RTU 平台 BSP 初始化服务方案设计

（V1.0）版 20260528

1， 软硬件的需求

硬件需求：具备运行嵌入式 Linux 操作系统的硬件平台，提供满足系统运行的计算与存储资源（包括 CPU、内存及存储介质），并支持 BSP-Framework 初始化所需的基础硬件接口及设备资源。

软件需求：基于支持 systemd 的嵌入式 Linux 操作系统，以实现 BSP-Framework 初始化服务的启动管理、依赖控制、状态监测及异常恢复，同时需提供必要的系统 root 权限。

2， 软件功能服务

为实现 RTU 平台基础能力的统一管理与模块化扩展，系统采用基于 systemd 的服务化架构设计，将各功能模块拆分为独立系统服务（Service）运行。各服务依据职责边界进行划分，通过标准化的启动、停止、依赖管理及异常恢复机制，实现系统功能的解耦与稳定运行。

软件功能服务主要围绕设备基础能力、硬件资源管理、远程运维及本地配置等场景展开，覆盖网络管理、时间同步、设备定位、系统状态采集、远程运维、应用安装及 Web 配置等核心功能。各服务可根据设备硬件能力及业务需求按需启用或扩展。

系统启动过程中，由 systemd 统一负责服务生命周期管理，包括服务启动顺序控制、依赖关系管理、运行状态监测及异常自动恢复，以保障设备在异常场景下具备较好的自恢复能力和持续运行稳定性。同时，平台提供本地扩展机制，支持根据具体硬件型号、项目需求及业务场景，通过本地启动服务或定制化服务实现功能扩展与适配，降低 BSP 初始化与设备适配复杂度，提高系统可维护性与可移植性。

