d2ypp2 项目源码分析、业务功能报告与优化方案

生成时间：2026-06-02
源码路径：`C:\project\sh\d2ypp2`
分析范围：当前工作区源码快照，包括当前未提交的本地改动、已生成的前端静态产物和近期部署脚本调整。
目标读者：项目负责人、后端/前端/运维工程师、需要接手无人直播设备部署与维护的技术人员。

---

1. 总体结论

`d2ypp2` 已经不是单一的"抖音直播录制工具"。它是在原有多平台直播源抓取与录制代码基础上，逐步扩展出来的 Linux 无人直播一体化平台。当前项目同时承担以下职责：

1. 抖音、TikTok、YouTube 等直播源抓取、转推、录制和推流。
2. FastAPI 控制平面，提供服务状态、配置编辑、进程启停、Android 控制、WireGuard、ShellCrash、Neko 等 API。
3. Next.js Web 控制台，面向普通操作者提供直播、浏览器、安卓、网络、服务、配置等可视化入口。
4. Linux 一键部署栈，负责用户、权限、systemd、Docker、Caddy、mDNS、Neko、ShellCrash、SRS、Redroid、File Browser、Netdata 等组件。
5. 远端部署工具，通过 Paramiko 把源码、前端产物和 systemd 配置同步到 Ubuntu/Armbian 类设备。

项目具备明显的"产品雏形"：它已经有统一入口 `live.local`、服务总览、配置中心、Neko 多实例、YouTube Pro 多频道、Android Web 控制、WireGuard 远控、ShellCrash 代理、安装验收脚本。但从工程角度看，它仍有较重的"脚本型系统"特征：大文件过多、职责混杂、默认密码较多、Shell 脚本承担大量业务逻辑、前后端类型契约不够强、部署脚本和运维脚本还有较高耦合。

短期最重要的优化不是继续堆功能，而是建立稳定边界：

1. 把业务直播、系统服务、网络远控、浏览器会话、安卓控制拆成明确模块。
2. 把 `service_ctl.sh` 和 `common.sh` 的巨大脚本拆成可测试的小命令。
3. 把配置从散落的 `.env`、`.ini`、JSON、Shell 默认值逐步收敛到一个配置模型。
4. 把默认密码、sudoers、Web 写文件、Docker 控制等高风险能力加上权限、审计和初始化改密流程。

5. 给 Neko、ShellCrash、WireGuard、YouTube 推流建立端到端健康检查，而不是只检查容器或端口存在。

---

2. 项目定位与业务价值

2.1 项目定位

项目当前定位可以概括为：

> 面向 Linux 小主机、开发板或局域网服务器的一键无人直播与远程运维平台，通过 Web 控制台集中管理直播推流、代理网络、远程浏览器、Android 设备和系统服务。

它不是纯软件 SaaS，也不是纯脚本工具，而是“设备端软件栈”。它需要在真实 Linux 主机上运行，依赖 systemd、Docker、ffmpeg、PM2、ADB、Caddy、Neko、ShellCrash 等外部组件。

2.2 典型使用场景

1. 无人转播
   把抖音、TikTok 或其他直播源转推到 YouTube Live、SRS、HDMI 或本地播放链路。

2. 多频道 YouTube 运营
   为多个 YouTube 直播频道分别保存推流密钥、源 URL、PM2 进程和 Neko 工作室入口。

3. 浏览器远程操作
   通过 Neko 在服务器上运行 Chrome/Chromium，用于 Google 账号登录、YouTube Studio 操作、Tampermonkey 自动化脚本、远程值守。

4. 局域网/远控切换
   平时通过 `live.local` 和局域网 IP 访问；远程时通过 WireGuard 切换 Neko WebRTC NAT 地址，让远端访问不再依赖内网直连。

5. Android 云控
   对 Redroid 或真实 Android 设备进行 ADB 发现、截图、点击、输入、包管理、UI 节点读取和 scrcpy 视频流控制。

6. Homelab 管理
   通过 File Browser、WebTTY、Netdata、Cockpit、Uptime Kuma、VS Code Server 等模块把设备做成可维护的运维节点。

2.3 商业价值

项目的业务价值主要来自四个方面：

1. 降低无人直播设备搭建成本
   使用者不需要分别配置 ffmpeg、PM2、SRS、Neko、ShellCrash、Android、Caddy、mDNS，而是通过一键安装和统一控制台完成。

2. 把直播工作流产品化
   YouTube 推流密钥、抖音 URL、日志、进程状态、Neko 工作室入口被聚合在同一个面板里，减少人工 SSH 操作。

3. 适合国内网络环境
   ShellCrash、Neko 代理注入、Google/YouTube 登录入口、WireGuard 远控等设计明显面向中国大陆设备访问海外平台的现实问题。

