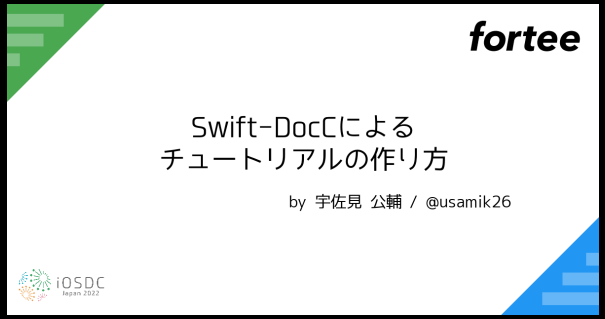
SwiftPMのプラグイン機能を iOSアプリ開発に活用する

宇佐見公輔 / 株式会社ゆめみ

自己紹介

- 宇佐見公輔（うさみこうすけ）
- 株式会社ゆめみ / iOSテックリード
- このトーク以外にも、パンフレット記事を2つ書きました。



このトークの内容

- SwiftPM（Swift Package Manager）とは
- iOSアプリ開発でSwiftPMを活用する
- SwiftPMのプラグイン機能とは
- iOSアプリ開発でSwiftPMプラグインを活用する

※ Swift Package Managerを略してSwiftPMと呼ぶことにする。

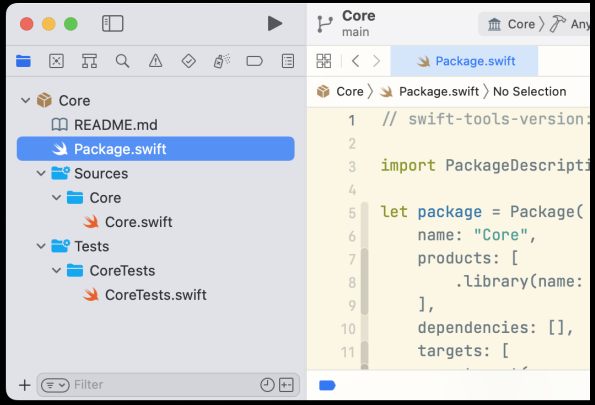
Swift Package Managerとは

Swift Package Managerとは

- Swiftコードをパッケージとして管理する
- パッケージをビルドしてライブラリや実行プログラムを生成する
  - ライブラリ：他のSwiftコードでインポートできるモジュール
  - 実行プログラム：シェル上で実行できるCLIツールなど
- 他のパッケージを依存物として利用できる

パッケージ

- パッケージはSwiftソースファイルと `Package.swift` で構成される



Package.swift

```
import PackageDescription

let package = Package(
    name: "MyLibrary",
    products: [
        .library(name: "MyLibrary", targets: ["MyLibrary"]),
    ],
    dependencies: [],
    targets: [
        .target(name: "MyLibrary", dependencies: []),
    ]
)
```

配布されているパッケージの利用

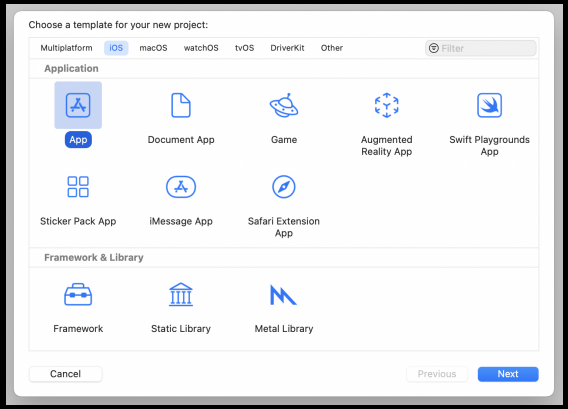
```
import PackageDescription

let package = Package(
    name: "MyLibrary",
    products: [
        .library(name: "MyLibrary", targets: ["MyLibrary"]),
    ],
    dependencies: [
        .package(url: "https://example.com/AwesomePackage", from: "1.0.0"),
    ],
    targets: [
        .target(name: "MyLibrary", dependencies: []),
    ]
)
```

