

# Flutter における FFI

**FFI ?**

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

今回は C++/C の呼び出しの話

# 話すこと

- **Dart, Flutter で FFI どうやるか**
- **(Flutter の) FFI は何が難しいか**

# 自己紹介

サーバ

オセロ



**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
C++	簡単に呼べる。 <code>extern "C"</code> で C++ の名前マングリングを無効にできる。
Go	<a href="#">cgo</a> を使う
Java	<a href="#">JNI</a> や <a href="#">JNA</a> , <a href="#">SWIG</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

# 例: Go -> C

```
package main
```

```
/*
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
void hello() {
```

```
    printf("Hello\n");
```

```
}
```

```
*/
```

```
import "C"
```

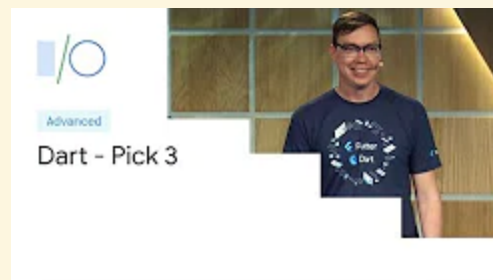
```
func main() {
```

```
    C.hello()
```

```
}
```

**Dart は？**

# 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**

”

**① Native Extension**

**② dart : ffi**

# ① Native Extension



# C++ 側

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "include/dart_api.h"
#include "include/dart_native_api.h"

Dart_NativeFunction ResolveName(Dart_Handle name, int argc, bool* auto_setup_scope);

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;
}
```

## Dart 側

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

参考: [dart-lang sample extension](#)

## ② dart:ffi

“ The extension mechanism discussed in this page is for deep integration of the VM.  
If you just need to call existing code written in C or C++, see [C & C++ interop using FFI](#). ”

引用元: [Native extensions for the standalone Dart VM](#)

```
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)

**さて、Flutter では？**

**dart : ffi のサポートが進んでいる**

# Dart VM FFI VISION について



そもそも FFI の実装で何が難しいの？

Android では `dart:ffi` がすでに使える

## Flutter における FFI の展望

ありがとうございました

## 参考

- [Dart VM FFI Vision](#)
  - [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](#)
  - [Dart Native platform](#)
  - [dart:ffi sqlite sample](#)
  - [sensuikan1973/Dart FFI Hello World](#)
- [The Engine architecture](#)
  - [Writing custom platform-specific code](#)
  - [Custom Flutter Engine Embedders](#)