-lang/sdk#34452](#) for current status and additional information



553



52



68



14



117



21

Ass



Lab

de

eng

p: 1

plu

sev

**たくさんの  の思いは？**



# ① 既存ソフトをより統合しやすくしてほしい

- **大量のグルーコードがづらい**
- **低オーバーヘッドがいい**

**SQLite**

**Realm**

**OpenCV**

**crypto, ssh ... libraries**

などが具体例として挙げられている

## ② 大量のデータを効率よく出し入れしたい

なお、Dart 2.4 から [TransferableTypedData](#) ができるようになったので、ある程度はそれで間に合いそう

**こういう要望にどう応えるか？**

# ① C++ でメソッドチャンネルを提供する？

FFI の実装が進んでいる背景 > Flutter/Dart における Dart->C をどう実現するか？



**メソッドチャンネルがオーバーヘッド  
高いので、目的に合わない**



## ② Native Exstention でサポートできるようにする？

FFI の実装が進んでいる背景 > Flutter/Dart における Dart->C をどう実現するか？



## 【理由 1】

# 名前ベースの API

```
// dart-lang/sdk/runtime/include/dart_api.h より引用  
DART_EXPORT DART_WARN_UNUSED_RESULT Dart_Handle  
Dart_SetField(Dart_Handle container, Dart_Handle name, Dart_Handle value);
```

- 👉 AOT コンパイラさんには辛い
- 👉 名前解決がキャッシュされない

## 【理由 2】

# Reflective Marshaling は効率良くない

```
void isEmailAddress(Dart_NativeArguments arguments)
```

`void` `arguments` 🙄

⇒ 引数/返り値が静的に型付けされた上での Marshaling の方が効率良い

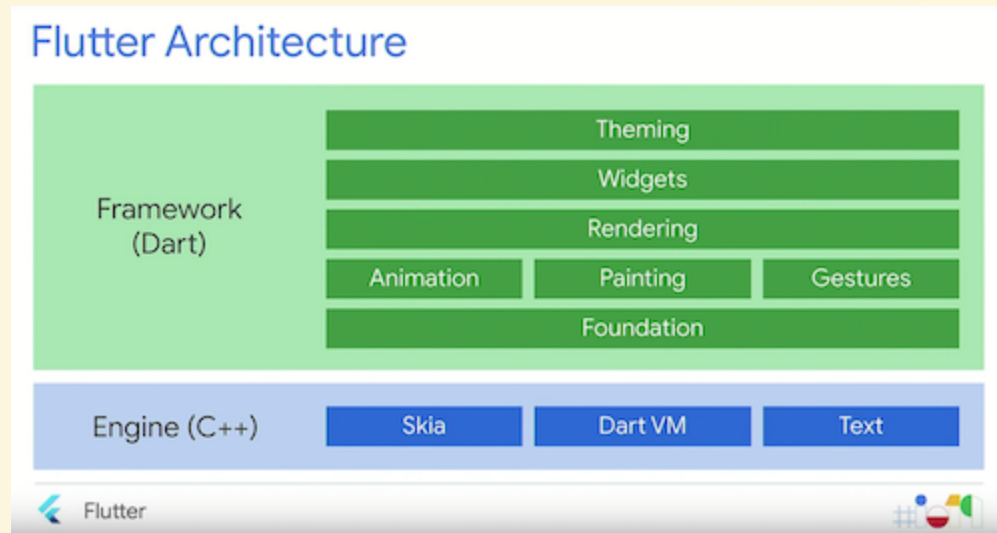
⇒ その点は FFI が優れている 🙌

# Flutter/Dart チームが採った方法は？

**dart : ffi** 👍

<https://github.com/dart-lang/sdk/tree/master/sdk/lib/ffi>

## Google I/O'19 でも言及あり



“

**We are working on a new foreign function interface. This should help you reuse existing C and C++ code, which is important for some critical stuff**

”

## ちなみに

“ we expect that moving Flutter Engine from C API to FFI should significantly reduce overheads associated with crossing the boundary between Dart and native code ”



**結果どう使えるのか？**

```
import "dart:ffi" as ffi;
import 'dart:io' show Platform;

void main() {
  final libHelloWorld = ffi.DynamicLibrary.open(
    "./libHelloWorld.dylib");
  final helloWorld = libHelloWorld.lookupFunction
    <ffi.Void Function(), void Function()>("helloWorld");

  helloWorld();
}
```

[https://github.com/sensuikan1973/Dart\\_FFI\\_Hello\\_World](https://github.com/sensuikan1973/Dart_FFI_Hello_World)

**そして、先週、、、**

# Flutter stable 版に preview が!

(Android だけの試験的なもの)

**どんな感じの構成になるのか**

App Developer	Package Developer			Dart VM Team	Package Developer	Native Library Developer
Flutter App  (Imports package)	Package API  (Does not expose dart:ffi)	Package Implementation  (Code which converts C++ abstractions into Dart abstractions)	Bindings	dart:ffi	Glue code  (Code which takes care of things such as C++ exceptions)	Native Library
Dart					C / C++	

ぜひ dart:ffi に **FB** を送みましょう 🍑

(Dart VM FFI の進行状況は ココ)

**ありがとうございました**



# リンク一覧

- [Dart VM FFI Vision](#)
  - [Introduction to Dart VM](#)
  - [Design and implement Dart VM FFI](#)
  - [Flutter Support integrating with C/C++ in plugin framework](#)
  - [Native extensions for the standalone Dart VM](#)
  - [Support for Dart Extensions](#)
- [C & C++ interop using FFI](#)
  - [sdk/lib/ffi/](#)
  - [Dart Native platform](#)
  - [dart:ffi sqlite sample](#)
- [The Engine architecture](#)
  - [Writing custom platform-specific code](#)
  - [Custom Flutter Engine Embedders](#)
- [Language features for FFI](#)
- [sensuikan1973/flutter-ffi-slide](#)
- [sensuikan1973/Dart FFI Hello World](#)