iOSアプリ開発で
SwiftPMを活用する

Xcodeプロジェクト

- iOSアプリはXcodeプロジェクトを使って開発する



余談：Swift Playgrounds App

- Swift Playgroundsでも開発可能、プロジェクト形式が異なる
- 「ゆめみ大技林 '22」に書いた（技術書典で配布）

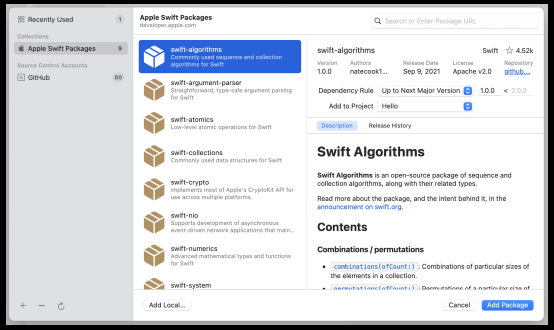
※ これも面白いが、このトークではこれ以上述べない。

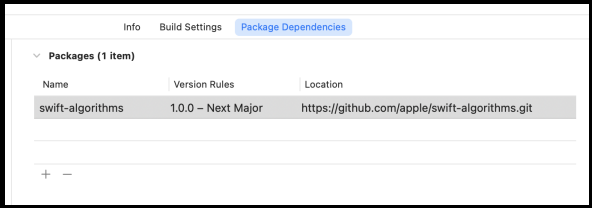
XcodeプロジェクトとSwiftPM

- 配布されているパッケージを利用する
  - CocoaPodsやCarthageで配布ライブラリを利用する代わりに、SwiftPMで配布ライブラリを利用する
- アプリのコード（の一部）をパッケージ化する
  - コードをXcodeプロジェクトの管理外に置ける

Xcodeで配布パッケージを利用する

- Xcodeの「File→Add Packages...」で依存パッケージを追加できる

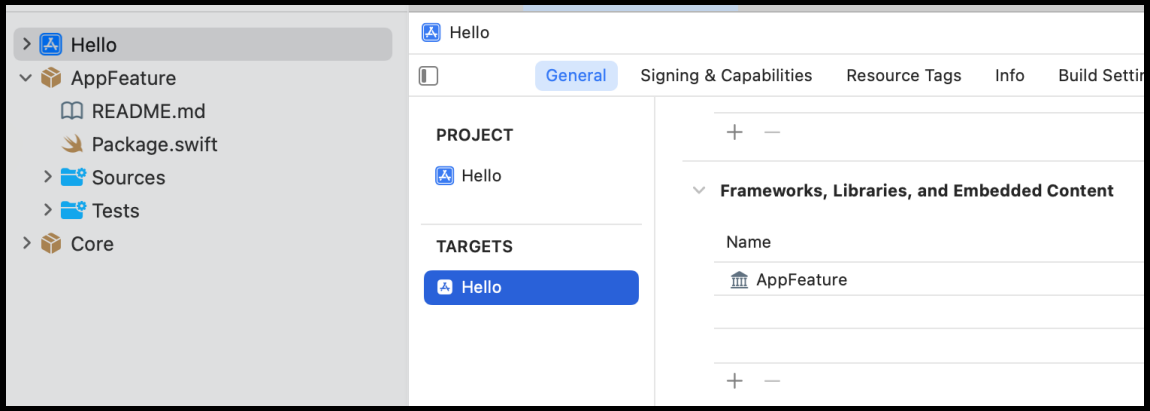




※ ライブラリ管理の手法として有益だが、このトークではこれ以上述べない。

アプリのコードをパッケージ化する

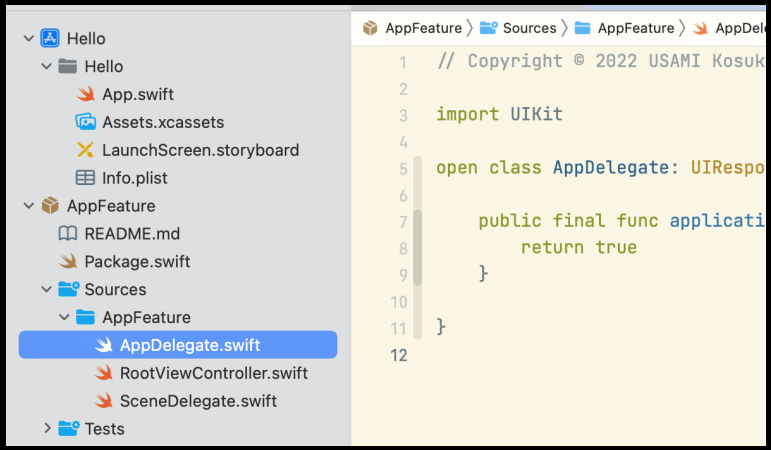
- ローカルのSwiftパッケージをアプリでインポートする



※ このトークでは、こちらの手法を扱う。

アプリのコードをパッケージ化する

- 一部だけでなく、ほとんどのコードをパッケージに入れても良い



Xcodeプロジェクト内のソース

- App.swift

```
import UIKit
import AppFeature

@main
final class AppDelegate: AppFeature.AppDelegate {}

final class SceneDelegate: AppFeature.SceneDelegate {}
```

Swiftパッケージ内のソース

- AppDelegate.swift

```
import UIKit

open class AppDelegate: UIResponder, UIApplicationDelegate {

    public final func application(_ application: UIApplication, ...) -> Bool {
        return true
    }

}
```

パッケージ化のメリット

- Xcodeプロジェクト（xcodeproj）でのソースコード管理が減る
  - xcodeprojは、ファイルの追加や削除などでGitのコンフリクトを招く
  - Swiftパッケージ管理だと、Gitのコンフリクトを起こしにくい
- アプリ内のモジュール分割が容易になる
  - Swiftパッケージのほうが簡単に扱える

パッケージ化で未解決の問題

- ビルドスクリプトはXcodeプロジェクトで管理する必要がある
  - SwiftGenでコード生成
  - SwiftLintでコードチェック
- 実はこの問題は、SwiftPMのプラグイン機能で解決できる

