

# Swiftで! Androidアプリ開発!

# Skip実践入門

## 基礎からリリースまで徹底解説!

yamaken / 山口賢登  
X : @yamakentoc

Skipは、SwiftだけでiOS/Android向けアプリを開発できるツールです。2024年にリリースされ、iOS界隈では度々話題になっています。最近Swift.orgでは、AndroidをSwiftの公式サポートに加えることを目的としたAndroid Workgroupを発足したことによりSkipの注目度が更に高まっています。そんなSkipですが、国内での導入事例は少なく、最新情報を網羅した日本語の解説記事も存在しません。この記事では、筆者のSkip を用いたアプリのリリース経験をもとに、基礎からリリースまでの実践的なTipsをまとめて紹介します。

## Skipの基礎

Skipを使うことでXcodeとSwiftを用いてiOS/Androidアプリを同時に開発できる。具体的にはビルド時にSkipのアレコレを行うことで、コンパイルされたSwiftからKotlin/Javaを呼び出したり、SwiftコードをKotlinにトランスパイルなどすることで実現している。

FlutterやKMPなどマルチプラットフォームツールは様々あるが、Skipは我々が普段から使っているSwift/SwiftUIを用いてOSごとにnativeなUIを提供できることが、他のクロスプラットフォームにない大きな強みである。

## Skipを使ったアプリの作成方法

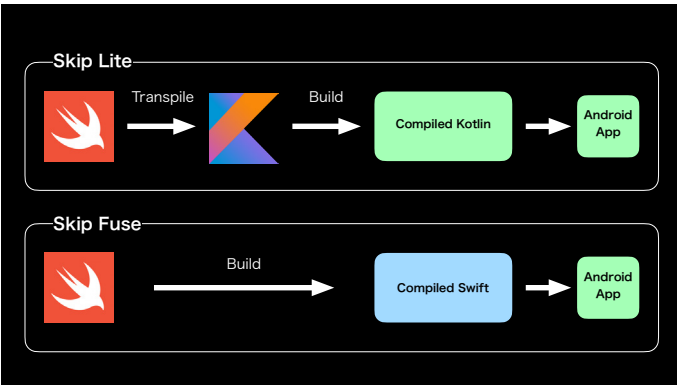
2つの方法がある

- Skipをアプリ全体で使う方法
  - ・ 単一のXcode ProjectでiOS/Androidアプリを作る
- Skipを共通部分でのみ使う方法
  - ・ KMPのようにビジネスロジックだけ共通化するイメージ
  - ・ iOSはXcode, AndroidはAndroid Studioで開発する

この記事では主に前者についてまとめる。

## Mode

SkipにはSkip LiteとSkip Fuseという2つのモードがある。



- **Skip Lite(Transpiled Mode)**
  - ・ SwiftのコードをKotlinにTranspileし、そのKotlinのコードを標準のKotlinコンパイラでビルドするモード
  - ・ v1.0リリース当初はこのモードのみ展開されていた
- **Skip Fuse(Native Mode)**
  - ・ Swift-on-Android toolchainを使用し、コンパイルしたSwiftコードからKotlin/JavaのAPIを呼び出すモード
  - ・ v1.5.0未満まではFull NativeではなくUI部分だけTranspiled Mode、

ロジックはNative Modeだったが、v1.5.0以上からはUIも含めFull Nativeになった

Skip LiteとSkip Fuseでは後者の方が優れてる点が色々あるので推奨されている

## 環境構築

以下は執筆時点(2025年7月時点)で確認している環境

- ・ macOS Sequoia 15.5
- ・ Xcode 16.3
- ・ Android Studio Narwhal | 2025.1.1
- ・ Skip(v1.6.5)

## Skipのinstall

```
# Skipのinstall
$ brew install skiptools/skip/skip

# 開発条件が満たされてるか確認
# エラーが出る場合は`$skip checkup --verbose`で詳細を表示
$ skip checkup

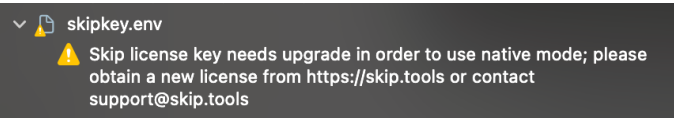
# swift android toolchainのinstall
$ skip android sdk install

# Native Modeで開発条件が満たされてるか確認
# Nativeが後発のため分かれてる)
$ skip checkup --native
```

## License Keyの発行

Skipをinstallしてから30日間はLicense Keyなしでビルドできる。それ以降はLicense Keyが必要になるので以下の手順を踏む。

License Keyを発行するにはいくつかプランがあり、個人で開発する分にはIndieで問題ない。そしてこのIndieは無料で制限なく使える…というのが少し前までの話。今は無料で使えるには変わらないが、IndieだとSkip Liteしか使えず、Skip Fuseが使えなくなってしまった。ただ、この変更が行われる前にIndieで発行したKeyでもWarningは発生するがビルド自体は執筆時点でできている。



