🧩 用 Swift Package Manager 写一个可复用的 SwiftUI 小组件

> iOS Club 课程讲义
> 主讲主题：Swift Package Manager（SwiftPM）进阶实战
> 目标：掌握如何使用 SwiftPM 创建、管理、发布一个 SwiftUI 小组件

一、SwiftPM 简介

Swift Package Manager 是什么？

SwiftPM（Swift Package Manager）是苹果官方的依赖管理与模块化工具，用于：

- 管理第三方库（类似 CocoaPods / Carthage）
- 构建自己的独立模块或组件
- 让代码可在多个项目间复用

为什么要学 SwiftPM？

- Xcode 内置支持，无需额外工具
- 配合 GitHub 可直接分发模块
- 在团队协作中，便于模块化开发
- 对 SwiftUI 组件开发特别友好

二、课程目标

我们将从零开始创建一个 Swift Package，名为 **FancyButton**：

> 一个带渐变背景、圆角和点击动画的 SwiftUI 按钮组件。

最终我们能在任何项目中：

```
import FancyButton

FancyButton("Press Me") {
    print("Tapped!")
}
```

三、创建 Swift Package

打开终端并输入以下命令：

```
mkdir FancyButton
cd FancyButton
swift package init --type library
```

目录结构如下：

```
FancyButton/
├── Package.swift
├── Sources/
│   └── FancyButton/
│       └── FancyButton.swift
└── Tests/
    └── FancyButtonTests/
```

四、配置 Package.swift

打开 `Package.swift` 文件，将内容修改为：

```
// swift-tools-version: 5.9
import PackageDescription

let package = Package(
    name: "FancyButton",
    platforms: [
        .iOS(.v15) // 指定最低 iOS 版本
    ],
    products: [
        .library(
            name: "FancyButton",
            targets: ["FancyButton"]
        ),
    ],
    dependencies: [
        // 如需第三方包可在此声明
    ],
    targets: [
        .target(
            name: "FancyButton",
            dependencies: []),
        .testTarget(
            name: "FancyButtonTests",
            dependencies: ["FancyButton"]),
    ]
)
```

> 💡 提示：
> `products` 指明包对外暴露的模块；
> `targets` 定义模块内部结构；
> `platforms` 决定支持哪些系统。

五、编写 SwiftUI 小组件

编辑 `Sources/FancyButton/FancyButton.swift`：

```swift
import SwiftUI

public struct FancyButton: View {
    private var title: String
    private var action: () -> Void

    public init(_ title: String, action: @escaping () -> Void) {
        self.title = title
        self.action = action
    }

    public var body: some View {
        Button(action: action) {
            Text(title)
                .font(.headline)
                .padding()
                .frame(maxWidth: .infinity)
                .background(
                    LinearGradient(
                        colors: [.purple, .blue],
                        startPoint: .leading,
                        endPoint: .trailing
                    )
                )
                .foregroundColor(.white)
                .cornerRadius(12)
                .shadow(radius: 5)
        }
        .buttonStyle(.plain)
    }
}

#Preview {
    FancyButton("Tap Me") {
        print("Button tapped!")
    }
    .padding()
}
```

✅ **要点说明：**

- `public` 关键字让外部项目能访问组件。
- `#Preview` 可直接在 Xcode 预览。
- 完全独立模块，可单独编译。

六、在其他项目中使用

方式一：本地引用

1. 打开目标 Xcode 项目
2. 选择菜单 **File → Add Packages...**
3. 点击 “Add Local Package…”
4. 选择刚创建的 `FancyButton` 文件夹
5. 在代码中使用：

```
import FancyButton

struct ContentView: View {
    var body: some View {
        FancyButton("Press Here") {
            print("Hello from FancyButton!")
        }
        .padding()
    }
}
```

方式二：发布到 GitHub

1. 在终端初始化 Git 仓库：

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/你的用户名/FancyButton.git
git push -u origin main
```

2. 在其他项目中添加：

```
https://github.com/你的用户名/FancyButton.git
```

3. Xcode 自动识别 Swift Package 并拉取代码。

七、延伸：SwiftPM 与 GitHub 的结合

SwiftPM 与 Git 深度绑定：

- 任何公开的 Git 仓库都可被引用为依赖。
- SwiftPM 会自动解析 `Package.swift` 与版本标签（Tag）。
- 可使用 tag 版本化管理，如：

```
git tag 1.0.0
git push origin 1.0.0
```

八、团队协作与模块化应用

在 iOS Club 或多人协作中，可以用 SwiftPM 拆分模块：

```
iOSClubApp/
├── Packages/
│   ├── UIComponents/
│   ├── Networking/
│   └── Utils/
└── iOSClubApp/
```

好处：

- 各模块独立测试与编译
- 代码清晰、低耦合
- 可复用在多个项目
- 新人可单独负责一个模块开发

> 🧠 课堂建议：
> 让学生把项目里用到的常用函数拆出成 `Utils` 包。

九、SwiftPM 与 Swift Playgrounds

Swift Playgrounds 4 支持导入 Swift Package：

- 可在 iPad 上直接导入组件练习
- 让没有 Mac 的同学也能尝试 Swift 模块化开发
- 可以布置作业：
  「写一个 SwiftUI 小组件 Package，并导入 Playground 预览」

十、版本管理技巧

场景	推荐方式
锁定稳定版本	使用 `Up to Next Major`
试验性功能	指定分支（如 `main`）
依赖冲突	检查 `Package.resolved` 或清除缓存：`File → Packages → Reset Package Caches`

十二、课堂总结

通过本节课，你应该能够：

✅ 理解 SwiftPM 的基本结构
✅ 能从零创建一个 SwiftUI 小组件
✅ 学会在项目中导入或发布 Swift Package
✅ 理解 SwiftPM 在团队项目中的价值
