Task Engine · Day 1

Phase 0 · Swift Core Engine

———

Day 1 的核心目标

Day 1 并不是在复习或学习 Swift 语法，
而是在用 Swift **正确地建模一个真实世界中的业务概念（Task）**。

这是 Apple 原生开发真正的起点：
先把“世界是什么样的”定义清楚，而不是先写 UI。

———

今日完成的工程成果

1) Core Engine 运行环境已建立

- 使用 Xcode 创建 Command Line Tool (Swift)
- 无 UI、无 SwiftUI、无 UIKit
- 无第三方依赖
- 通过 main.swift 成功运行并输出业务结果

这一步确认了一件事：
业务逻辑可以脱离 App、脱离界面独立存在。

2) 领域模型 (Domain Model) 已完成并跑通
已实现并运行以下模型：

- Task
- TaskStatus
- TaskTag
- TaskError

这些模型的共同特征：

- 使用值类型 (struct / enum)
- 不依赖任何 UI 框架
- 可直接复用到未来的 iOS / SwiftUI App
- 表达的是业务事实，而不是界面状态

———

Task 模型中的关键设计认知

- note、dueAt 使用 Optional

→ 表达“业务上允许不存在”，而不是用空字符串或 0 伪装

- tags 使用 Set

→ 明确业务规则：标签不允许重复

- TaskStatus 使用 enum

→ 将任务生命周期限制在合法状态集合中

- overdue 判断属于模型自身能力

→ 这是业务逻辑，不是 UI 逻辑

———

Day 1 真正获得的能力（非语法层面）

1) 值类型建模思维

- 一个 Task 是一个完整的值
- 状态变化 ≠ 修改对象
- 状态变化 = 生成一个新的状态值

这是 SwiftUI、并发模型、响应式编程的底层思想。

2) Optional 的语义意识
Optional 不再是“Swift 强迫写的语法”，
而是用于表达现实世界中**允许不存在的状态**。
避免使用空字符串、0、魔法值来冒充“不存在”。

3) enum 用来表达业务规则
enum 的作用是封装规则、限制非法状态、
让系统状态具备可预测性和可推理性。

———

关于 public 的工程理解（Day 1 的重要认知）

使用 public 并不是因为当前项目“必须”，
而是因为在**以“可复用模块”的视角设计 Core Engine**。

这是在提前思考：

- 哪些类型是模块对外的契约
- 哪些类型未来会被 UI / App 使用

这是 Apple Framework 一贯的设计方式。

———

关于 Xcode 中 Group 与 Folder 的区别

Group：

- 逻辑结构
- 用于表达代码架构层次（Models / Services 等）
- 不一定对应磁盘上的文件夹

Folder：

- 物理结构
- 真正存在于磁盘中
- 更偏向大型工程或 Swift Package 管理

当前阶段遵循的原则是：
先把逻辑结构设计清楚，再决定是否需要物理结构。

———

Day 1 阶段性结论

Day 1 的价值不在于写了多少代码，
而在于：

第一次用 Swift，克制、准确、可扩展地描述了一个现实世界概念。

这一步，奠定了后续 SwiftUI、系统框架、并发模型学习的地基。

———

Day 1 自检清单与问题体系的真正含义（6 + 3）

这些自检项和问题并不是用来“检查有没有写对代码”，
而是用来**校准自己是否在用“正确的工程视角”思考问题**。

———

一、Day 1 自检清单在检查什么（6 项）

1) 是否全部使用值类型（struct / enum）
检查的不是语法选择，而是思维是否仍停留在“引用对象惯性”。
值类型意味着：

- 数据是完整的
- 状态变化是可预测的
- 不依赖隐式共享引用

这是 SwiftUI、并发模型安全性的基础。

2) Optional 是否只用于“业务上允许不存在”的字段
Optional 的存在是为了表达现实的不确定性，而不是为了通过编译。
需要区分：

- 可以不存在（note、dueAt）
- 绝对不能不存在（title）

这是对业务规则的诚实建模。

3) enum 是否在限制非法状态
enum 的核心价值在于：

- 把业务状态限制在一个封闭集合中
- 在编译期直接禁止非法值

这是一种“让系统帮你守规则”的设计方式。

4) 集合类型是否在表达业务规则
选择 Set 而不是 Array，本质是在让数据结构本身承担规则责任。

- Set = 不允许重复
- Array = 允许重复

这是用类型系统替代人为约定，减少隐性 bug。

5) 模型是否可以独立运行并解释结果
能够运行、打印、预测输出结果，
说明对模型的理解是“确定的”，而不是“模糊的”。

6) 是否能说清楚每个字段和设计选择的业务含义
如果某个字段存在，但说不清它在现实中代表什么，
那说明模型已经开始偏离真实世界。

———

二、Day 1 的三个问题真正想训练的能力（3 项）

1) 为什么使用 struct，而不是 class
这道问题不是在问“Swift 推荐什么”，
而是在问：

- 这个概念在业务上是“值”，还是“身份对象”？

Task 在业务上是一个完整的值，
复制它、比较它、传递它都是合理的行为。

2) Optional 在这里表达了什么不确定性
这是一道业务建模问题，而不是语法问题。
需要思考的是：

- 这个字段为空时，现实是否仍然成立？

Optional 是对现实世界的诚实表达，而不是妥协。

3) 如果未来需求变复杂，会拆哪一层
这道问题在训练“架构意识”，而不是让当下给出完美答案。
它的意义在于：

- 是否已经意识到模型、规则、创建逻辑的边界
- 是否在为系统的“生长空间”预留可能性

只要开始意识到“哪些东西不该继续堆在一起”，
就已经进入工程思维阶段。