4. 可复制部署
   如果工程化进一步完善，项目可以形成"设备镜像 + 一键部署 + 远程运维"的交付模式。

---

3. 总体架构

3.1 四层架构



3.2 核心分层

层级	职责	主要文件
基础系统层	用户、sudoers、systemd、Docker、Caddy、mDNS、网络、硬件探测	`scripts/linux/lib/common.sh`, `scripts/linux/install_*.sh`, `scripts/hardware_probe.py`
控制平面层	API、服务状态、配置托管、部署健康检查、系统快照	`web.py`, `src/control_plane.py`, `src/hub_routes.py`
业务直播层	抖音/TikTok/YouTube 直播源解析、录制、转推、PM2 进程	`main.py`, `youtube.py`, `tiktok.py`, `src/spider.py`, `src/stream.py`, `douyin_youtube_ffplay.py`
浏览器与远控层	Neko 多实例、Google/YouTube Chrome、WireGuard 切换、ShellCrash 代理	`services/neko/*`, `tools/render_neko_compose.py`, `src/wireguard_config.py`, `scripts/linux/service_ctl.sh`
Android 层	ADB、Redroid、截图、输入、scrcpy H.264 WebSocket	`src/android_control.py`, `src/redroid_control.py`, `src/scrcpy_stream.py`
前端产品层	总控台、高级控制台、Android 页面、ShellCrash 页面	`web-console/components/live-product/*`, `web-console/components/console/*`, `web-console/app/*`
验收与部署层	远端部署、安装验证、live.local 探测、Neko 验证	`tools/deploy_live_paramiko.py`, `tools/verify_local_install.py`, `tools/verify_live_console.py`

---

4. 功能清单

4.1 Web 控制台

前端位于 `web-console`，使用 Next.js 14、React 18、Tailwind、lucide-react、framer-motion、echarts、three。

主要页面：

路径	页面	功能
`/`	产品化控制台	总览、服务、YouTube、TikTok、Android、网络、设置
`/console`	高级工作台	模块、ShellCrash、配置中心、系统快照
`/android`	Android 控制页	设备选择、截图、点击、scrcpy
`/shellcrash`	ShellCrash 控制页	代理配置、面板入口

关键组件：

组件	职责
`LiveControlApp.tsx`	主产品控制台，聚合所有模块，是前端最大文件
`LiveYoutubeUnmannedView.tsx`	YouTube 单频道/多频道 Pro 工作流
`LiveTiktokHdmiView.tsx`	TikTok HDMI 链路控制
`LiveAndroidWorkstation.tsx`	Android 设备管理、包管理、UI 节点
`LiveAndroidStreamConsole.tsx`	Android 视频流和截图降级控制
`LiveConsoleWorkspace.tsx`	高级配置和模块控制台
`ConfigAdvisorPanel.tsx`	配置分析、优化、备份、恢复
`LiveStudioScene3D.tsx`	首页 3D 视觉组件

评价：

1. 前端功能完整度较高，已经超出“脚本控制面板”。
2. `LiveControlApp.tsx` 超过 2500 行，承担过多状态、渲染和业务逻辑。
3. API 类型在前端手写，后端 Pydantic 类型没有自动生成 TypeScript 契约。
4. 页面有大量业务文案和容错提示，对操作者友好，但长期维护成本偏高。

4.2 后端 API

后端入口是 `web.py`，FastAPI 负责静态页面、传统业务 API、新 `/hub/*` API、Android API 和配置 API。

核心 API 分组：

分组	示例接口	功能
健康与部署	`/health`, `/deploy/check`, `/server_info`	健康检查、部署验收
栈状态	`/stack/summary`, `/stack/services`, `/hub/dashboard`	服务、硬件、网络、业务进程总览
服务控制	`/service_action`	对 ShellCrash、Neko、SRS、WebTTY 等执行 start/stop/restart
配置中心	`/managed_roots`, `/managed_files`, `/managed_file`	托管根下文件读写删除
配置治理	`/config_insights`, `/config_optimize`, `/config_restore`	配置分析、自动修复、备份恢复
ShellCrash	`/shellcrash/apply_profile`, `/shellcrash/web_panel`	应用 YAML、安装/打开 zashboard
WireGuard	`/hub/network/wireguard/*`	导入、保存、应用、状态、Neko 远控切换
YouTube Pro	`/hub/youtube_pro_channels`	多频道配置持久化和 Neko 容量联动
Android	`/android/devices`, `/android/action`, `/android/screenshot`, `/android/scrcpy/ws`	ADB 设备发现、点击、截图、视频流
Redroid	`/redroid/summary`, `/redroid/action`	Redroid 实例状态和操作
直播进程	`/start`, `/stop`, `/restart`, `/status`, `/process_monitor`	PM2 或本地进程启停和状态

评价：

1. API 覆盖面广，能支撑实际产品使用。
2. 旧 API 与新 `/hub/*` API 并存，迁移边界需要继续收敛。
3. 部分接口承担文件写入、sudo 服务控制、Docker 操作，必须配合权限边界和审计。
4. 后端已经有较多单元测试，是该项目较好的工程基础。

