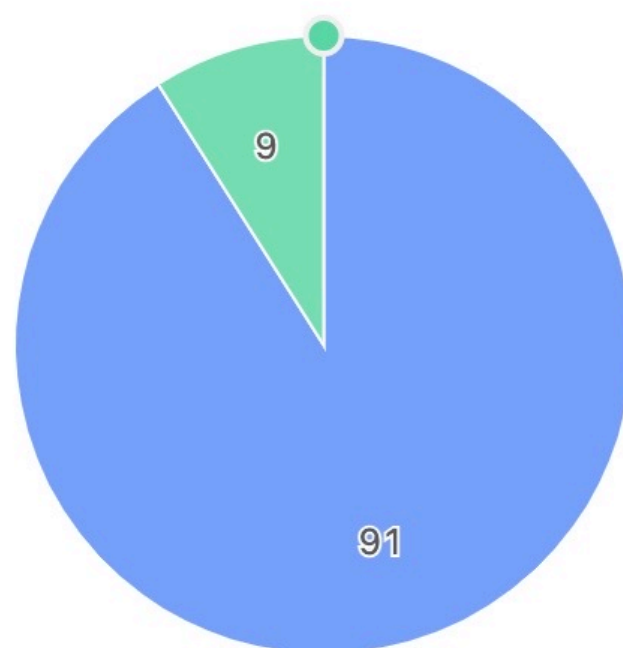


基于 *Swift* 编写严肃脚本工具

微信 *mango* [方秋枋]

