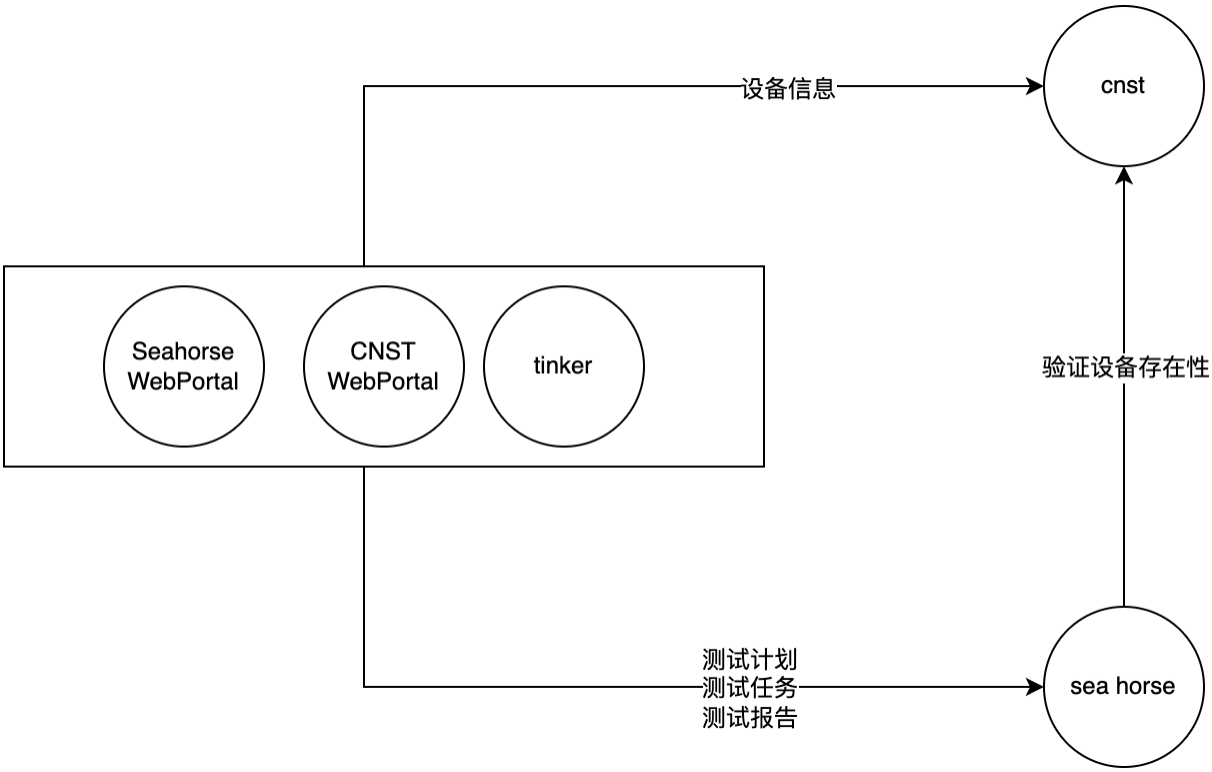
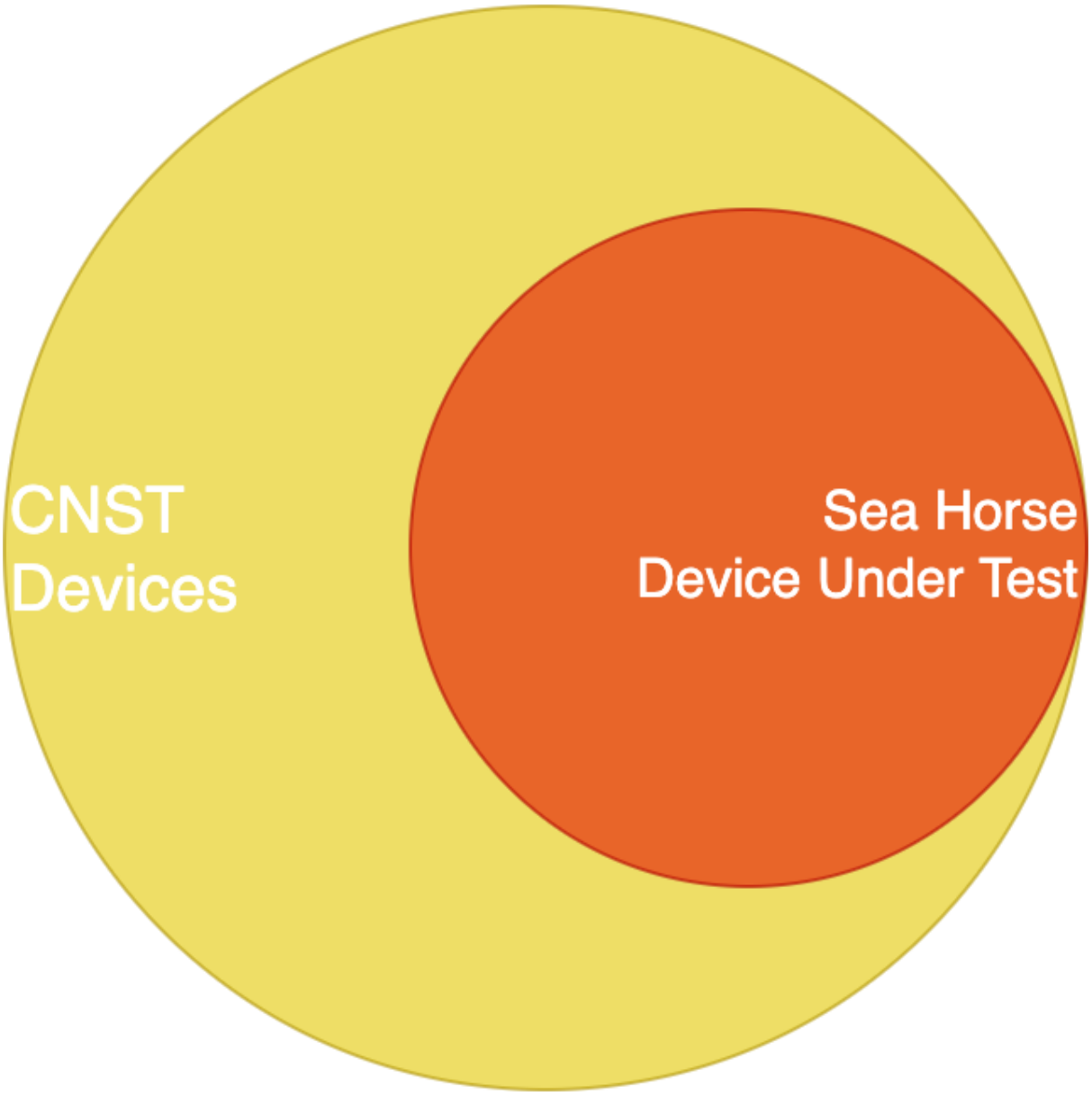
# cnst 2.0 与 sea horse 和tinker的关系