4.3 直播源解析与推流

核心文件：

文件	行数	职责
`src/spider.py`	约 3397	多平台直播间信息抓取、登录、签名、流地址获取
`src/stream.py`	约 450	按平台提取 m3u8/flv/原画/多清晰度 URL
`main.py`	约 2213	传统录制主程序
`youtube.py`	约 2201	YouTube 推流业务入口
`tiktok.py`	约 2276	TikTok 业务入口
`douyin_youtube_ffplay.py`	约 1000+	抖音到 YouTube/ffplay/ffmpeg 链路辅助

支持平台非常多，包括抖音、TikTok、快手、虎牙、斗鱼、YY、B 站、小红书、Bigo、SOOP、Twitch、YouTube、淘宝、京东等。

业务模式：

1. 从 URL_config 或前端写入的 URL 读取直播间地址。
2. 根据平台调用 `spider.py` 获取直播间状态和流地址。
3. 根据配置选择清晰度、协议、代理、cookies。
4. ffmpeg 拉流、录制或转推到 YouTube RTMPS/SRS/HDMI。
5. PM2 或本地进程 registry 维持进程生命周期。
6. Web 控制台读取日志和 ffmpeg 子进程判断业务状态。

评价：

1. 平台覆盖广，是项目原始核心资产。
2. 大文件和平台逻辑混杂严重，维护新平台或修复单平台容易影响其他平台。
3. 部分平台依赖逆向签名、cookies、登录状态，稳定性天然受平台变动影响。
4. 当前业务状态主要靠日志解析和进程检测，应逐步改成结构化事件。

4.4 YouTube Pro 多频道

关键文件：

文件	职责
`src/hub_kv_store.py`	用 SQLite 保存多频道列表，数据库是 `config/hub_state.sqlite`
`src/youtube_pro_runtime.py`	规范化频道、生成独立 ini、URL_config、PM2 名称
`src/neko_capacity.py`	根据频道数量自动提升 Neko 实例数
`web-console/lib/youtubeProChannels.ts`	前端多频道本地缓存与 API 同步
`LiveYoutubeUnmannedView.tsx`	前端单频道/多频道操作界面

当前能力：

1. 多频道列表持久化到 SQLite，而不是仅保存在浏览器 localStorage。
2. 每个频道可独立设置 PM2 名、YouTube key、抖音 URL。
3. 每个频道生成独立 `youtube.<pm2>.ini` 和 `URL_config.<pm2>.ini`。
4. 保存频道时会调用 Neko 容量逻辑，确保 Neko 实例数量不少于频道数量，最低 2 个，最高 64 个。
5. PM2 名称使用 `youtube2__<channelId>` 规则，避免与旧进程名冲突。

业务价值：

1. 把“多频道无人直播”从手工复制配置变成面板操作。
2. 每条频道和 Neko 工作室形成可预期映射，便于登录不同 YouTube 账号。
3. 对业务扩容有明显意义：新增频道不再需要手改 Docker Compose。

主要风险：

1. 频道数与机器资源没有强约束，64 个只是配置上限，不代表硬件可承载。
2. 多频道的日志、错误、推流状态仍需要更细粒度结构化。
3. Neko 容量扩容会触发容器重建，需要明确告知用户会短暂影响浏览器会话。

4.5 Neko 浏览器系统

Neko 是当前项目的关键模块之一，作用是提供远程 Chrome/Chromium 桌面，用于 Google/YouTube 登录、YouTube Studio 操作、Tampermonkey 自动化等。

关键文件：

文件	职责
`services/neko/docker-compose.yml`	兼容模板和人工排障参考
`tools/render_neko_compose.py`	根据环境变量动态生成 `docker-compose.generated.yml`
`services/neko/browser-launch.sh`	Chrome/Chromium 启动器、profile 修复、代理注入、健康检查
`services/neko/policies/*`	Chrome managed policy
`services/neko/tampermonkey-seed/*`	Tampermonkey 扩展和脚本种子
`scripts/linux/neko_ip_watch.sh`	监听 IP 变化并重配 Neko
`scripts/linux/service_ctl.sh`	Neko start/stop/restart/reconfigure/status

当前能力：

1. 支持 x86_64 使用 `ghcr.io/m1k1o/neko/google-chrome:3.0.0`，ARM 使用 Chromium 镜像。
2. 默认实例数不少于 2，按 `NEKO_INSTANCE_COUNT` 动态扩展。
3. 每个实例独立 HTTP 端口、WebRTC TCPMUX/UDPMUX、UDP range、profile 目录。
4. profile 持久化，保留 Google 登录态、Tampermonkey、YouTube Studio 状态。
5. 启动前清理 Chrome 锁文件、缓存、Crashpad、DevToolsActivePort 等容易导致黑屏或无法启动的状态。
6. 可以按 `NEKO_BROWSER_RENDER_MODE` 选择 GPU、软件、legacy 渲染。
7. ShellCrash 可用时才注入 `NEKO_BROWSER_PROXY=http://host.docker.internal:7890`，避免代理端口不可用导致 Chrome 进入错误页。
8. `NEKO_ACCESS_MODE=lan|remote` 支持局域网/远控切换，远控时使用 WireGuard 地址写入 `NEKO_WEBRTC_NAT1TO1`。