系统服务架构设计如下图所示：

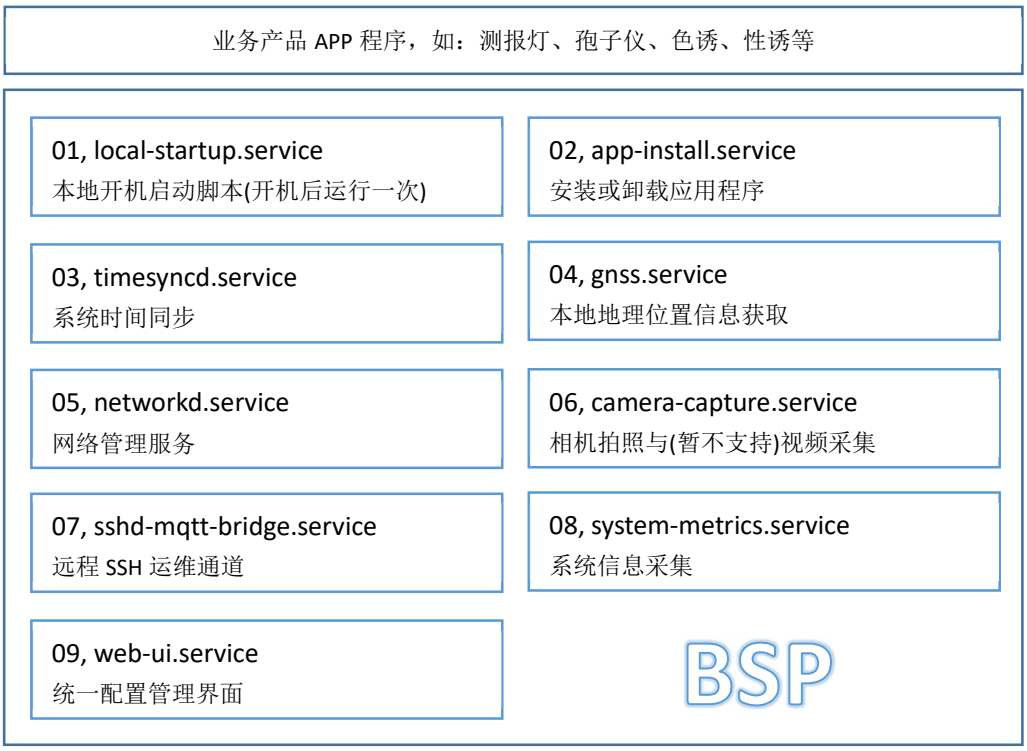

图 1 系统服务架构图

如上图所示，当前系统共包含 9 个基于 systemd 管理的核心服务模块，各服务分别承担不同的

系统功能职责，包括应用管理、时间同步、I 联网管理、相机拍照、地理位置信息获取、远程运维、系统监控以及 Web 配置等功能。

各服务之间相互解耦，均以独立进程形式运行，并通过标准化接口对外提供能力支撑。系统统一采用 JSON-RPC 2.0 作为跨进程通信协议，上层应用或外部客户端可通过该协议以统一的请求/响应方式调用各服务接口，实现服务能力的标准化暴露与调用。

该架构具备良好的模块化设计特点：

√ 通过 JSON-RPC 2.0 统一接口规范，屏蔽底层实现差异

√ 支持跨语言、跨进程调用，增强系统扩展性与兼容性

√ 便于后续功能扩展与维护升级

√ 最大程度的减轻、减少对工业设备厂商的依赖，完全自主可控

√ 提供统一可视化的设备操作管理页面，便于售后维护自主运维

系统服务功能设计如下表所示：

编号	名称	功能描述
01	local-startup.service	系统启动后执行本地初始化任务，用于完成硬件初始化及必要的业务启动配置
02	app-install.service	负责应用程序的安装与卸载管理，提供软件生命周期管理能力
03	timesyncd.service	负责系统时间同步功能，通过网络时间源对系统时钟进行校准
04	gnss.service	提供 GNSS 定位能力，用于获取设备的地理位置信息
05	networkd.service	负责网络连接管理，包括网络配置、链路维护及状态管理
06	camera-capture.service	提供摄像头采集服务，支持图像与视频数据获取，现阶段只需支持图片的静态采集，视频流暂不需要
07	sshd-mqtt-bridge.service	提供 SSH 与 MQTT 的通信桥接能力，用于远程类 SSH 命令行运维与消息转发
08	system-metrics.service	负责系统运行状态采集，包括 CPU、内存及系统资源监控等
09	web-ui.service	提供统一的 Web 可视化管理界面，用于系统配置、状态查看、故障定位、一键自检等操作

表 1 系统服务功能表

3， 工作开发计划

开发计划阶段流程如下图所示：

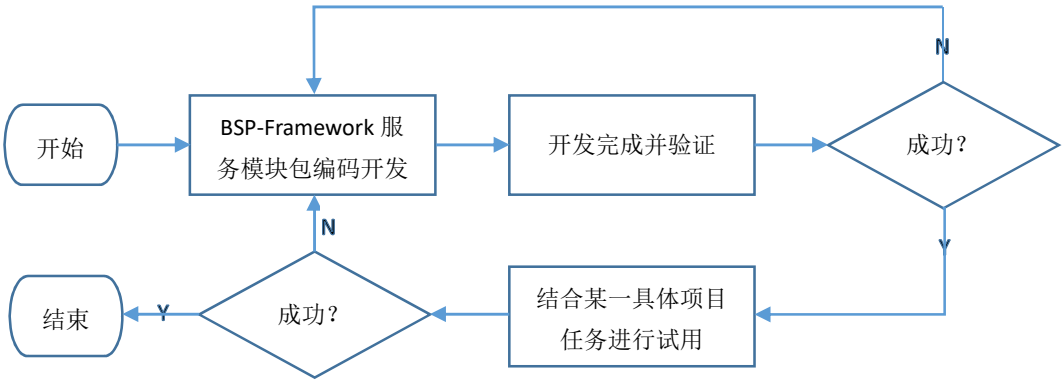

图 2 开发计划阶段流程图

本阶段主要完成板级支持（BSP-Framework）相关开发工作，对底层硬件资源进行初始化与管理，并基于上文设计的 9 个核心系统服务模块（如启动任务、应用管理、时间同步、网络管理和系统信息采集等）构建完整的系统能力，为上层应用提供稳定、高效的硬件抽象接口与运行支撑。