国外情况

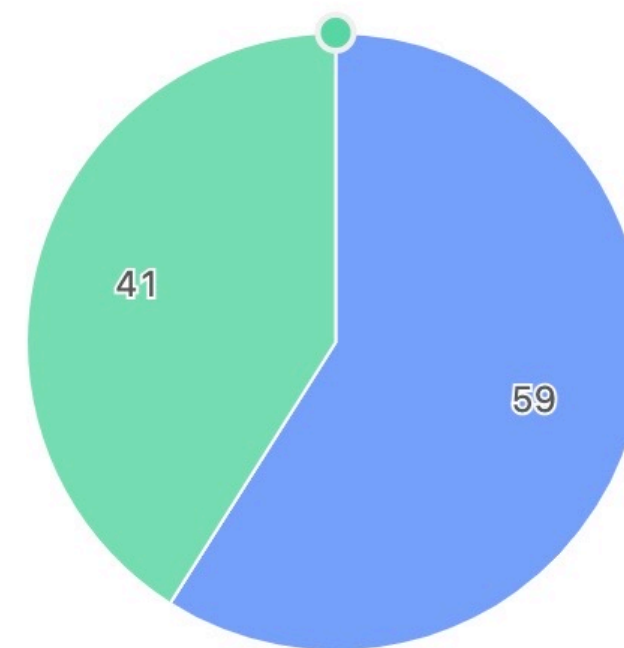
使用 Swift 语言的 App 占比



● 使用 Swift ● 未使用 Swift

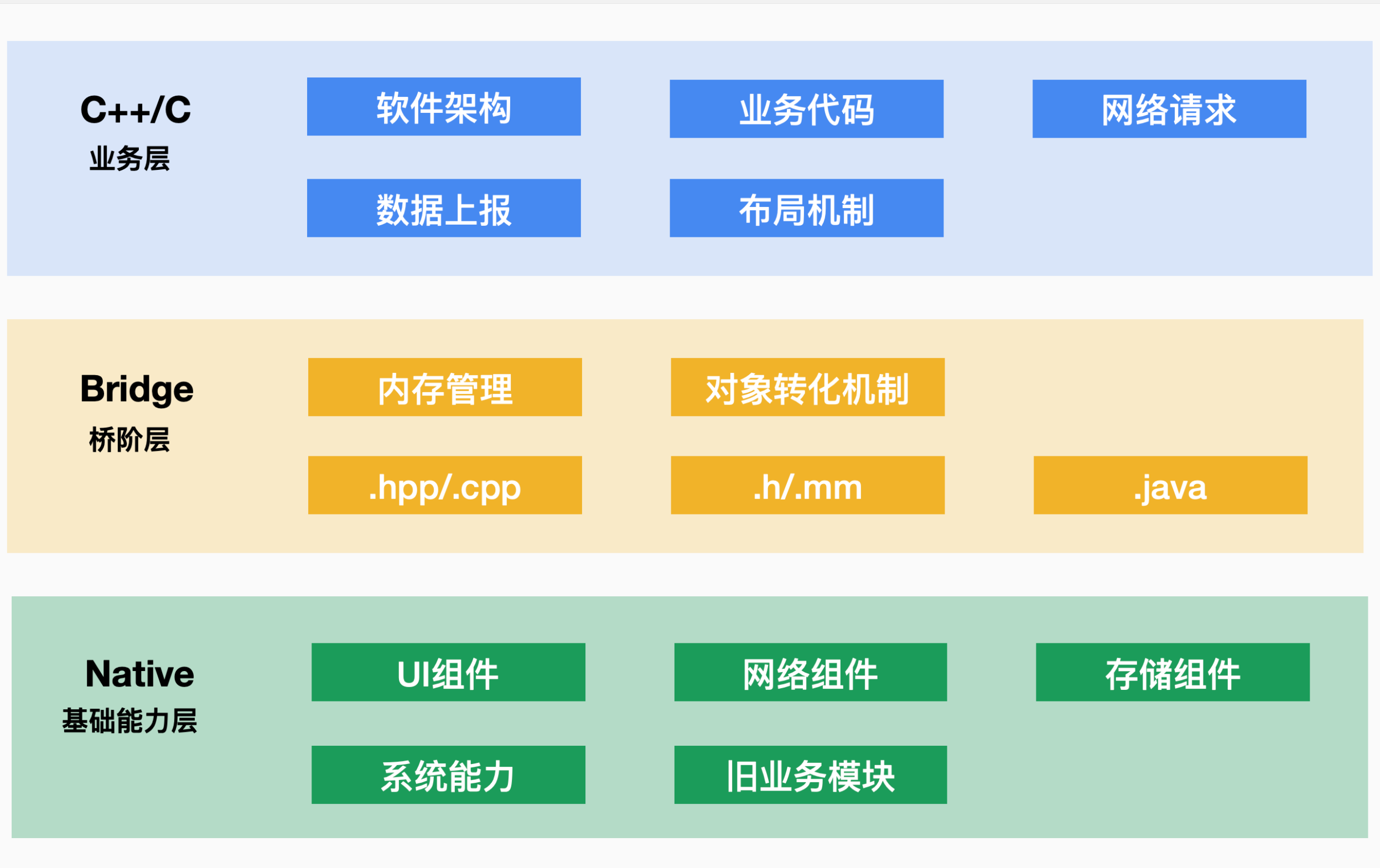
国内情况

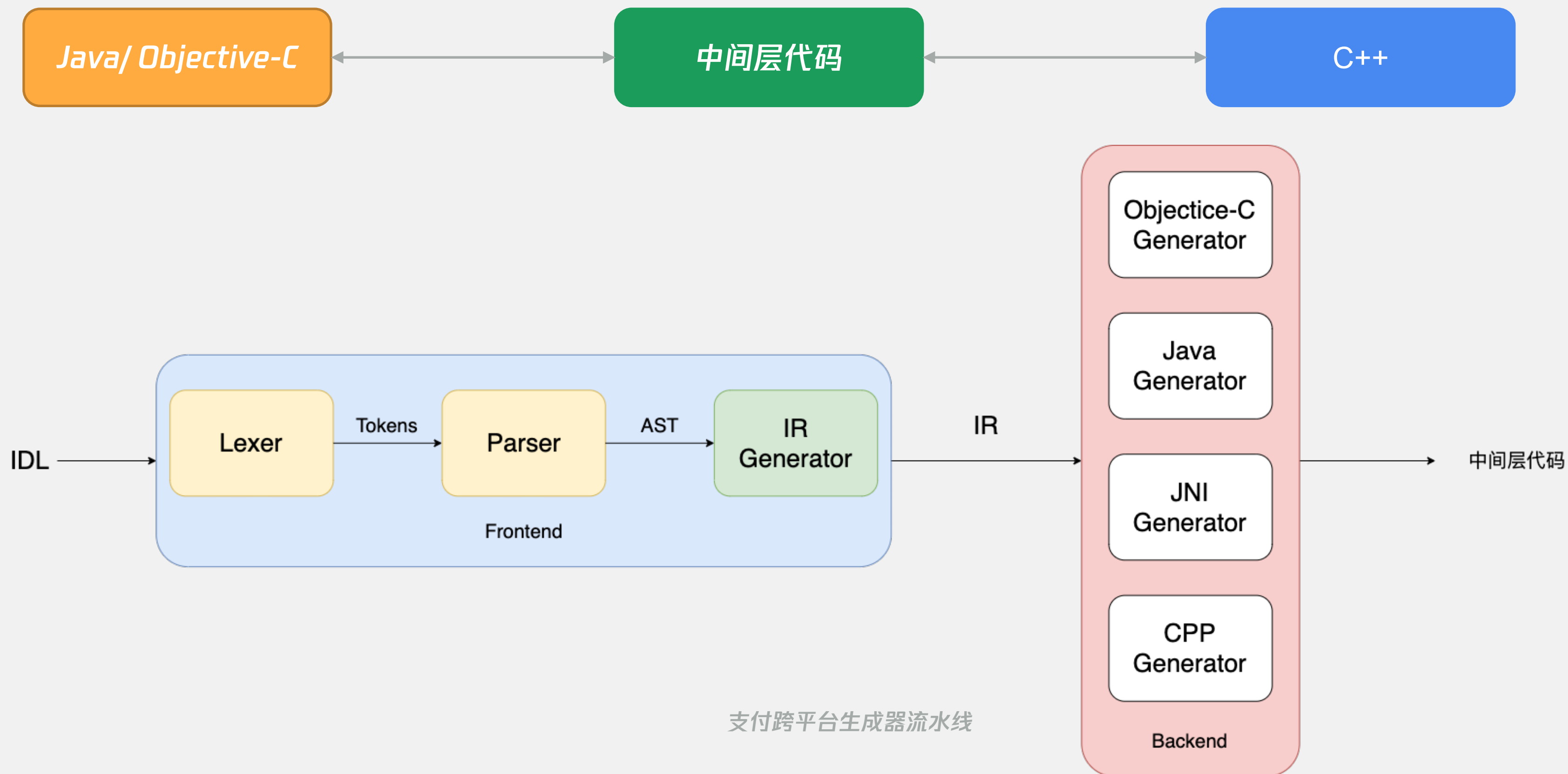
使用 Swift 语言的 App 占比



● 使用 Swift ● 未使用 Swift

1. 微信 *Apple Watch* 端