近期已修正的关键点：

1. Neko 与 WireGuard 的关系改为可配置，默认 LAN，不再无条件把 WireGuard 地址写入 Neko。
2. 只有 WireGuard 启动成功且用户切到远控时，才修改 Neko Docker 环境并重启容器。
3. Neko 不会因为切换 NAT1TO1 而清理 profile/data。
4. ShellCrash 代理只在服务 active 且 7890 监听在非 loopback 地址时注入。
5. ShellCrash 启动路径已改为 systemd enable，避免重启后 inactive。

优化重点：

1. 增加 Neko 浏览器真实页面探测：CDP 截图、目标 URL、错误页检测、代理失败检测。
2. 给 Neko 显示“容器健康”和“Chrome 健康”两个不同状态，避免只看 Docker healthy。
3. 增加每个实例的资源估算：CPU、内存、带宽、WebRTC 端口占用。
4. 对 profile 做版本迁移和备份，而不是无限复用同一个目录。

4.6 ShellCrash / Mihomo 代理

ShellCrash 的角色是提供本机代理和路由策略，当前主要服务于：

1. Linux 主机访问 GitHub、Google、YouTube 等。
2. Neko Chrome 通过 `host.docker.internal:7890` 使用代理。
3. YouTube 推流前网络预检。
4. Web 面板配置 YAML 和节点选择。

关键文件：

文件	职责
`scripts/linux/service_ctl.sh`	ShellCrash 状态、启动、重启、配置同步、zashboard
`scripts/linux/lib/common.sh`	ShellCrash 安装和默认加固
`src/control_plane.py`	ShellCrash YAML 规范化、Web 面板、托管文件
`src/network_preflight.py`	YouTube/ShellCrash 网络预检

当前能力：

1. Web 控制台可编辑 ShellCrash `configs`、`yamls`、`jsons`。
2. 应用 profile 时规范化 YAML：mixed-port、allow-lan、external-controller、规则补齐。
3. 会同步 `/etc/ShellCrash/yamls/config.yaml` 到 `/tmp/ShellCrash/config.yaml`。
4. 会确保 `/tmp/ShellCrash/CrashCore` 存在。
5. 会安装/修复 zashboard Web UI。
6. 当前修复后，项目启动路径会 `systemctl enable --now shellcrash.service`，避免服务只启动不自启。

风险：

1. Web 可写代理配置，必须限制路径和文件类型，目前有 managed root 限制，但还需要审计日志。

2. ShellCrash 节点可用性不能只看服务端口，需要端到端访问 Google/YouTube。
3. 默认 `allow-lan: true` 方便 Neko 容器访问，但也扩大了局域网暴露面。
4. 当前配置默认口令和 Web 入口需要更强访问控制。

4.7 WireGuard 与远控

关键文件：

文件	职责
`src/wireguard_config.py`	WireGuard 配置解析、保存、应用、运行状态、Neko 远控
`src/hub_routes.py`	`/hub/network/wireguard/*` API
`web-console/components/live-product/LiveControlApp.tsx`	前端网络页、导入、状态、局域网/远控 switch
`docs/wg-easy-operations.md`	运维记录和推荐参数

当前能力：

1. Web 导入 WireGuard `.conf`。
2. 对 `PrivateKey` 等敏感字段进行脱敏显示。
3. 支持保存到 `config/wireguard/imports/<name>.conf`。
4. 可应用到 `/etc/wireguard/<name>.conf` 并启动 `wg-quick@<name>`。
5. 能检测本机 WireGuard 接口、peer、allowed IP、handshake、endpoint 路由。
6. 远控模式会选择可用 WireGuard 地址，并重配 Neko NAT1TO1。
7. 可生成 Windows Endpoint 旁路路由建议，避免 Windows Mihomo/TUN 抢走 WireGuard Endpoint。

优化重点：

1. 把 WireGuard 应用操作从 Web 表单进一步加权限确认，避免错误配置导致远程断连。
2. 对 Endpoint 可达性、UDP 51820、防火墙、MTU、keepalive 给出更结构化的诊断。
3. 给“局域网/远控”切换增加状态机：未连接、连接中、可切换、已切换、回滚。
4. 增加切换前后 WebRTC 媒体路径验证。