次のコマンドでHost Identifierを取得する。

```
$ skip hostid
ABCDEFGH-IJKL ...
```

公式サイトのPricingページからプランを選択し、Host Identifierを送信することで、メールでLicense Keyが送られる。

<https://skip.tools/purchase/indie/>

そしてLicenseKeyを~/.skiptools/skipkey.envに入力することで完了。

```
# Obtain a Skip key from https://skip.tools for the SKIPKEY property
# Be sure that the key is on a single line, and that
# there is a space between the colon and the key string
SKIPKEY: ABCDEFGH....
```

## Skipアプリの作成

次のようなコマンドでプロジェクトを作成する。

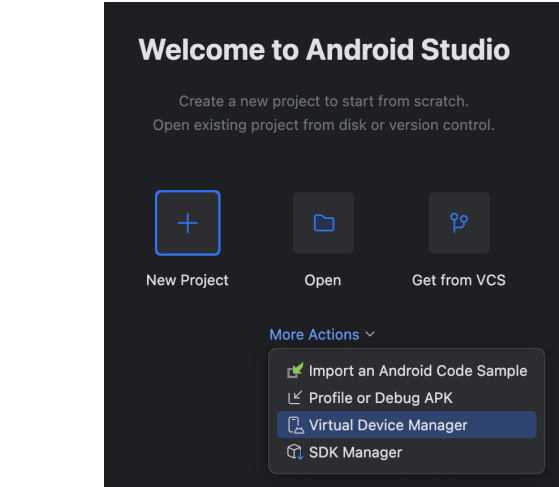
```
$ skip init --open-xcode --native-app \
--appid=bundle.id.HelloSkip hello-skip HelloSkip
```

各指定は以下のようになっている。

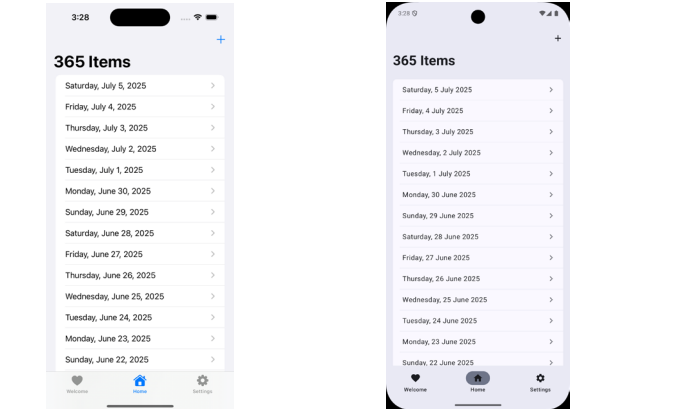
```
# --native-app: Skip FuseのProjectを作成。
# Skip Liteの場合は--transpiled-appを指定
# bundle.id: アプリのBundle IDを指定
# project-name: SwiftPMのPackage名 (Rootディレクトリ名) を指定
# AppName: アプリ名 (.xcworkspace名) を指定
$ skip init --open-xcode --native-app --appid=bundle.id \
project-name AppName
```

プロジェクトの作成に成功するとXcodeが開く。

Buildする前にAndroidのエミュレータを立ち上げる必要があるのでAndroid StudioでVirtual Device Managerを起動し、適当なエミュレータを起動させる。



その後XcodeでRunすると、AndroidとiOSでtemplateアプリが起動する。



## マルチモジュールアプリの作成

Skipではマルチモジュール構成のアプリの利用が推奨されている。が、前述したコマンドで作成される単一モジュール構成の方がクセが少ないので、まずはそちらを利用するのが良さそう。

以下のコマンドでは、HelloSkip, HelloModel, HelloCoreの3つのモジュールが生成される。

```
$ skip init --native-app --appid=bundle.id.HelloSkip \
multi-project HelloSkip HelloModel HelloCore
```

## 既存アプリの移行

やるとしたらskip initでSkip向けの様々な設定がされたProjectを作成したのちに、コードを少しずつコピペする形が現実的。既存のXcode Projectを直接移行するのは無理に近いと思う。

## Skipの設定

### iOSだけビルドする

SkipプロジェクトではBuild Phaselにgradle build用のコマンドが入っているため、Buildする度にAndroid側でアプリが起動される。


Build時のAndroid側の挙動を変更するには.xcconfigファイルの指定を変更する

```
8 // Xcodeのビルド時のAndroid側の挙動を設定
9 SKIP_ACTION = launch // Androidのビルドと実行を行う
10 //SKIP_ACTION = build // Androidのビルドのみを行う
11 //SKIP_ACTION = none // Androidのビルドも実行もしない
```

## Build Settings

Skipプロジェクトでの設定は主に.envファイルと.xcconfigファイルを通じて行う。これらはXcodeの機能として存在し、次の役割を持つ。

- ・ .envファイル
  - ・ 環境変数を定義するためのファイル
  - ・ 主にgitで管理したくないような値をまとめるために使う

 **なぜSkipが いいの？**

Flutter・KMP・React Nativeなどマルチプラットフォームツールは様々あるが、普段から使い慣れているSwift/SwiftUI/Xcodeを使いつつ、OSごとにネイティブなUIを表現できるのがSkipの強み。純粋にSwiftを扱っているので、iOSのアップデートなどにもすぐに適用できる。実際WWDC25で発表されたLiquid GlassにはDay1 対応されていた。Document1には具体的な比較表まで存在するので参考にしてみるといい。