SwiftPMのプラグイン機能とは

SwiftPMのプラグイン機能

- 2022年3月のSwift 5.6で追加された機能
- コマンドプラグイン
  - ビルド以外のタスクを定義できる
- ビルドツールプラグイン
  - ビルド時に行う処理を追加できる

プラグイン機能の活用方法

- 配布されているプラグインを使う
  - 配布されているものは、現時点では多くはない
- プラグインを自分で実装する
  - 独自の処理を行いたい場合はこの方法になる

※ プラグインの実装方法はパンフレット記事を参照。

ビルドツールプラグイン

```
let package = Package(
    targets: [
        .target(
            name: "MyTarget",
            plugins: [
                .plugin(name: "MyPlugin"),
            ]
        ),
        .plugin(
            name: "MyPlugin",
            capability: .buildTool()
        ),
    ]
)
```

ビルドツールプラグインの処理内容

- 以下の2つのタイミングで処理が実行される
  - ビルド前（pre-build）
  - ビルド中（in-build）
- プラグインであらかじめ定義された処理が実行される
  - 処理内容を自分で決めたい場合は、プラグインを自分で実装する

外部ツールを使う

- プラグイン外のツールを実行できる
- Mac内のコマンドを実行できる
- 公開されているコマンドラインツールをダウンロードできる
  - artifact bundle形式で公開されているバイナリが使える

iOSアプリ開発でSwiftPMプラグインを活用する

XcodeとSwiftPMプラグイン

- XcodeでもSwiftPMプラグインは動作する
  - Xcode 13.3以降で動作する
  - Xcode 14でSwiftPM対応が改善されている（ビルドログなど）
- ただし、一部の動作に問題がある（後述）

事例：SwiftGenプラグイン

- SwiftGen公式から、プラグインとartifact bundleが提供されている
- ビルド前（pre-build）にソースコード生成処理が行われる
- 生成先は `` `${DERIVED_SOURCES_DIR}` `` 以下となる
  - `` `swiftgen.yml` `` で定義する
- なお、ビルドツールだけでなくコマンドプラグインも提供されている

SwiftGenプラグインの利用 (1)

- 注意：この方法が正式だが、現時点では問題がある

```
let package = Package(
    dependencies: [
        .package(url: "https://github.com/SwiftGen/SwiftGenPlugin", from: "6.6.2")
    ],
    targets: [
        .target(
            name: "MyTarget",
            plugins: [
                .plugin(name: "SwiftGenPlugin", package: "SwiftGenPlugin")
            ]
        ),
    ]
)
```

Xcodeで発生する問題

- 外部プラグイン利用時、Xcodeが重くなる
  - XcodeのCPU使用率が100%以上になる
  - Xcodeのエディタの動きがもたつく
- 外部プラグインの中でartifact bundleを使っていると発生する
  - 外部ツールをダウンロードする機能
  - SwiftGenプラグインは `swiftgen` コマンドをartifact bundleで使用

Xcodeで発生する問題の回避方法

- Xcodeの問題を回避するには、プラグインを自分で実装する
- artifact bundleの利用自体は問題ない
  - 外部プラグインの中でartifact bundleが使われているとダメ
  - ローカルプラグインの中でartifact bundleを使うのは大丈夫

SwiftGenプラグインの利用 (2)

```
let package = Package(
    targets: [
        .plugin(
            name: "SwiftGenPlugin",
            capability: .buildTool(),
            dependencies: ["swiftgen"]),
        .binaryTarget(
            name: "swiftgen",
            url: "https://github.com/SwiftGen/SwiftGen/releases/...",
            checksum: "..."
        ),
    ]
)
```

事例：SwiftLintプラグイン

- SwiftLint公式から、artifact bundleが提供されている
  - これを利用して、自分でプラグインを実装すればよい

SwiftLintプラグインの実装

```
struct SwiftLintPlugins: BuildToolPlugin {
    func createBuildCommands(context: PluginContext,
                             target: Target) async throws -> [Command] {
        return [buildCommand(
                    displayName: "Linting \(target.name)",
                    executable: try context.tool(named: "swiftlint").path,
                    arguments: [
                        "lint",
                        "--in-process-sourcekit",
                        target.directory.string
                    ],
                    environment: [:])]
    }
}
```

SwiftLintプラグインの利用

```
let package = Package(
    targets: [
        .plugin(
            name: "SwiftLintXcode",
            capability: .buildTool(),
            dependencies: ["SwiftLintBinary"]
        ),
        .binaryTarget(
            name: "SwiftLintBinary",
            url: "https://github.com/realm/SwiftLint/releases/...",
            checksum: "..."
        ),
    ]
)
```

まとめ

- Swift Package Manager（SwiftPM）とは
- iOSアプリ開発でSwiftPMを活用する
- SwiftPMのプラグイン機能とは
- iOSアプリ開発でSwiftPMプラグインを活用する
- サンプル：
  - https://github.com/usami-k/XcodeSwiftPMSample