4.8 Android / Redroid / Scrcpy

关键文件：

文件	职责
`src/android_control.py`	ADB 设备、截图、点击、输入、包管理、UI 节点
`src/redroid_control.py`	Redroid Docker 实例和 ADB 连接
`src/scrcpy_stream.py`	scrcpy server、H.264 WebSocket
`src/scrcpy_vendor.py`	scrcpy jar 准备
`LiveAndroidStreamConsole.tsx`	前端视频流和截图降级
`LiveAndroidWorkstation.tsx`	Android 工作台

当前能力：

1. 发现 ADB 设备并区分 USB、TCP、Redroid。
2. 执行 tap、double tap、long press、text、swipe、keyevent。
3. 拉取截图并把坐标映射到设备屏幕。
4. 读取 UI XML 节点并提供快速点击。
5. 列出和搜索应用包，可启动/停止/清理。
6. 支持 scrcpy H.264 WebSocket，失败时前端可降级到截图轮询。
7. Redroid 支持实例状态、Docker 运行条件和 `adb connect`。

优化重点：

1. WebSocket 真流、截图轮询、ADB 命令三条链路需要独立健康状态。
2. 增加 Android 操作审计，记录谁对哪个设备执行了什么操作。
3. Redroid 启动失败时，应显示 binder、ashmem、GPU、镜像拉取、端口冲突的明确原因。
4. scrcpy jar 版本、WebCodecs 支持、H.264 SPS/PPS 状态应显示在诊断面板。

4.9 配置中心与治理

关键文件：

文件	职责
`src/control_plane.py`	managed roots、文件读写、服务规格、部署检查
`src/config_governance.py`	配置分析、备份、优化、恢复
`src/live_config/loader.py`	JSON 配置根加载
`config/system-stack.env.example`	主运行环境变量模板
`live-platform/config/defaults/*`	JSON 默认配置中心

托管根：

1. `app-config`：`config.ini`，`youtube.ini`，`URL_config.ini`，`runtime.env` 等。
2. `stack-config`：`system-stack.env`。
3. `shellcrash-configs`：ShellCrash cfg/env/list/txt。
4. `shellcrash-yamls`：ShellCrash YAML。
5. `shellcrash-jsons`：sing-box JSON。
6. `srs-config`：SRS 配置。
7. `homepage-config`：Homepage YAML/JSON。

当前治理能力：

1. 文件类型白名单。
2. 禁止绝对路径和路径穿越。
3. 保存前备份。
4. URL_config 去重、INI 关键项检查、env/json 分析。
5. 可恢复历史备份。

优化重点：

1. 配置 schema 化，使用 Pydantic 或 JSON Schema 统一校验。
2. 增加“配置变更影响范围”：哪些服务需要重启、是否会影响在线直播。
3. 敏感字段脱敏，例如 YouTube stream key、代理订阅、密码、WireGuard 私钥。
4. 配置保存动作写入审计日志。

---

5. 部署与运行链路

5.1 一键安装

入口：

文件	说明
`install.sh`	生产 bootstrap，负责下载源码、加锁、日志、分发 profile、验收
`install-core.sh`	顶层兼容入口
`install-live.sh`	顶层兼容入口
`install-all.sh`	顶层兼容入口
`scripts/linux/install_stack.sh`	profile 分发：hub/business/all
`scripts/linux/install_hub.sh`	基础 hub
`scripts/linux/install_business.sh`	业务直播模块
`scripts/linux/install_all.sh`	hub + business

安装流程大致为：

1. 检测 Debian/Ubuntu/Armbian 类系统。
2. 安装 bootstrap 依赖：git、curl、rsync、flock、sudo。
3. 获取源码到目标目录。
4. 加载 `config/system-stack.env`。
5. 创建 `live` 用户，准备运行目录。
6. 安装 Python、Node、Docker、ffmpeg、ADB 等基础依赖。
7. 准备前端静态产物和 FastAPI systemd 服务。
8. 安装 ShellCrash、Neko、Redroid、WebTTY、File Browser、Netdata、SRS 等可选模块。
9. 运行硬件探测。
10. 安装 Caddy live-edge，使 `http://live.local` 指向控制台。
11. 执行验收脚本。

5.2 远端部署

`tools/deploy_live_paramiko.py` 是当前实际远端部署的重要工具。它会：

1. 连接 SSH。
2. 校验远端源码目录。
3. 调整远端源码 owner。
4. SFTP 上传项目文件。

5. chmod 运行脚本。
6. 恢复 runtime 可写目录权限。
7. 准备远端 Python venv。
8. 检查 `loguru`、`distro`、`fastapi`、`uvicorn`。
9. 做 Python compile。
10. 构建或复用 web-console out。
11. 刷新 live-console systemd unit。
12. 刷新 mDNS。
13. 重启 live-console/live-edge。
14. 执行 `/deploy/check` 和 `verify_live_console`。

评价：

1. 自动化程度高，适合当前快速迭代。
2. 逻辑过长，包含上传、构建、权限、systemd、健康检查，建议后续拆分。
3. 使用密码 SSH 和 sudo stdin，适合开发阶段，不适合生产规模化。
4. 对远端环境依赖强，应增加 dry-run、回滚和分阶段部署。