2. 订阅号助手
3. 部分机器学习逻辑
4. 脚本工具





大纲

01 创建工程

02 编写代码

03 测试

04 发布

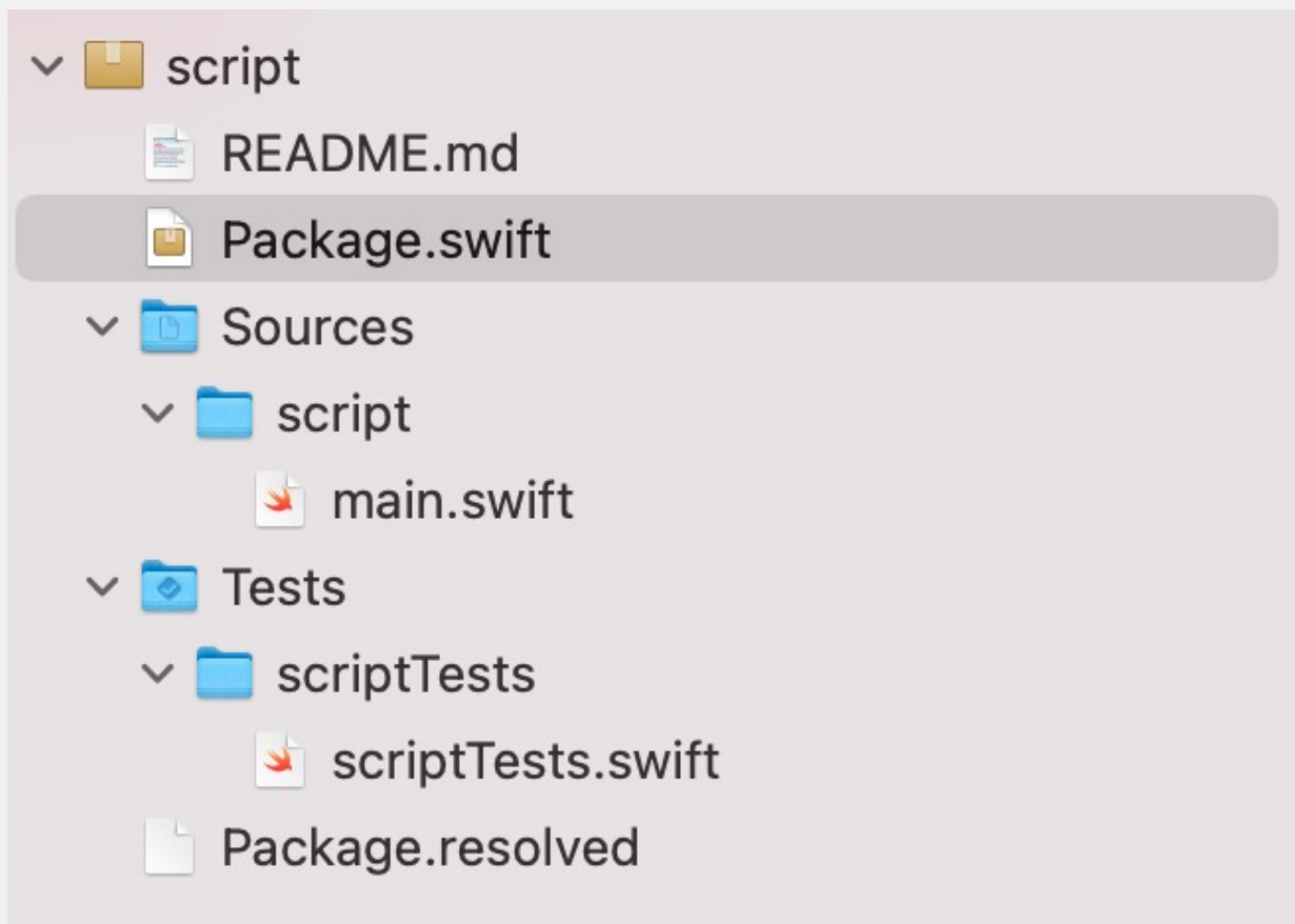
05 进阶操作

创建工程

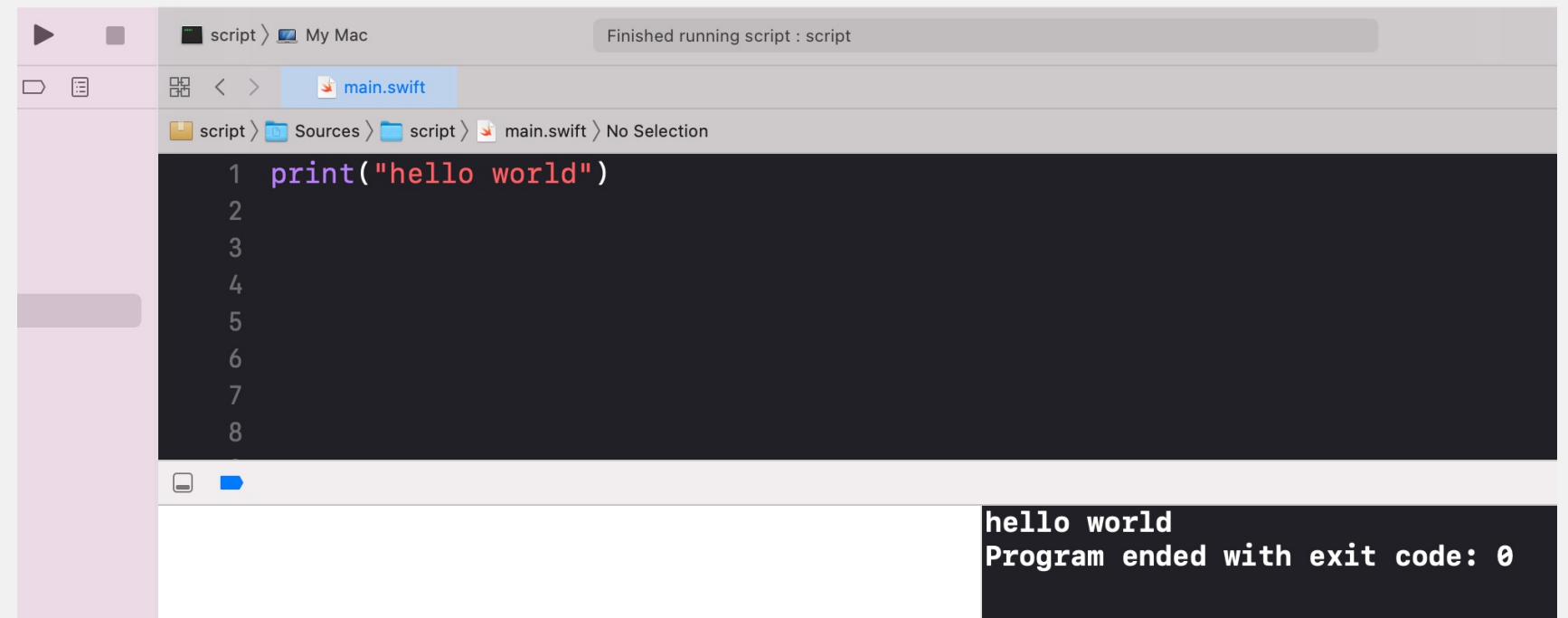
```
$ mkdir script
```

```
$ cd script
```

```
$ swift package init --type executable
```