- ・xcconfigファイル
  - ・Xcodeのビルド設定 (Build Settings) をテキストで指定するもの
- ・Build Settingsから設定すると.xcodeproj内に設定値が定義されるが、xcconfigならテキストで見れて管理が楽

Skipでは通常、.envファイルにアプリ名やversionなどを指定し、.xcconfigとAndroidManifest.xmlの両方から参照するようになっている。なので共通の環境変数はここに定義するのがいい。

しかしデフォルトだとPRODUCT\_NAMEしかAndroidManifest.xmlから参照されてないので、必要に応じて呼び出してあげるとよい。

```

HelloSkip ) Skip.env ) No Selection
7 // アプリの名前。Swift Moduleの名前と同じにする必要がある。
8 PRODUCT_NAME = HelloSkip
9
10 // Bundle ID
11 PRODUCT_BUNDLE_IDENTIFIER = bundle.id.HelloSkip
12
13 // semantic version
14 MARKETING_VERSION = 0.0.1
15
16 // アプリのバージョン
17 CURRENT_PROJECT_VERSION = 1
18
19 // Androidのエントリポイントのパッケージ名。
20 ANDROID_PACKAGE_NAME = hello.skip
```

.envとxcconfigを使うように推奨されてるが直接Build Settingsに指定するのでも動作には問題ない。値が重複しないように気をつけよう。

## Modeの確認

このプロジェクトがSkip FuseかSkip LiteなのかはSources/module名/Skip/skip.ymlに定義されている。

Modeを基本的に途中で切り替えるのは不可能。というのもinit時にnativeかtranspileか指定するが、それによってプロジェクトの設定なども諸々変わってるため、指定を変えてビルドしてもエラーが発生する。

```
skip:
  mode: 'native' # 'transpiled'
```

## フラグ系

AndroidとiOSで実行する処理を純粋に切り替えたい場合は以下のようにする。

```

#if os(Android)

Text("Android側の処理")
#else

Text("iOS側の処理")
#endif
```

これらはSwiftUIのmodifier間でも使用できる。

```
Text("Hello World")
    .italic()
    #else
    .bold()
    #endif
```

### Skipフラグ

Skipの魅力はSwiftだけでAndroidアプリをビルドできることだが、実現したい機能によってはSwiftからKotlinやJavaのAPIを呼び出す必要がある。これはSkipフラグを使うことで実現できる。

```

func printFormatted(time timeInMills: Int64) {
    #if os(Android)
    let formatted = androidTimeString(milliseconds: timeInMills)
    #else
    let formatted = ... iOS code path ...
    #endif
    print(formatted)
}

#if SKIP
func androidTimeString(milliseconds: Int64) -> String {
    let dateFormat = java.text.SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'",
        java.util.Locale.getDefault())
    dateFormat.timeZone = java.util.TimeZone.getTimeZone("GMT")
    return dateFormat.format(java.util.Date(milliseconds))
}
#endif
```

上記の例ではAndroid側の処理として、SwiftからJavaのAPIを呼び出すメソッドを呼び出している。

#if SKIP内に定義されたコードは全てTranspileされる。

Skip Liteを使用している場合、全てのSwiftコードはTranspileされるため、#if SKIPと#if os(Android)は同等の意味となる

### Debugフラグはどうする？

Android側では#if DEBUGとしても常にfalseが返る。そのため以下のようにして判定する。

```

let isDebugBuild: Bool = {
    #if SKIP
    return
        (ProcessInfo.processInfo.androidContext.getApplicationInfo()
        .flags & android.content.pm.ApplicationInfo.FLAG_DEBUGGABLE) != 0
    #elseif DEBUG
    return true
    #else
    return false
    #endif
}()
```

### canImport

iOSだけで使用するmoduleをimportする際、#if os(iOS)のようにしてもいいが、canImportも活用できる。

```

#if canImport(Uikit)
import UIKit
#endif
```

## Composeを呼び出す

SkipではSwiftUIからComposeのAPIを呼び出したり、ComposeからSwiftUIのAPIを呼び出したりをシームレスにできる。これはSwiftUIで定義したコードがAndroid側だと挙動が異なる場合に使うことが多いイメージ。

```

import SwiftUI

struct SampleView: View {
    var body: some View {
        VStack {
            #if os(Android)
            ComposeView {
                SampleCompose(text: "Android")
            }
            Text("Hello, Android")
            #else
            Text("Hello, iOS")
            #endif
        }
    }
}

#if SKIP
// ContentComposerに準拠したstructを定義
// #if SKIP内のコードなのでTranspileされる
struct SampleCompose: ContentComposer {
    let text: String

    @Composable func Compose(context: ComposeContext) {
        androidx.compose.material3.Text("Hello \(text)",
            modifier: context.modifier)
    }
}
#endif
```