## 系统之间的关系




CNST管理设备

Seahorse管理测试（但将CNST作为Device Registry，Seahorse不存储设备信息，只存储设备Id作为DUT的Id）




# 前端与后端服务cornerstone+后端服务seahorse之间的数据交互原则

1. 我们不希望不同系统之间，有多个存储测试用例的地方，或者有多个存储Device信息的地方。我们需要single source of truth。
2. tinker中产生的数据发送给cnst以创建/修改设备信息，
3. tinker中产生的测试报告会发送给seahorse
4. seahorse 存储DUT时，需2向CNST确认设备是否存在。
5. CNST本身不存储测试报告信息，哪个前端需要与测试信息互动，就直接向SeaHorse发送请求。

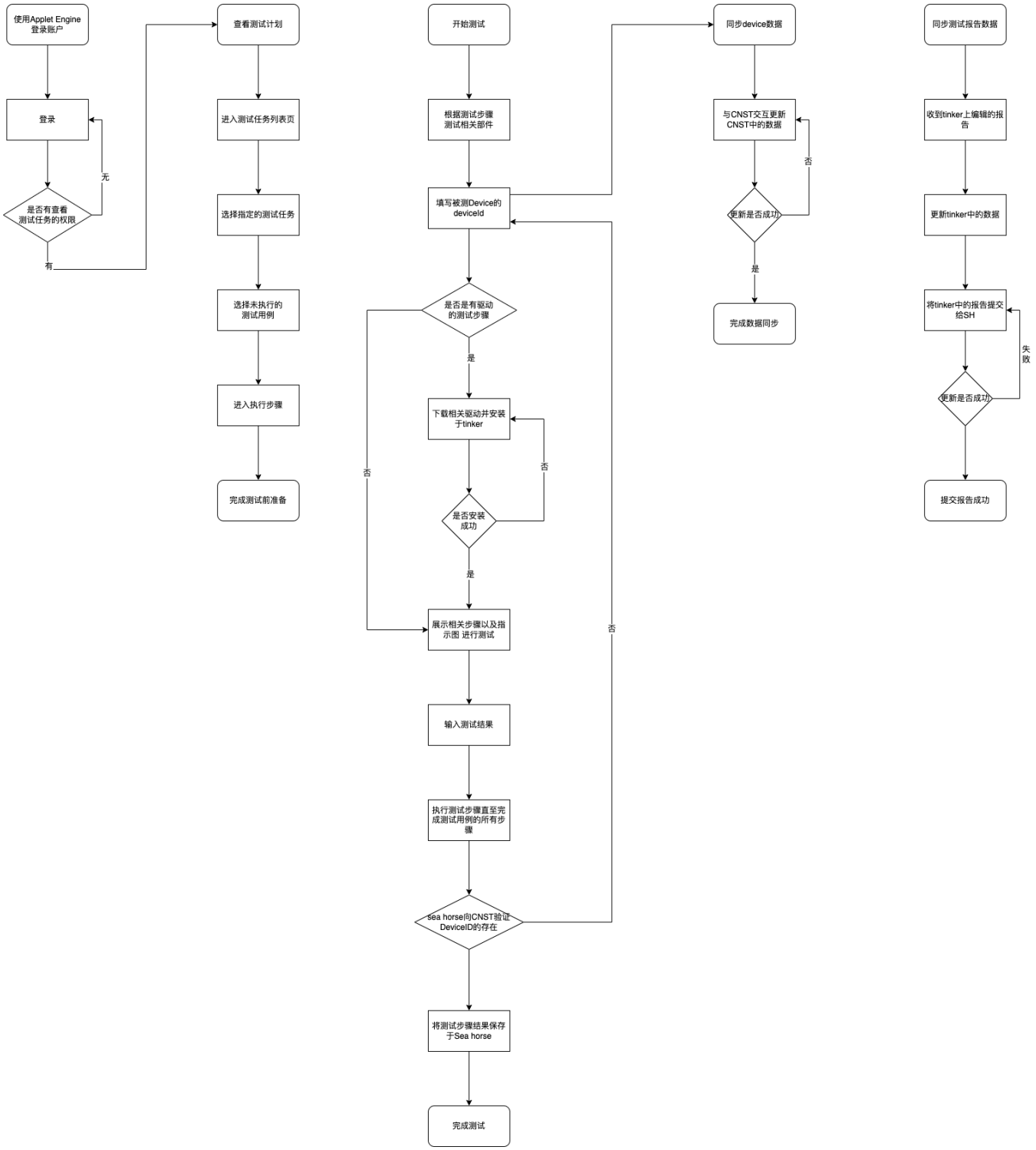