具体的开发计划如下表：

编号	名称	缓急权重	工时	完成指标	备注
01	local-startup.service	☆☆☆☆☆	3	代码编写和自测完成并上传 Git	已完成
02	app-install.service	☆☆☆	10	代码编写和自测完成并上传 Git	未开始
03	timesyncd.service	☆☆☆☆☆	3	代码编写和自测完成并上传 Git	未开始
04	gnss.service	☆☆☆☆☆	3	代码编写和自测完成并上传 Git	未开始
05	networkd.service	☆☆☆☆☆	4	代码编写和自测完成并上传 Git	未开始
06	camera-capture.service	☆☆☆☆☆	4	代码编写和自测完成并上传 Git	未开始
07	sshd-mqtt-bridge.service	☆☆☆☆	3	代码编写和自测完成并上传 Git	未开始
08	system-metrics.service	☆☆	5	代码编写和自测完成并上传 Git	未开始
09	web-ui.service	☆☆☆	10	代码编写和自测完成并上传 Git	未开始
总模块数：9 总工时：45 人/天		已完成模块数：1 待完成模块数：8 剩余工时：42 人天			

表 2 BSP-Framework 开发计划表

4， 生产烧录流程

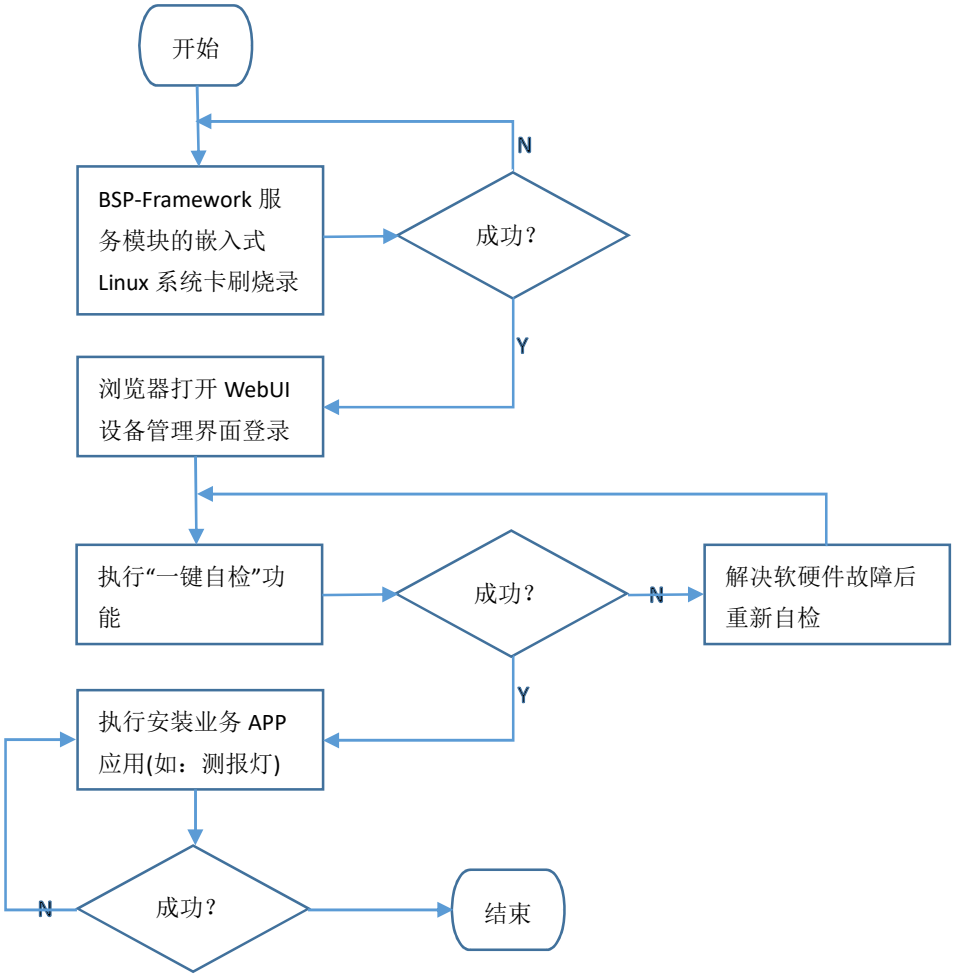

图 3 固件生产烧录流程图

由上图可知，RTU 固件生产烧录流程整体可划分为两个阶段：

阶段 1：基础系统烧录

该阶段主要完成 BSP-Framework 与嵌入式 Linux 系统镜像的烧录，包括底层驱动、基础运行环境及核心系统服务模块的部署。烧录完成后，可通过“一键自检”机制对硬件状态、系统运行环境及关键服务进行自动检测与验证，确保设备具备稳定运行条件，并形成统一、可靠的基础软件运行平台。