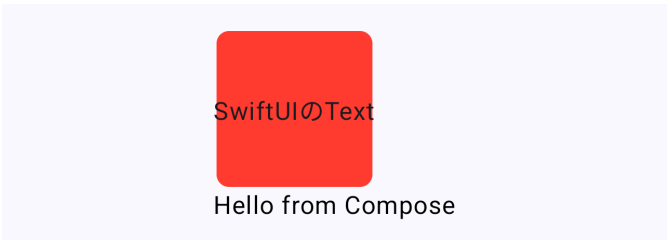
Compose内からSwiftUIを呼び出す場合は.Compose(context:)を使用することで組み合わせることができる。

```

#if SKIP
import androidx.compose.foundation.layout.Column

struct SampleCompose: ContentComposer {
    @Composable func Compose(context: ComposeContext) {
        Column(modifier: context.modifier) {
            ZStack { // SwiftUIのZStack
                RoundedRectangle(cornerRadius: 8)
                    .fill(Color.red)
                    .frame(width: 100, height: 100)
                Text("SwiftUIのText")
            }
            // ComposeとSwiftUIを組み合わせる
            .Compose(context: context.content())
            androidx.compose.material3.Text("Hello from Compose")
        }
    }
}
#endif
```

↓ Android側での実行結果



Composeのmodifierを呼び出すこともできるが、それらについてはQiitaの記事「SkipでAndroidのコードを呼び出す方法まとめ」に記載しているので参考にしてください。

### Context・Activityの取得

Kotlinで定義したメソッドなどを呼ぶ出す際、ContextやActivityが必要となる場合がよくある。Projectのモードに合わせて以下のように取得できる。Skip Fuseの場合：

```

#if os(Android)
// contextの取得
let applicationContext = ProcessInfo.processInfo.dynamicAndroidContext()
if let packageName: String = try? applicationContext.getPackageName() {
}

// activityの取得
if let activity = UIApplication.shared.dynamicAndroidActivity() {
}
#endif
```

このようなAndroidのオブジェクトはAnyDynamicObjectType型(Kotlin/Javaのあらゆる型を表現する型)として返ってくるため、必要に応じてcastする必要がある。

Skip Liteの場合：

```

#if SKIP
// contextの取得
let applicationContext = ProcessInfo.processInfo.androidContext
let packageName = applicationContext.getPackageName()

// activityの取得
if let androidActivity: androidx.appcompat.app.AppCompatActivity =
    UIApplication.shared.androidActivity {
    let intent = androidActivity.getIntent()
}
#endif
```

### 💡 実際にSkipを使って感じる課題は？

特にビルド速度とAPIのサポート周りはもう少し改善することを期待している。ビルドについては、キャッシュがない場合はSkip自体のビルドやGradleに関する依存のダウンロードなども行われるので仕方がないが、それ以外でも長く待つことが多く感じる。APIについては地味に使いたいAPIがサポートされてなかったりする。しかし頻繁にアップデートがされているのでいつの間にかサポートされてることもあったりする。

## 便利framework：SkipKit

Skipを使うことでSwiftだけでアレコレできるといっても、プラットフォーム固有のものまで全ては網羅できない。例えばカメラ機能などを実装しようにもSkipではUIKitをほぼサポートしてないため、UIImagePickerControllerやAVFoundationなどはiOS側だけの使用でないとエラーになる。Kotlinで実装するのも大変なのでSkipKitというframeworkを使用する。

### カメラの実装

```

import SkipKit

struct MediaButton : View {
    let type: MediaPickerType // .camera or .library
    @Binding var selectedImageURL: URL?
    @State private var showPicker = false

    var body: some View {
        Button(type == .camera ? "Take Photo" : "Select Media") {
            showPicker = true // activate the media picker
        }
        .withMediaPicker(type: .camera, isPresented: $showPicker,
            selectedImageURL: $selectedImageURL)
    }
}
```

これだけで両OS共通してカメラやフォトライブラリを起動できる。

### 許諾

SkipKitではiOS/Android両方の許諾状態の取得とリクエストを行うこともできる。

```

import SkipKit
/// カメラの権限をリクエストする
static func requestAndroidCameraPermission() async -> Bool {
    // 現在の権限の状態を確認
    let currentPermissionStatus =
        PermissionManager.queryPermission(.CAMERA)
    // 権限が不明または拒否されている場合はリクエスト
    if currentPermissionStatus == PermissionAuthorization.unknown ||
        currentPermissionStatus == PermissionAuthorization.denied {
        let permissionStatus = await
            PermissionManager.requestPermission(.CAMERA)
        switch permissionStatus {
            case .authorized, .limited:
                // 権限が許可された場合
                return true
            case .denied, .restricted:
                // 権限が拒否された場合
                return false
            case .unknown:
                // 権限の状態が不明な場合
                return false
        }
    } else if currentPermissionStatus ==
        PermissionAuthorization.authorized ||
        currentPermissionStatus ==
        PermissionAuthorization.limited {
        // すでに権限がある場合
        return true
    }
    return false
}
```



許諾ダイアログに表示する文言は.xcconfigに記載する。

```
// カメラにアクセスするためのKey
INFOPLIST_KEY_NSCameraUsageDescription = "Allow MyFitsLog to
access your camera to take outfit photos.";

// PhotoライブラリにアクセスするためのKey
INFOPLIST_KEY_NSPhotoLibraryUsageDescription = "Allow MyFitsLog
to access your photo library to save outfit photos.";
```

Android側はAndroid/app/src/main/AndroidManifest.xmlにカメラを使用することを記載する

```
<!-- カメラの使用権限 -->
<uses-permission android:name="android.permission.CAMERA" />
```