# device生命周期前端架构。

**tinker中device生命周期前端架构**：https://confluence.syriusrobotics.cn/x/UgXLDQ

## 数据存储/修改 任务调度前端架构

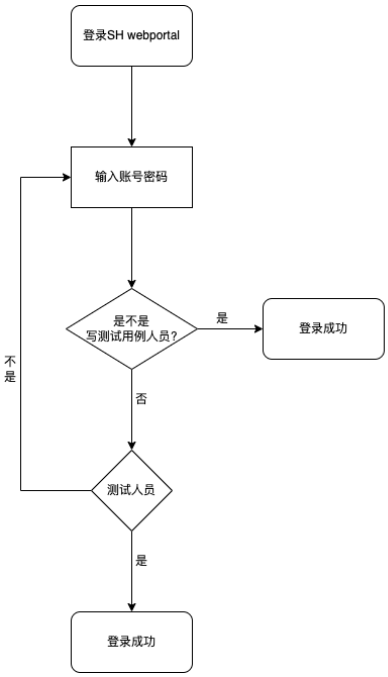
**tinker中的 发送/接受 数据任务的调度架构：https://confluence.syriusrobotics.cn/x/olPmDQ**

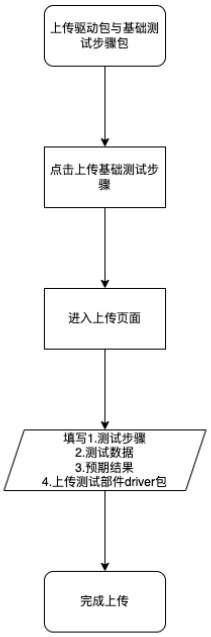
# tinker产品流程图：

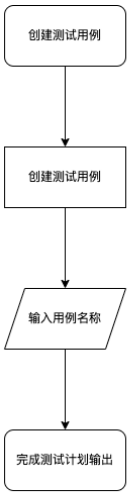


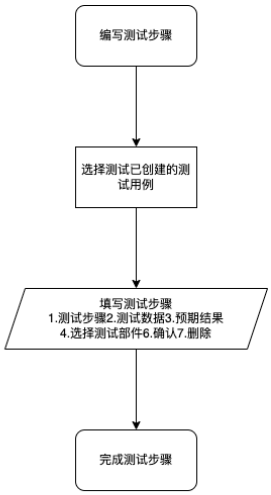

1. 所有向Sea horse 提交测试报告的请求都要再验证一遍deivce是否在CNST中注册
    a. 这个验证由Sea horse向CNST发起请求 而不是Tinker。