5.3 运行时服务

主要 systemd / Docker 服务：

服务	作用
`live-console.service`	FastAPI 控制台
`live-edge.service`	Caddy 反向代理 80 到控制台
`live-edge-watch.service`	自愈 live-edge
`shellcrash.service`	Mihomo/ShellCrash 核心
`live-neko-ip-watch.service`	Neko NAT1TO1 监控
`live-webtty.service`	WebTTY
`live-android-panel.service`	ws-scrcpy 或 Android 面板
`live-redroid-host.service`	Redroid host 辅助
Docker `live-neko-*`	Neko 浏览器
Docker `srs`	SRS 直播服务
Docker `filebrowser`	文件管理
Docker `uptime-kuma`	监控
Docker `openclaw`	自动化工作区

---

6. 测试与质量

测试目录包含 20 多个测试文件，覆盖面已经比较明显：

测试文件	关注点
`test_control_plane_services.py`	服务控制、配置、部署报告
`test_neko_config.py`	Neko compose、代理、profile、实例
`test_wireguard_config.py`	WireGuard 解析、保存、状态、Neko 远控
`test_web_process_backend.py`	PM2/本地进程、ffmpeg 子进程
`test_web_dynamic_processes.py`	动态业务进程
`test_config_governance.py`	配置治理、备份、优化
`test_verify_local_install.py`	本地安装验收
`test_android_redroid.py`	Android/Redroid
`test_youtube_pro_runtime.py`	YouTube Pro 文件生成
`test_neko_capacity.py`	Neko 容量

近期实际验证：

1. `python -m unittest tests.test_control_plane_services tests.test_neko_config` 通过。
2. 远端 `.venv/bin/python -m unittest tests.test_control_plane_services` 通过。
3. 远端 `bash -n scripts/linux/service_ctl.sh` 通过。
4. 远端 ShellCrash active/enabled，端口监听。
5. 远端 Neko 两个容器 healthy，代理已接入 ShellCrash。
6. 远端通过 ShellCrash 代理访问 Google/YouTube `generate_204` 返回 204。

质量评价：

1. 单元测试意识强，尤其 Neko/WireGuard/控制平面已有较多测试。
2. 测试偏“字符串断言”和“脚本存在断言”，真实集成测试仍不足。
3. 前端没有明显的组件测试或端到端浏览器测试。
4. Shell 脚本缺少 shellcheck 和模块化测试。
5. 对部署、systemd、Docker、Neko 真实运行的验证依赖手工或远端脚本。

---

7. 代码结构与主要问题

7.1 大文件问题

当前主要大文件：

文件	约行数	问题
`src/spider.py`	3397	多平台抓取混在一个文件
`scripts/linux/lib/common.sh`	2725	安装、权限、服务、Docker 全混合
`LiveControlApp.tsx`	2550	前端状态、UI、业务逻辑过大
`scripts/linux/service_ctl.sh`	2312	服务控制总线过大
`tiktok.py`	2276	业务入口大文件
`main.py`	2213	传统业务逻辑大文件
`youtube.py`	2201	YouTube 业务大文件
`LiveConsoleWorkspace.tsx`	1776	高级控制台过大
`web.py`	1704	API、静态、业务、文件操作集中
`src/wireguard_config.py`	1330	WireGuard 与 Neko 远控逻辑混合

风险：

1. 新人很难判断修改边界。
2. 单个 bug fix 容易影响无关逻辑。
3. 测试粒度难以精确。
4. review 成本高。

7.2 职责混杂

典型混杂点：

1. `web.py` 同时负责 API、静态文件、直播进程、Android、配置、ShellCrash、路由兼容。
2. `service_ctl.sh` 同时负责 mDNS、Wi-Fi、ShellCrash、Neko、SRS、WebTTY、Docker 模块。
3. `common.sh` 同时负责安装、权限、服务单元、Docker compose、浏览器、Neko、Redroid、Caddy。
4. `spider.py` 同时负责几十个平台的抓取和登录。
5. `LiveControlApp.tsx` 同时负责所有产品页面。

建议：

1. Python 后端按模块拆 router：`routes/services.py`，`routes/android.py`，`routes/network.py`，`routes/config.py`，`routes/live.py`。
2. Shell 控制按服务拆：`service_ctl/neko.sh`，`service_ctl/shellcrash.sh`，`service_ctl/srs.sh`。
3. 把平台抓取拆为 `src/platforms/douyin.py`，`src/platforms/tiktok.py`，`src/platforms/bilibili.py` 等。
4. 前端按页面拆 state hook：`useDashboard`，`useWireGuard`，`useNeko`，`useServices`，`useAndroid`。