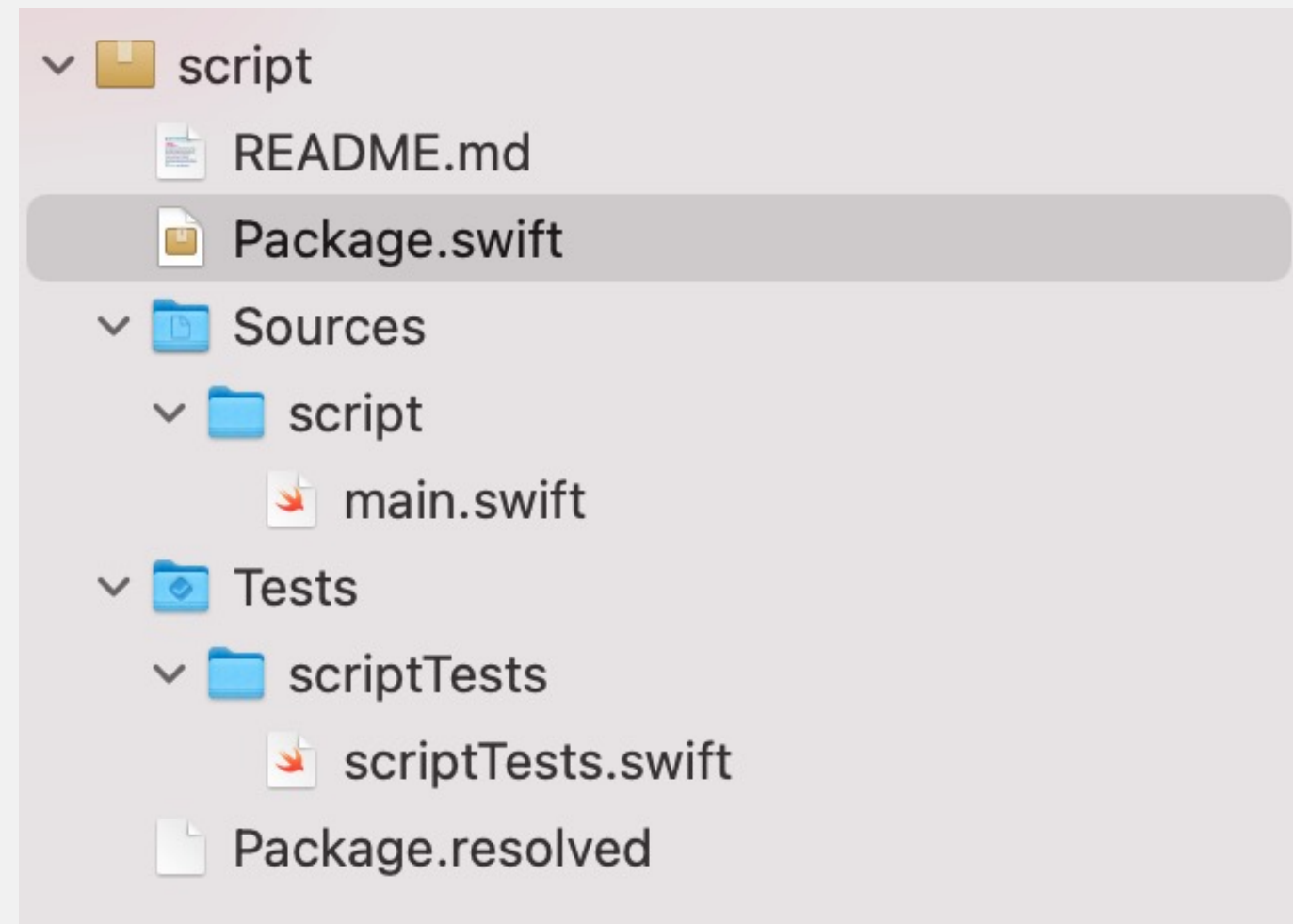


目录结构

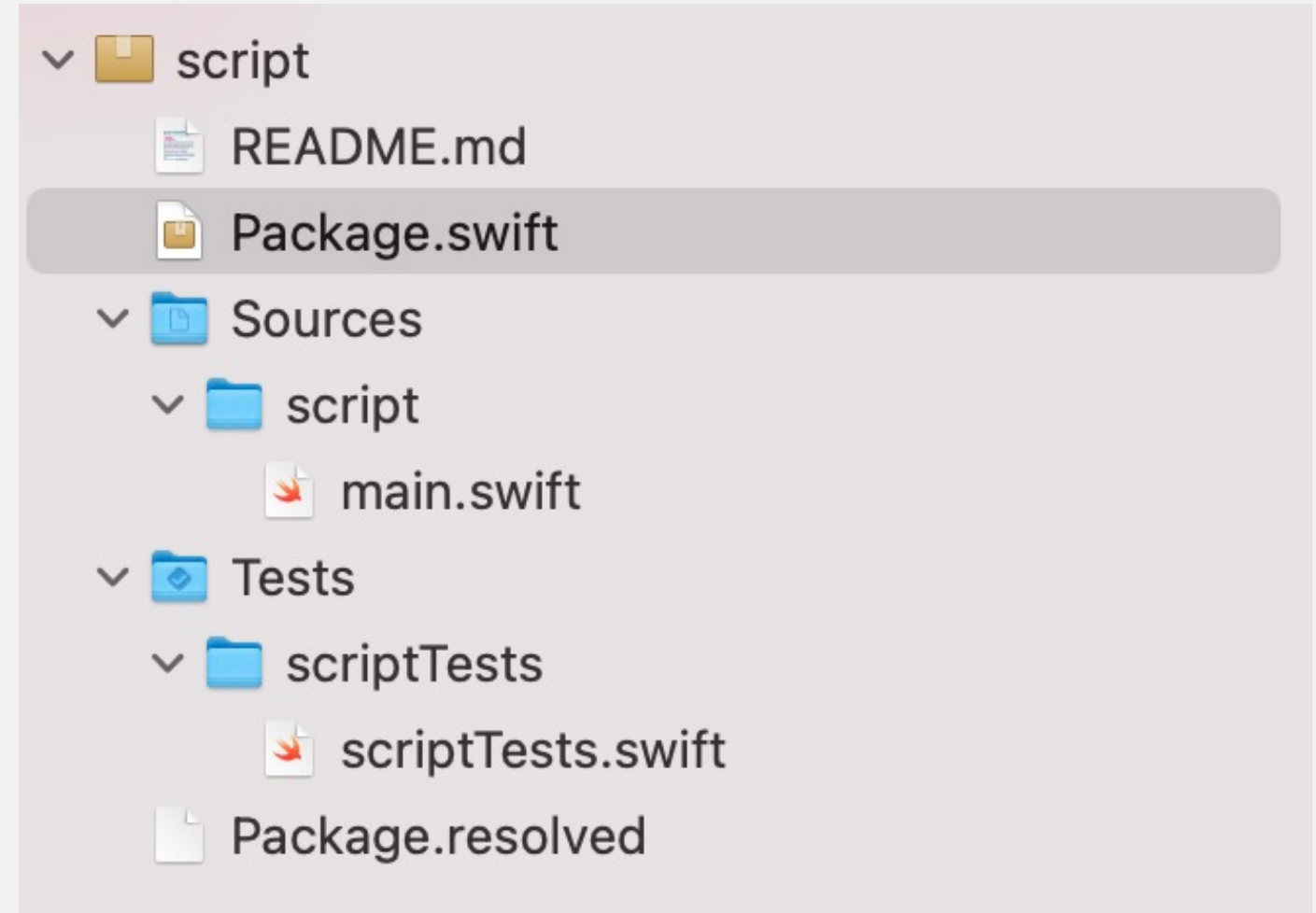


main.swift

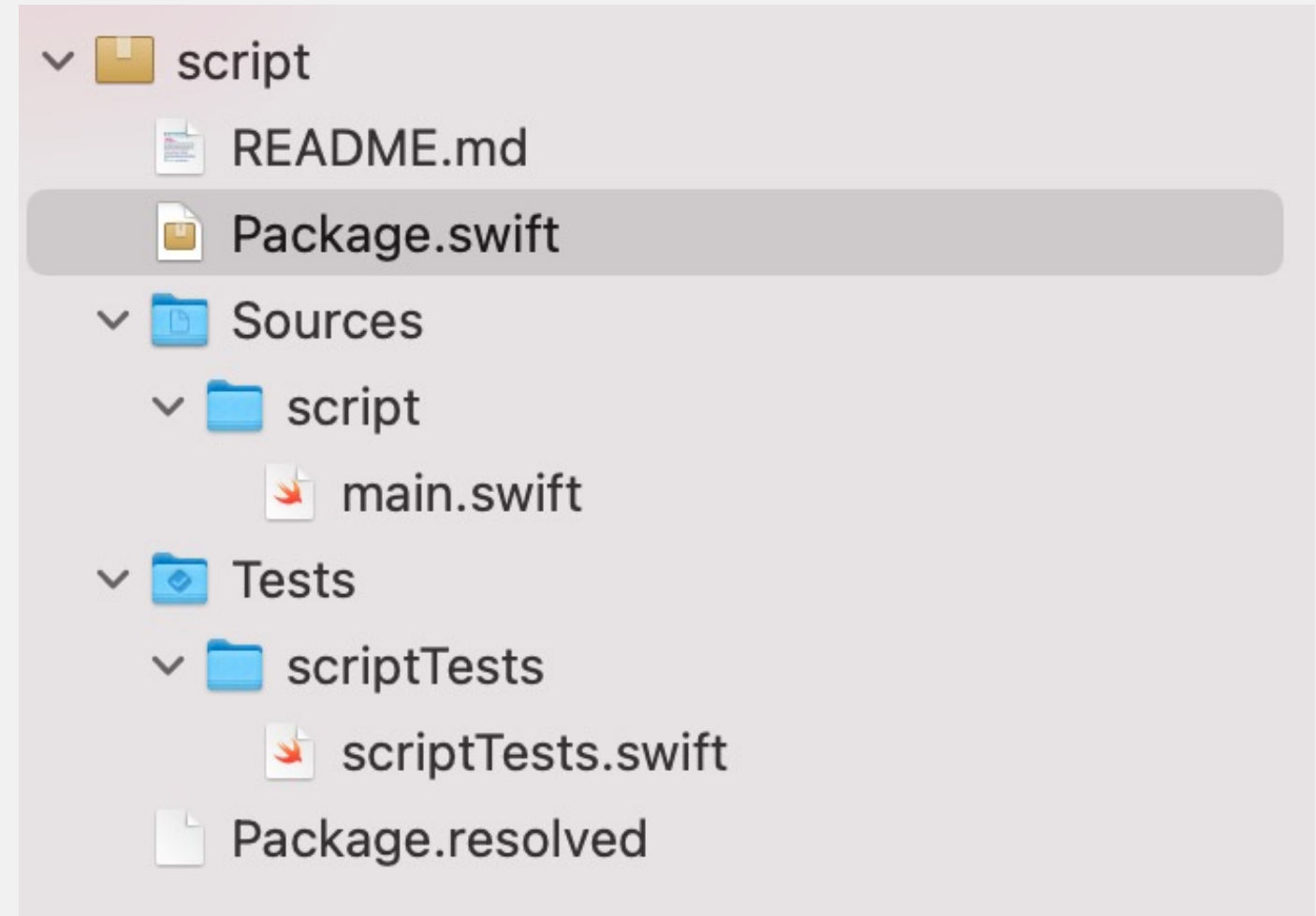
```
let package = Package(  
    name: "script",  
    dependencies: [],  
    targets: [  
        .target(  
            name: "script",  
            dependencies: []),  
        .testTarget(  
            name: "scriptTests",  
            dependencies: ["script"]),  
    ]  
)
```