また、Android/app/src/main/res/xmlfile\_paths.xmlを作成して次の内容を記載する必要がある。

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-path name="my_images" path="." />
    <cache-path name="*" path="." />
</paths>
```

Skipの便利なframeworkは他にも色々ある。例えば位置情報を取得するならSkipDevice、Lottieのanimationを使用するならSkipMotionなど。

## 画像の表示

両OS共通で使用するのは、Sources/ModuleName/Resources/に配置する。Resources/には予めLocalizable.xcstringsとModule.xcassetsが用意されている。

### 通常の画像

自身で用意した画像を表示する場合は、Module.xcassetsに画像を配置して以下のように指定する。

```
// bundleを指定して画像を表示
Image("yamakentoc.icon", bundle: .module)
    .resizable()
    .frame(width: 200, height: 200)
```

### SF Symbols

SF Symbolsを使用する場合は通常通り指定する。

Android側にはSF Symbolsと同じアイコンが表示されるわけではなく、material.Iconsから同等のアイコンを表示してくれる。ただ、全てのアイコンを網羅してるわけでもないの、どのアイコンが対応してるかはDocumentを確認した方がいい。

<https://skip.tools/docs/modules/skip-ui/#fallback-symbols>

```
// SF Symbolsを使用 (material.Iconsから同等のアイコンを表示)
Image(systemName: "checkmark.circle")
```

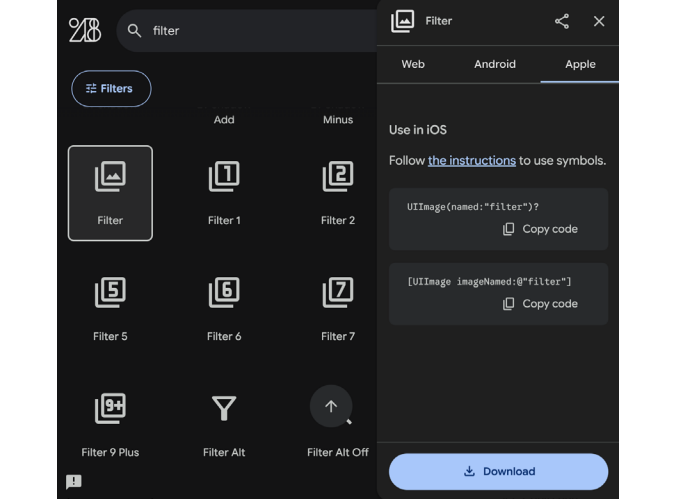
### Material IconsをDownloadする

同等のmaterial.Iconsがない場合、そのまま実行するとAndroid側ではWarningのアイコンが表示されてしまう。

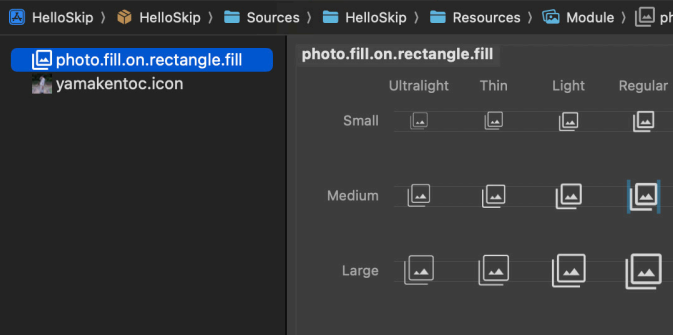
```
// 同等のmaterial.Iconsがない場合はAndroidで警告アイコンを表示
Image(systemName: "arrow.up")
```

この場合は.xcassetsに同名のアイコンを用意すればよい。アイコンはGoogleのMaterial Symbols and Iconsというサイトから似たようなアイコンを探し、DownLoadして.xcassetsにセットする。

例えば写真が重なっているようなSF Symbolsである`photo.fill.on.rectangle.fill`と同等のアイコンを用意する場合、似たようなアイコンを探し、AppleのTabからDownloadする。



Downloadしたファイルをそのまま.xcassetsにセットし、名前をSF Symbolsと同じ名前にすることで、iOSは通常のSF Symbolsのアイコンが表示され、Android側ではDownloadしたアイコンが表示される。