阶段二：业务应用部署

在基础系统正常运行的前提下，通过统一的 WebUI 管理界面完成业务 APP 程序的安装、卸载、更新及版本管理，实现业务功能的灵活部署与动态维护。该阶段将底层系统与上层业务解耦，在保证基础系统稳定性的同时，提高业务应用的可扩展性、可维护性以及后期升级效率。

整体来看，本固件烧录流程采用采用“基础平台预置 + 业务应用动态部署”的分层架构设计，有效降低了固件整体耦合度，提升了设备批量生产、版本维护及后续功能扩展的灵活性与可靠性。

5， 售后维护流程

一般售后维护人员：

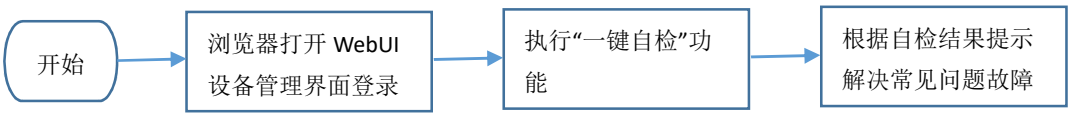

图 4 一般售后维护流程

通过统一 WebUI，售后维护人员可直观可视化的查看 BSP framework 核心模块的服务与任务状态，并独立访问每个模块的运行日志。界面支持实时刷新和状态汇总，可快速定位软硬件异常，提升系统可维护性和运行透明度。

高级售后维护人员：

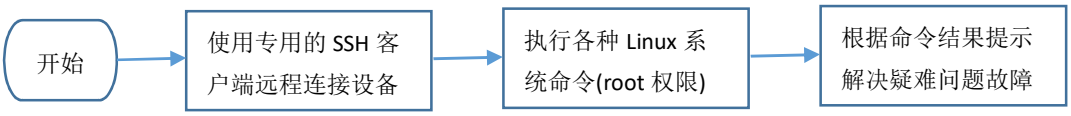

图 5 高级售后维护流程

针对复杂故障场景，高级售后维护人员可通过专用客户端工具经公网远程连接设备，在设备在线状态下执行各类 Linux 命令，实现类似 SSH 的远程运维能力。维护人员可像操作本地终端一样对设备进行实时诊断、系统状态检查、日志分析及配置调整，从而对系统异常、驱动故障、网络通信异常及其他深层次问题进行快速定位与排查，显著提升远程维护效率与问题处理能力。

同时，WebUI 亦提供 WebSSH 工具，主要用于局域网环境下的设备运维访问，维护人员可在浏览器中直接进入设备终端，实现命令行操作与实时监控能力，无需额外客户端软件，支持快速调试与轻量化运维。

高级售后维护人员需熟悉 Linux 命令和 Shell 脚本，能够快速操作系统、分析日志，并对设备功能和业务流程有深入了解，能够高效定位和解决现场问题。