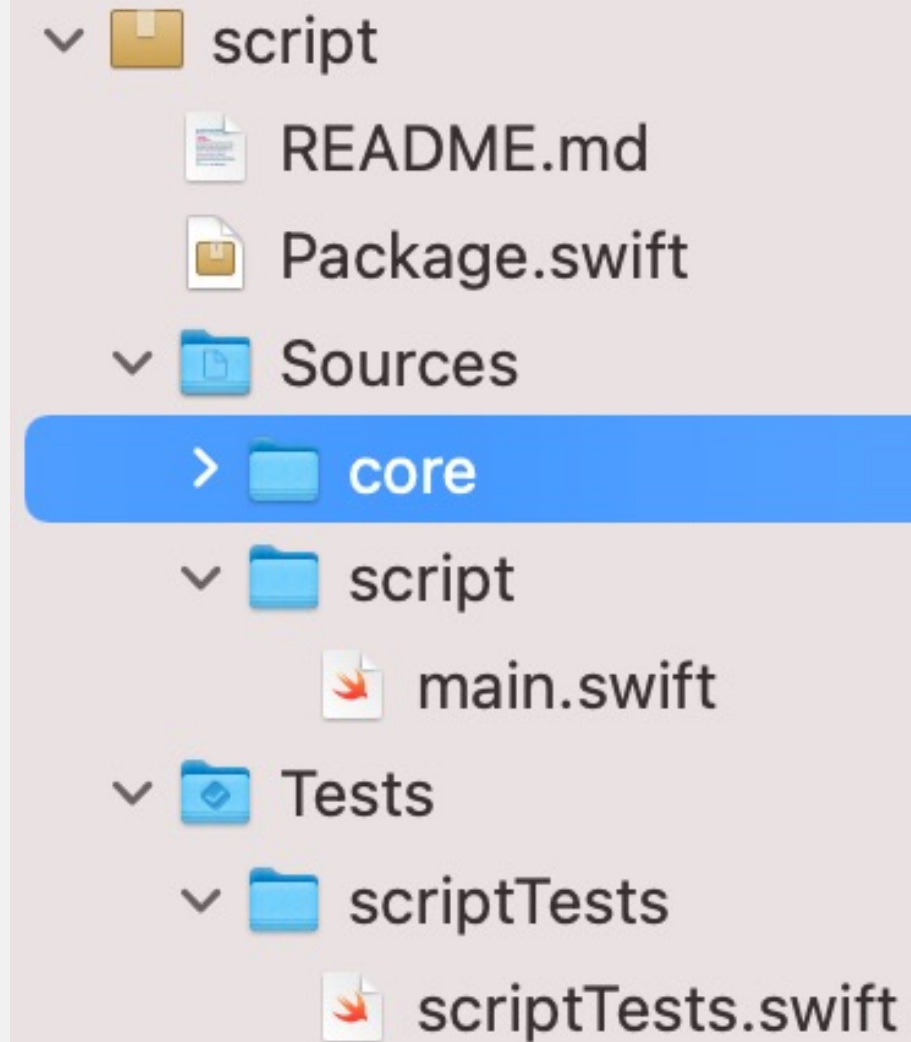

```
let package = Package(  
    name: "script",  
    dependencies: [],  
    targets: [  
        .target(  
            name: "script",  
            dependencies: [],  
            path: "./Sources/script"),  
        .testTarget(  
            name: "scriptTests",  
            dependencies: ["script"]),  
    ]  
)
```



```
let package = Package(  
    name: "script",  
    dependencies: [],  
    targets: [  
        .executableTarget(  
            name: "script",  
            dependencies: []),  
        .testTarget(  
            name: "scriptTests",  
            dependencies: ["script"]),  
    ]  
)
```



```
let package = Package(  
    name: "script",  
    dependencies: [],  
    targets: [  
        .executableTarget(  
            name: "script",  
            dependencies: ["core"]),  
        .target(  
            name: "core",  
            dependencies: []),  
        .testTarget(  
            name: "scriptTests",  
            dependencies: ["script"]),  
    ]  
)
```



1. 基于 *Swift Package manager* 搭建基础框架
2. 显式声明 *executable target*
3. 拆分核心逻辑模块和命令行模块