```
// Android用のMaterialアイコンをセットする場合
Image(systemName: "photo.fill.on.rectangle.fill")
```

### SF SymbolsをDownloadする

SF SymbolsアプリからアイコンをDownloadしてAndroid側にも全く同じアイコンを表示することも可能。しかし、SF SymbolsをAppleのプラットフォームを超えて使用するのは権利的にOKなのか?という疑問があるので正直使っていいか怪しいと個人的に思っている。とはいえSkipのドキュメントにはそうもできると書いてるので一応方法も示す。

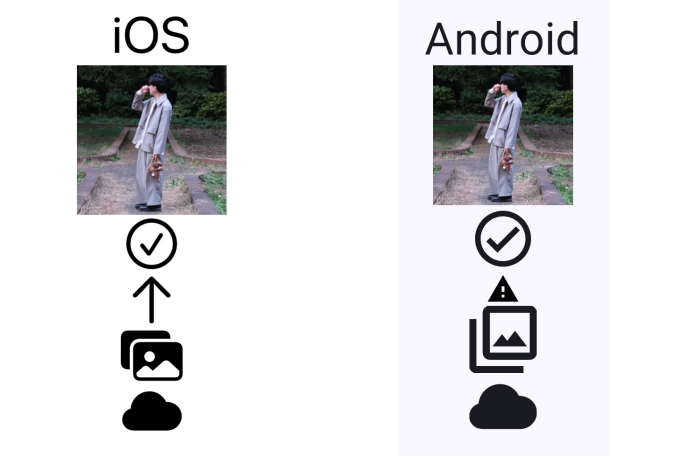
macOSのSF Symbolsアプリから対象のアイコンを選択してファイル→「シンボルを書き出す」で.svgをDownloadし、.xcassetsに入ればOK。

これでiOSもAndroidも全く同じアイコンで表示される。

しかし我々やiPhoneユーザにとって馴染み深いアイコンであっても、それはAndroidユーザにとってもそうか?というのは疑問に持って欲しい。

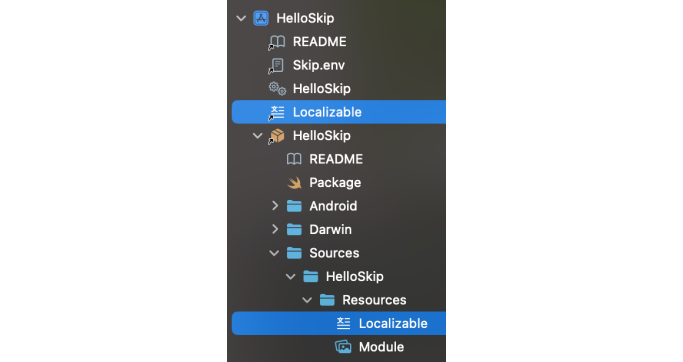


ここまでの画像の実行結果を載せておく。



## 多言語対応

SkipプロジェクトではデフォルトでString Catalogsが設定されており、今まで通りクセなく使える。もちろんAndroid側にも対応される。



XcodeのNavigatorにはLocalizable.xcstringsなどが2つ存在しているように見えるが、これは単にPackage内に定義されているLocalizable.xcstringsをProjectのRootでも参照してるだけなのでどちらを変更しても問題なし。

## framework

前項で説明した通り、SkipにはSkip専用のframeworkが様々存在する。例えば以下はプロジェクト生成時から存在する一例。

- skip-stone
  - skipの心臓部。ビルド時にBridgeを生成したりTranspileするなど
- skip-foundation
  - Skipで使用できるFoundation系のAPIが定義されている
- skip-ui
  - Skipで使用できるSwiftUIのAPIが定義されている。主にTranspiled Modeで使用

### frameworkの追加

Skipに関係なく通常通りframeworkを追加できる。が、それがAndroidに対応してないものだと#if os(iOS)などで分けるのはもちろん必要。

**💡 実際どのくらいAndroid用のコードを書いてる?**

私がリリースしているSkipのアプリでは、おそらく全体の5%ほど。例えばLazyVStackとImageを組み合わせてGrid状に画像を表示しているが、どうにもAndroid側だとスクロール時に重くなってしまう現象がある。そのため、Androidの場合のみComposeで定義をしている。それ以外にも画像の最適化などをKotlinで定義している。

例えばfirebaseについてはまだ対応されていないので直接は使用できないが、代わりにSkipFirebaseが用意されているのでこちらを使用する。(SkipFirebaseはSkip FuseとSkip Liteで実装方法が異なるので注意) 何がAndroidでも使えるかはswift-everywhere.orgで確認できる。

## Debug

iOS側のDebugについてはXcode側のDebug機能をフルで使える。Android固有の問題についてはAndroid Studio側から直接Androidアプリを実行し、Android Studio側のDebug機能を使えるらしいが、私の環境では確認できず...

### ログ

基本的にOSLogを使用する。それによりAndroid StudioのLogcatで確認できる。量が多いので基本的に指定した引数の値でfilterした方がいい。

```
// Skip Liteの場合はOSLogではなくSkipFuseをimport
import SkipFuse
...
let logger = Logger(subsystem: "my.subsystem", category: "MyCategory")
logger.info("\(logString)")
```

### よくあるエラー

```
No connected Android devices or emulators were reported by
`adb devices`. To launch the Skip app, start an emulator from
the Android Studio Device Manager or use the
~/Library/Android/sdk/emulator/emulator command
```

↑エミュレータ or 実機が起動してないのが原因。起動すると解決。

```
com.android.ddmlib.InstallException:
INSTALL_FAILED_INSUFFICIENT_STORAGE: Failed to override
installation location
```

↑容量が足りなく上書きできないと言ってるが、エミュレータで起動してるアプリを終了するかuninstallするかで解決する。それでも解決しない場合はWipe Dataを行うことで解決するはず。

```
import SwiftUI
#if SKIP
import androidx.compose.ui.graphics.Color
#endif

struct SampleView: View {
    var body: some View {
        #if Skip
        ComposeView { context in
            Color.red
        }
        #else
        Color.red
        #endif
    }
}
```

↑どちらのAPIか判別できない系。

SwiftUI側のColorとandroidx.compose.ui.graphics側のColorとで参照が混ざってるのが原因。SwiftUI.Color.redとするか、以下のように直接androidx側のColorを参照すれば解決。