# sea horse webportal产品流程



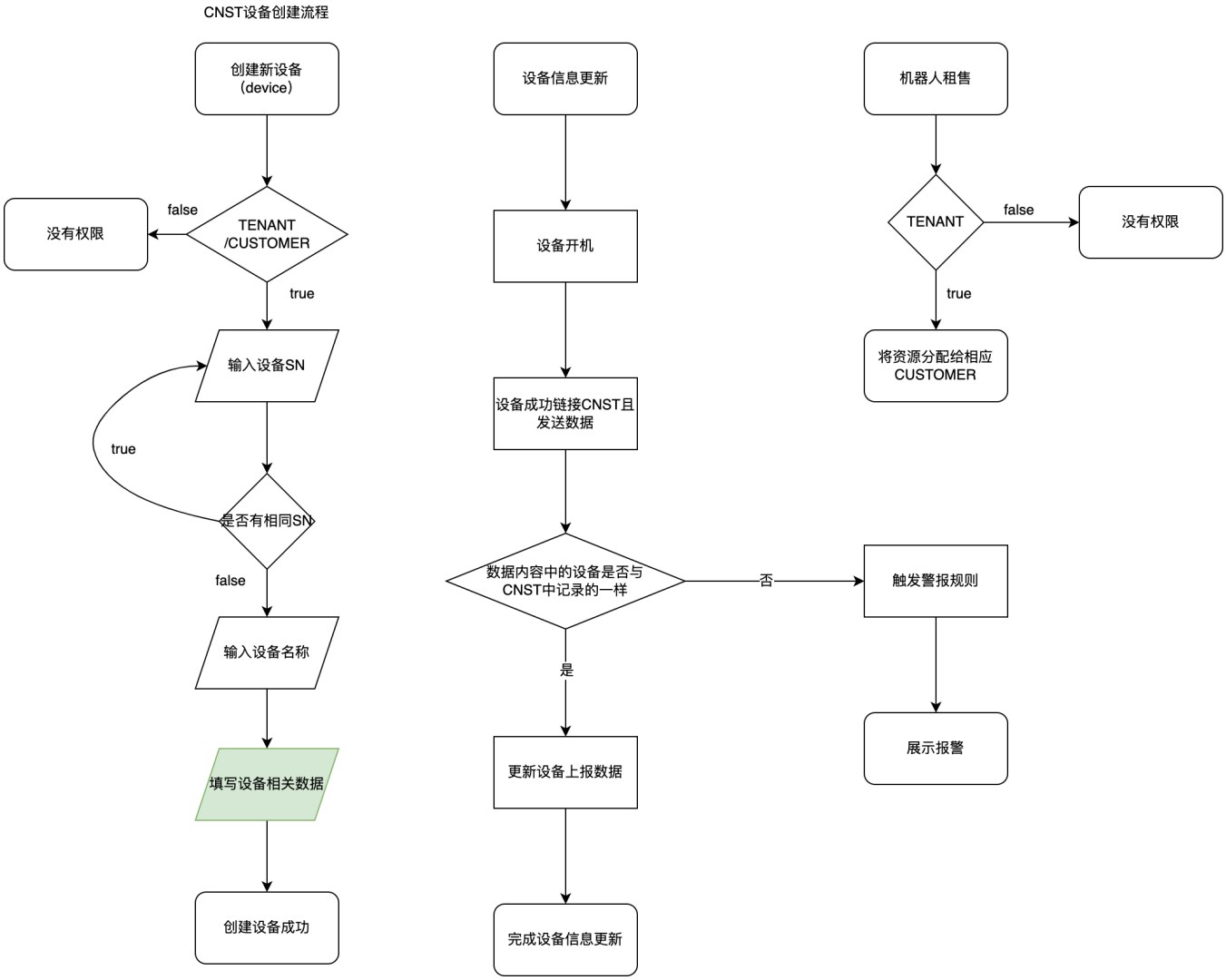




1. 写测试用例的人和driver的人是两批人
    a. 写用例的人是质量控制人员。
    b. 维护driver的人要知道怎么让部件在appletengine上运行。
2. 底层 driver设计要只表达当前硬件的状态，该状态与硬件为孪生关系。
    a. 断开连接时：表达断联状态
    b. 已连接时表达当前状态。
    c. driver的状态变化也要同时导致硬件产生相同的变化，断开连接时 要重置所有之前对变化的要求。
3. 不以PID VID决定一个部件及其driver
    a. 要用componentModelId来维护部件唯一标识
    b. 原因是有一些部件没有PIDVID 例如：项检查轮子的颜色是否和要求的一样
    c. 在上传driver的时候创建componentModelId的关联

# CNST产品说明




JSAPI 是暴露给小程序的接口这一层要十分稳定

如果不稳定会导致小程序开发者重新开发小程序

在这之下与hibrid通信可以通过OTA来升级