大纲

01 创建工程

02 编写代码

03 测试

04 发布

05 进阶操作

```
import Darwin
```

```
let arguments: [String] = Array(CommandLine.arguments.dropFirst())
```

```
guard let numberString = arguments.first else {  
    print("no argument")  
    exit(1)  
}
```

```
guard let number = Int(numberString) else {  
    print("not number")  
    exit(1)  
}
```

```
print(Int.random(in: Int.min...number))  
exit(0)
```

1. 没有 `--help` 方法， 没有使用说明
2. 一个复杂的命令行工具， 参数会有很多， 还有可选参数， *flag*, *Option* 等类型的参数。
3. 除此之外还有参数校验等需要。

```
let package = Package(  
  name: "script",  
  dependencies: [],  
  targets: [  
    .executableTarget(  
      name: "script",  
      dependencies: ["core"]),  
    .target(  
      name: "core",  
      dependencies: []),  
    .testTarget(  
      name: "scriptTests",  
      dependencies: ["script"]),  
  ]  
)
```


编写代码 / 引入 *Argument Parser*

```
let package = Package(
  name: "script",
  dependencies: [
    .package(url: "https://github.com/apple/swift-argument-parser", from: "0.4.0")
  ],
  targets: [
    .executableTarget(
      name: "script",
      dependencies: ["core",
        .product(name: "ArgumentParser", package: "swift-argument-parser")]),
    .target(
      name: "core",
      dependencies: []),
    .testTarget(
      name: "scriptTests",
      dependencies: ["script"]),
  ]
)
```

```
import ArgumentParser

struct Random: ParsableCommand {
    @Argument(help: "unsigned number")
    var highValue: UInt

    func run() {
        print(UInt.random(in: 0...highValue))
    }
}

Random.main()
```

参数校验:

Error: Missing expected argument '<high-value>'

Usage: random <high-value>

See 'random --help' for more information.

使用说明

USAGE: random <high-value>

ARGUMENTS:

<high-value>	unsigned number
--------------	-----------------

OPTIONS:

-h, --help	Show help information
------------	-----------------------

1. 通过一个生成随机数工具感受 *Swift* 编写命令行工具
2. 命令行工具的参数解析非常繁琐，我们引入 *Argument Parser* 库

大纲

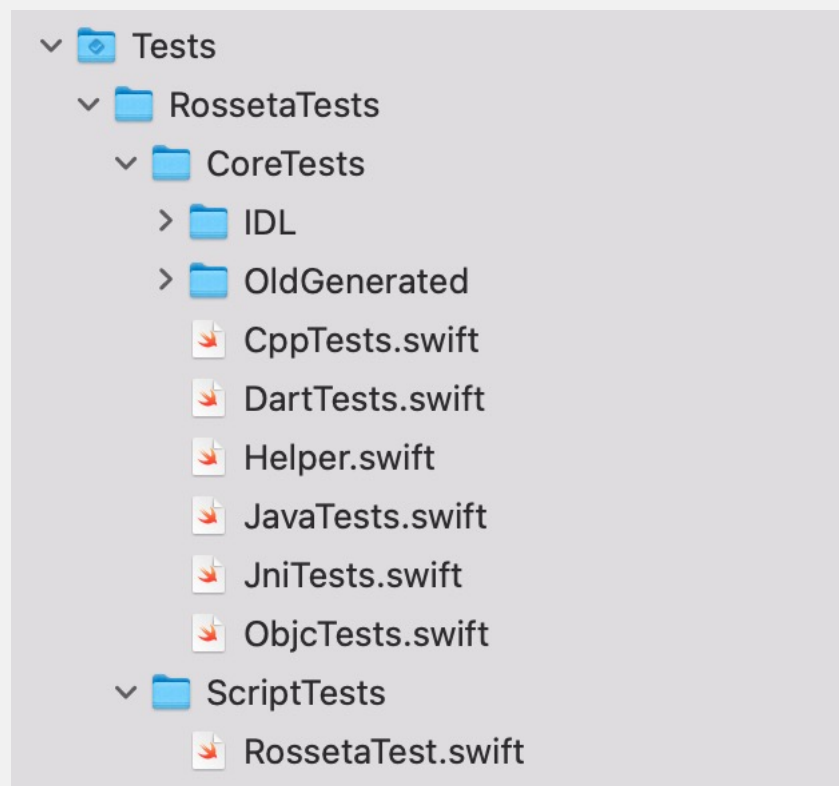
01 创建工程

02 编写代码

03 测试

04 发布

05 进阶操作



核心逻辑 和 命令行API 测试解耦

*CoreTests*负责核心逻辑测试, *ScriptTests* 负责命令行 API 测试

```
func testRosseta() throws {  
    //1. 创建进程  
    let process = Process()  
    //2. 设置可执行文件位置, 这里我们设置的是 Xcode 编出来的可执行文件路径  
    process.executableURL = Bundle.allBundles.first { $0.bundlePath.hasSuffix(".xctest") }!  
        .bundleURL.deletingLastPathComponent().appendingPathComponent("Rosseta")  
    //3. 设置参数  
    process.arguments = [  
        "--objc-path", "~/Desktop/RossetaGenerated",  
        "--cpp-path", "~/Desktop/RossetaGenerated",  
    ]  
    //4. 设置Pipe, 用于获取打印内容  
    let pipe = Pipe()  
    process.standardOutput = pipe  
    //5. 启动进程  
    try process.run()  
    process.waitUntilExit()  
    //6. 读取命令行工具输出内容, 判断是否符合预期  
    let data = pipe.fileHandleForReading.readDataToEndOfFile()  
    let output = String(data: data, encoding: .utf8)  
    XCTAssertEqual(output, "success")  
}
```