```
import SwiftUI

struct SampleView: View {
    var body: some View {
        #if Skip
            ComposeView { context in
                androidx.compose.ui.graphics
                    .Color.red
            }
        #else
            Color.red
        #endif
    }
}
```

## リリース系

リリースに関連することをまとめる。

### アプリアイコン

iOSのアイコンは通常通りApp/Assets.xcassetsに設定する。

AndroidはAndroid/app/src/main/res/に各サイズごとに配置する必要があるが大変なので、次のコマンドを使用するのもよい。1つのpngやSVGからAndroidに必要なアイコンサイズを全て生成してくれる。

```
$ skip icon --android path/to/icon.png
```

先に1024\*1024のiOS側のアイコンを作成してそれを使うのが良さそう。一応iOS側のアイコンも生成できるが、現在のiOSではライト用/ダーク用で設定するだけなので基本不要だと思う。

### アーカイブの準備

Skipプロジェクトでは予めfastlaneが組み込まれているため、基本的にfastlane経由でアーカイブとアップロードを行う。iOSについてはXcodeから直接アーカイブできる。(https://skip.tools/docs/deployment/)
Androidの場合はGoogle Play ConsoleでDeveloperアカウント発行後、以下の手順を行う。

#### • apiKey.jsonの発行

これは一般的なことなのでfastlaneのドキュメントを参考に行う。

https://docs.fastlane.tools/actions/upload\_to\_play\_store/#setup
発行したjsonをapiKey.jsonという名前に変え、Android/fastlane/に配置する。通常だとAppfileにjsonのpathを指定する必要があるが、元々指定がされてるのでこのままでOK。

#### • keystore.jksの生成

Android/app/で次のコマンドを叩く。

```
$ keytool -genkeypair -v -keystore keystore.jks -alias release -keyalg RSA -keysize 2048 -validity 10000
```

色々聞かれるがpassword以外適当でOK。

#### • keystore.propertiesの作成

Android/app/に以下の内容で手動で作成する。

```
storePassword=<keystore.jks作成時のpassword>
keyPassword=<keystore.jks作成時のpassword>
keyAlias=release
storeFile=keystore.jks
```

本来ならapiKey.json, keystore.jks, keystore.propertiesは.gitignoreに定義しておくべきことだが、これも元々定義されてるので問題ない。

### アーカイブとアップロード

準備ができればAndroid/でコマンドを叩く。

```
$fastlane release
...
[19:10:47]: fastlane.tools finished successfully 🚀
```

おそらく最初はうまくいかず、以下のエラーで躓くので対応する。

### fastlaneコマンドでのエラー

#### • local.propertiesのsdk.dirのpathが読み取れない

→ Androidの環境変数が.zshrcなどに定義されていない可能性が高いので要チェック。

#### • Package not found: (Bundle ID)

→ 最初は手動で.aabをアップロードする必要があるらしい。

以下のpathに.aabがあるので、まずはこれをアップロードする。

ProjectのRoot/.build/Android/app/outputs/bundle/release/app-release.aab

#### • Only releases with status draft may be created on draft app.

→ まだ初回のリリースができてないdraft状態のため？発生。

Android側のFastfileを以下のように変えることでdraft状態でも配信可能になる。

```
upload_to_play_store(
  aab: '../.build/Android/app/outputs/bundle/release/app-release.aab',
  track: 'internal',          # 内部テストトラックへ配信
  release_status: 'draft'    # ドラフトリリースとして登録
)
```

### Release版でのみクラッシュ

Skipの最新versionでは発生しない可能性が高いが、少し前のversionで発生していた、Release版でのみ起動時にクラッシュする問題についてはこちらで取り上げる。

Androidではbuild.gradle.ktsでisMinifyEnabled = trueという指定をすると、コードの最適化・難読化が行われ、不要なコードの削除なども行われるようになる。

```
buildTypes {
    release {
        signingConfig = signingConfigs.findByName("release")
        isMinifyEnabled = true // 最適化を行う指定
        isShrinkResources = true
    }
}
```

これにより不要なものが削除されるが、少し前のversionでは必要なもので削除されていた。

### AI Coding ToolでSkipを使う時のTips

• Claude Codeなどが書いたコードがSkipでサポートされてるか

不明なことが多いので、実装後に必ずビルドしてもらう

• ビルドに時間がかかることが多いのでタイムアウトを長めにする

• ビルドはXcodeBuildMCPなどを使うとエラー時の詳細を見れないので、ビルドのみは通常のxcode buildコマンドを使用する

• SkipのDocumentを渡してAndroid側の書き方など参照させる

• たまにエラーになった時、なぜかskip checkup --nativeを行なってtokenを大量に消費してしまうので禁止させる

具体的にはRelease版の起動時に以下のエラーが吐かれていた。

```
SkipBridge/AnyDynamicObject.swift:222: Fatal error:
'try!' expression unexpectedly raised an error:
java.lang.ClassNotFoundException: Didn't find class
"tools.skip.bridge.Reflector" on path: ~~~
```

tools.skip.bridge.Reflectorが見つからないと言われている。

最適化時には削除しないものを指定するproguard-rules.proというファイルが存在する。ここをよく見ると前のversionでは以下のような指定であったが、tools.skip.bridge.Reflectorのような指定はない。

