Task Engine · Day 2

主题：Validation · Factory · Error as Business

———

Day 2 在系统层面解决的是什么问题

Day 2 的核心不是“学会 throws 语法”，
而是让系统具备一种能力：

主动拒绝非法世界，而不是被动承受错误后果。

从这一天开始，Task 不再是“谁都能随便造的对象”。

———

关于 Factory（受控创建）的认知

从 Day 2 起，Task 的创建入口被收敛到一个受控方法中：

- 调用者不再直接 init
- 所有创建行为必须经过系统校验
- 校验规则集中在一个明确的位置

这意味着一件非常重要的事情：

一个 Task 一旦存在，就一定是“业务上合法的”。

Factory 的意义不是代码形式，而是系统边界的建立。

———

关于 throws / do-catch 的真实含义

throws 并不等同于“异常情况”，
而是用于表达一种事实：

创建 Task 这件事，在现实世界中是可能失败的。

失败不是程序错误，也不是 crash 条件，
而是业务的一部分，必须被显式建模、显式处理。

do-catch 的作用不是兜底，
而是给调用者一个清晰的选择权：
如何面对失败的现实。

———

关于“失败是业务”的工程视角

在 Day 2 中，以下情况被明确建模为失败结果：

- 空标题
- 标题过长
- 非法时间关系（dueAt < createdAt）

这些失败不会导致程序崩溃溃、不会产生非法状态，
而是被系统**冷静地拒绝**。

这是一种成熟系统的行为模式。

———

关于系统稳定性的一个重要标志

当错误发生时，程序能够：

- 不 crash
- 不 silent fail
- 不污染系统状态
- 平静结束并返回 exit code 0

这说明系统已经具备“可预期失败”的能力。

———

Day 2 阶段性结论

如果 Day 1 是在定义世界的结构，
那么 Day 2 是在定义世界的边界。

从这一天开始：

- Task 不会在非法条件下出生
- 系统不再信任调用者
- 规则由系统统一掌控
- 错误被当作现实的一部分对待

这是后续状态流转、更新规则、并发安全的前提。

———