大纲

01 创建工程

02 编写代码

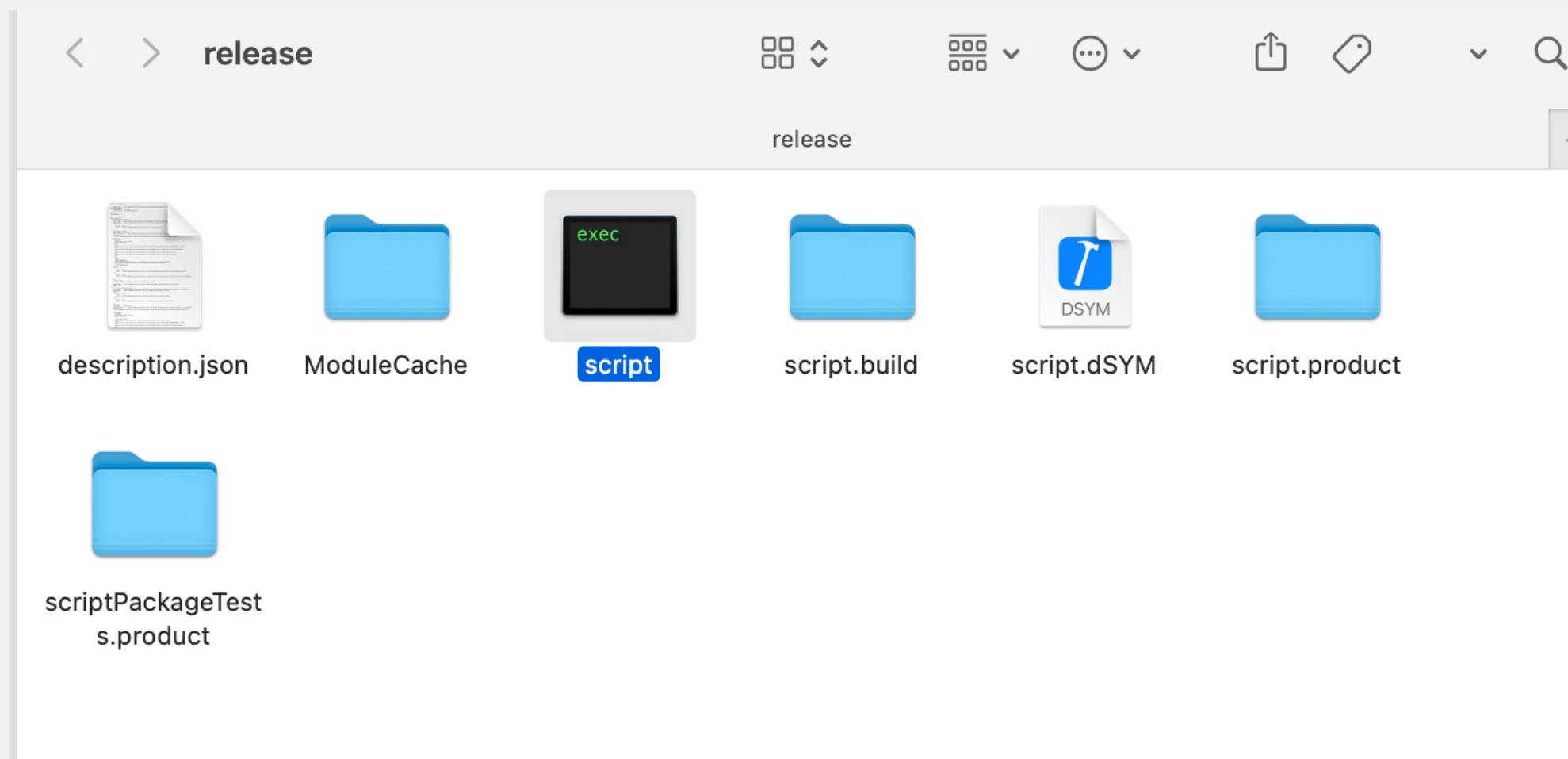
03 测试

04 发布

05 进阶操作

swift build --configuration release

生成的可执行文件位于 *.build/release/* 目录下



大纲

- 01 创建工程
- 02 编写代码
- 03 测试
- 04 发布
- 05 进阶操作

```
NSURLSession.shared.dataTask(with: url, completionHandler: { __, response, __ in  
    print(response)  
}).resume()
```

```
NSURLSession.shared.dataTask(with: url, completionHandler: { _, response, _ in  
    print(response)  
    //退出  
    exit(EXIT_SUCCESS)  
}).resume()
```

```
//启动RunLoop  
RunLoop.main.run()
```

Demo: 与 Appkit / SwiftUI 交互

```
NSApplication.shared.setActivationPolicy(.accessory)
```

```
func selectFile() -> URL? {  
    let dialog = NSOpenPanel()  
    dialog.allowedFileTypes = ["jpg", "png"]  
    guard dialog.runModal() == .OK else { return nil }  
    return dialog.url  
}
```

```
print(selectFile()?.absoluteString ?? "")
```

进阶操作 / 与 SwiftUI 交互

```
import Foundation
import SwiftUI

struct App: SwiftUI.App {
    @State var fileUrl: URL?
    @State var showFileChooser = false

    var body: some Scene {
        WindowGroup {
            HStack {
                Button("select File")
                {
                    let panel = NSOpenPanel()
                    panel.allowsMultipleSelection = false
                    panel.canChooseDirectories = false
                    if panel.runModal() == .OK {
                        self.fileUrl = panel.url
                    }
                }
                if let url = fileUrl, let nsImage = NSImage(contentsOf: url) {
                    Image(nsImage: nsImage)
                }
            }
            .frame(maxWidth: .infinity, maxHeight: .infinity)
        }
        .windowStyle(HiddenTitleBarWindowStyle())
    }
}
App.main()
```

总结

01 创建工程

02 编写代码

03 测试

04 发布

05 进阶操作