7.3 默认密码与权限风险

源码中存在多个默认密码或 token：

项	默认值
WebTTY	`live / 12345678`
File Browser	`live / 12345678`
Wi-Fi onboarding	`live / 12345678`
Neko user/admin	`12345678`
VS Code Server	`12345678`
OpenClaw token	`12345678`
部署工具默认 SSH 密码	`orangepi`

高风险点：

1. `live` 用户 sudoers 允许无密码执行 `service_ctl.sh *`。
2. Web 控制台可以触发服务启停、Docker、systemd、文件写入。
3. Neko profile 权限里出现 `chmod 0777`。
4. ShellCrash `allow-lan` 使代理端口对局域网可用。
5. 控制台显示 Neko 默认口令，便利但有泄露风险。

优化建议：

1. 首次启动强制改密，生成随机密码写入 `/opt/live/config/secrets.env`。
2. 控制台口令类字段默认隐藏，只显示是否已设置。
3. service_ctl sudoers 限制为具体 allowlist 子命令，不允许任意参数扩展。
4. 管理接口增加登录鉴权，至少支持本地 token。

5. ShellCrash LAN 代理默认只允许 Neko Docker 网段或本机。
6. 部署工具改为 SSH key + sudo NOPASSWD 指定命令，减少密码传递。

7.4 配置分散

当前配置来源：

1. `config/system-stack.env`
2. `config/*.ini`
3. `live-platform/config/defaults/*.json`
4. Shell 脚本里的默认值
5. Docker Compose 环境变量
6. SQLite `config/hub_state.sqlite`
7. Neko profile 内部状态
8. `/etc/ShellCrash/*`
9. `/etc/wireguard/*`

问题：

1. 同一个配置可能在 env、JSON、Shell 默认值里重复。
2. 用户不容易知道改哪个文件会影响哪个服务。
3. 配置变更后是否需要重启没有统一规则。

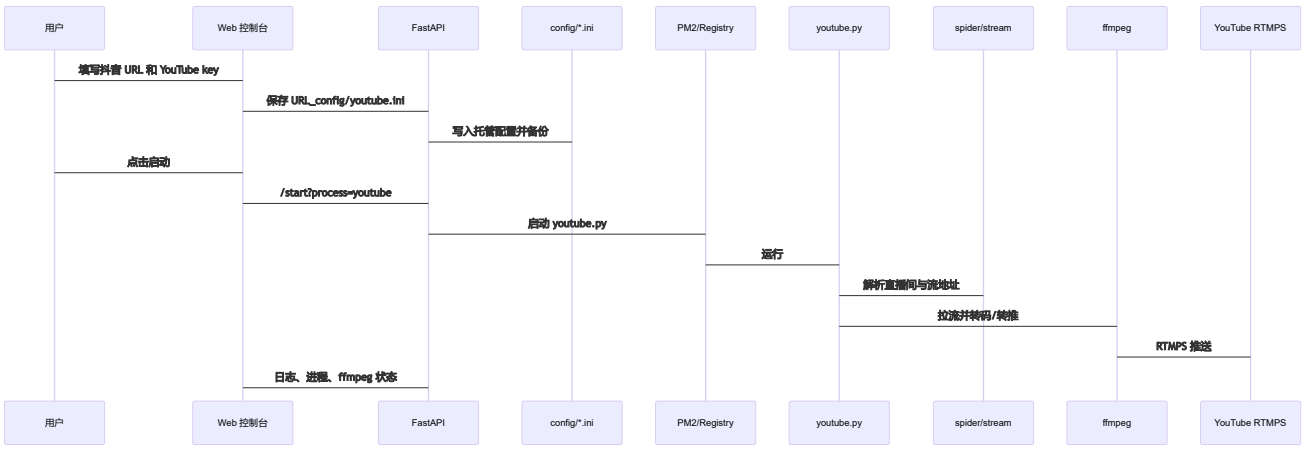
优化建议：

1. 建立统一配置模型：`LiveStackConfig`。
2. 每个配置项有：默认值、来源、敏感性、影响服务、是否需要重启、校验规则。
3. 前端配置页基于 schema 自动渲染。
4. 保存配置时返回 restart plan，而不是让用户猜。

---

8. 业务流程详解

8.1 YouTube 单频道推流流程



关键风险：

1. YouTube key 错误、过期或未填。
2. 抖音 URL 不在线或平台接口变动。
3. ShellCrash/代理不可用导致海外平台失败。
4. ffmpeg 编码参数与硬件不匹配。
5. PM2 进程在线但 ffmpeg 子进程已死。