```
-keeppackageNames **
-keep class skip.** { *; }
-keep class kotlin.jvm.functions.** {*; }
-keep class com.sun.jna.** { *; }
-keep class * implements com.sun.jna.** { *; }
-keep class outfit.management.** { *; }
```

最新のSkipのversionで新規でプロジェクトを立ち上げると追加で以下の指定が入っていたので、これを追加してみると無事にクラッシュを回避できた。

```
-keep class tools.skip.** { *; }
-dontwarn java.awt.**
-keep class * implements skip.bridge.** { *; }
-keep class **._ModuleBundleAccessor_* { *; }
```

Release版でのみクラッシュが発生する場合はproguard-rules.proの内容をまず疑ってみるのがよい。最悪isMinifyEnabledをfalseにすれば解消されるが最適化が行われずにAPKのサイズが大きくなったり、逆コンパイルされる可能性も上がるので推奨されてないようだ。

## Android版のリリースについて

ここまで読んでいる方はSkipを実際に使ってアプリをリリースしたいという気持ちがある方でしょう。素晴らしいです！しかし、リリースにあたってGoogleさんの試練を乗り越えないといけません。

まず、AndroidアプリをリリースするにはGoogle Play ConsoleでDeveloperアカウントを作成する必要があります。Developerアカウントには、組織アカウントと個人アカウントの2種類があります。

2023年11月以降に個人アカウントを作成した場合、12人以上のテスターを集め、インストールして14日以上アンインストールせずに端末に残す条件が追加されました。また、そのアプリで課金や広告が発生する場合、アカウントに登録した開発者の住所がPlay Storeで公開されるようになります。

この2つの条件が厳しいので、現状Androidアプリを個人でリリースするには個人事業主として開業し、組織アカウントを取得するのがベストかと思います。収益化せずAndroid端末を持つてる知り合いがたくさんいれば問題ないのですが…私の場合は思い切って開業する道を選びました。

### 組織アカウントを発行する場合

私の場合はfreee開業を使ってオンラインで開業届を提出しました。そして事業用の住所としてDMMバーチャルオフィスというサービスを使い、青山というリッチな場所の住所を得ました。

申請時に屋号は絶対つけましよう。組織アカウントを発行する際、DUNS Numberという企業の識別子が必要になるのですが、その申請をするのに屋号が必須となります。私の場合最初の提出時に屋号を付けなかったため、税務署に更新用として再度提出しに行きました…。DUNS Numberは申請してから1週間後に発行完了のメールが届きました。発行まで時間がかかるようなので、なるべく早めに申請するのを勧めます。

## その他

### 開発で参考になるもの

#### • Skip Showcase

• ShowcaseアプリはApp StoreやPlay Storeに公開されているかつオープンソースなので、UIコンポーネントごとの挙動などを確かめるには便利なのでおすすめ

#### • 公式のサンプルアプリ集

• SkipのDocumentを見るとSkip Fuse, Skip Liteごとにサンプルアプリを多く紹介してるので色々見てみるといい

#### • fromkk/PhotoExhibition-skip

• かつくん(@fromkk)さんがSkipを用いてリリースした写真展示アプリ「exhivision」のソースコード

• 私自身もめちゃくちゃ参考にさせていただきました！

https://github.com/fromkk/PhotoExhibition-skip/

### SkipのSlackに入るう

Skipを使っている以上、どこかしらで躓く可能性が高いです。謎のエラーやAndroid側での不具合などに遭遇することでしょう。そんな時はSkipのSlackのチャンネルに入り、気軽に質問してみるとSkipの創設者の方を中心に回答していただけます。

また、Skipの進化が凄まじく、知らない間に色々アップデートされ、前までできなかったことがいつの間にかできるようになってたりしています。そういった最新情報を得るためにも、Skipを使って何かアプリを作ろうとしている方は是非コミュニティのSlackへ入ってみましょう。

https://skip.tools/slack/

## 最後に

私はこのSkipを用いて、コーデ写真管理アプリ「MyFitsLog」というアプリをリリースしました。執筆時点ではiOSアプリ版のみリリースしていますが、この記事が公開されてる頃にはAndroid版もリリースされてると思います。実装の詳しい部分なども共有できますので、もし興味ある方はiOSDC当日に声かけたり、XにDM投げたりしてくださいmm



MyFitsLog



App Store用リンク

### この記事を書いた人



yamaken / 山口賢登 /@yamakentoc

「Apple Store」で検索するとりんごの直売所がヒットする青森県で1997年に生まれる。大学に入ってからiOSアプリ開発に目覚め、個人アプリをリリース。気づけば社会人6年目で最近はiOSアプリ開発者兼ScMを担当。毎年のWWDCで発表される新しい技術のキャッチアップや、ASOなどが好き。クルクルな髪とコミュ力を取り柄。前年度は周りにいる赤い髪の人や非常口な方からの圧を感じて登壇しまくるように。try! Swift Tokyo 2025でもSkipのトークで登壇。