8.2 YouTube Pro 多频道流程

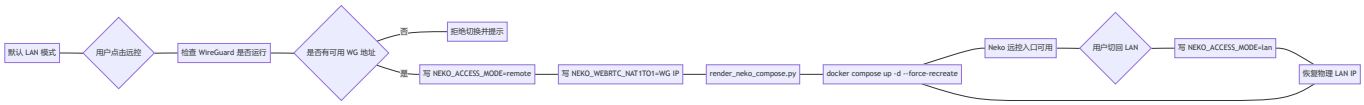
1. 用户在前端创建频道。

2. 前端保存频道列表到 `/hub/youtube_pro_channels`。
3. 后端规范化频道，写入 SQLite。
4. 后端确保每个频道有独立的 URL_config 和 youtube.ini。
5. 后端调用 `ensure_neko_capacity_for_youtube_channels`。
6. 如果频道数超过 Neko 实例数，更新 `NEKO_INSTANCE_COUNT` 并重启 Neko。
7. 前端显示每条频道对应的 Neko 工作室入口和启动按钮。

关键风险：

1. 保存频道时触发 Neko 重启，影响当前正在操作的浏览器。
2. 多频道同时推流的 CPU/内存/上行带宽瓶颈。
3. PM2 名称和配置文件映射必须稳定。

8.3 Neko 局域网/远控切换流程



已实现的关键保护：

1. WireGuard 不再无条件影响 Neko。
2. 切换只改 Docker 环境和 NAT1TO1，不清理 profile/data。
3. ShellCrash 代理只在可用时注入。

仍需优化：

1. 切换前保存当前 Neko 连接状态。
2. 切换后自动进行 WebRTC 连接测试。
3. 增加失败回滚。

8.4 ShellCrash 代理流程

1. 用户在控制台编辑 YAML。
2. 后端规范化 YAML，确保 mixed-port、allow-lan、controller、rules。

3. `service_ctl.sh` 同步 config 到 `/tmp/ShellCrash/config.yaml`。
4. 确保 CrashCore 存在。
5. systemd 启动或重启 `shellcrash.service`。
6. Neko reconfigure 时检测 ShellCrash active 和 7890 监听。
7. 可用则向 Neko 注入 `host.docker.internal:7890`。

近期真实问题结论：

ShellCrash 启动失败并不是 `config.yaml` 损坏，而是 systemd unit 处于 `disabled`，项目控制台原先优先走 `/etc/ShellCrash/start.sh`，只启动不 enable。已调整为优先 systemd 并 enable。

8.5 Android 控制流程

1. 前端获取 `/android/devices`。
2. 选择设备 serial。
3. 优先尝试 `/android/scrcpy/ws` 真流。
4. 真流失败时降级 `/android/screenshot` 轮询。
5. 用户点击画面，前端换算屏幕坐标。
6. 调用 `/android/action` 执行 ADB 输入。
7. 可进一步读取 UI 节点、包列表、logcat。

风险：

1. 坐标映射要处理截图尺寸、CSS 缩放、旋转。
2. ADB 授权、无线断连、Redroid 未启动会导致空白。
3. scrcpy H.264 解码依赖浏览器 WebCodecs。

---

9. 优化方案

9.1 P0：稳定性和安全优先

P0-1：给所有高风险入口加鉴权

范围：

1. `/service_action`
2. `/managed_file`
3. `/shellcrash/apply_profile`

4. `/hub/network/wireguard/apply`
5. `/android/action`
6. `/redroid/action`
7. `/start`, `/stop`, `/restart`

建议：

1. 新增 `LIVE_ADMIN_TOKEN`，首次部署自动生成。
2. 前端登录后保存在 sessionStorage。
3. API 要求 `Authorization: Bearer <token>`。
4. 局域网也不豁免，最多允许本机 loopback 开发模式豁免。

验收：

1. 未登录不能启停服务。
2. 未登录不能写配置。
3. token 错误返回 401。

P0-2：默认密码治理

建议：

1. 首次安装生成随机密码：
   - `WEBTTY_PASS`
   - `FILEBROWSER_PASS`
   - `NEKO_USER_PASS`
   - `NEKO_ADMIN_PASS`
   - `VSCODE_SERVER_PASSWORD`
   - `OPENCLAW_GATEWAY_TOKEN`
2. 控制台显示“已设置/未设置”，不默认明文展示。
3. 提供“复制一次性初始密码”或“重置密码”功能。

P0-3：Neko 真实健康检查

当前 Docker healthy 不代表 Chrome 可用。应增加：

1. Neko HTTP 可达。
2. Chrome DevTools 9222 可达。
3. 当前页面不是 `chrome-error://`。
4. CDP 截图非空。

5. Google/YouTube 登录页能渲染。
6. 代理状态：direct / shellcrash / failed。

前端展示：

状态	含义
容器健康	Docker/Neko server alive
浏览器健康	Chrome 窗口和 CDP alive
代理健康	ShellCrash 端到端可访问 Google/YouTube
WebRTC 健康	当前访问模式下媒体连接成功

P0-4：ShellCrash 端到端代理预检

不能只看 7890 端口。增加：

1. `http://www.gstatic.com/generate_204`
2. `https://www.google.com/generate_204`
3. `https://www.youtube.com/generate_204`
4. DNS 解析检查。
5. 当前 Mihomo selector 和 now 节点。
6. 最近错误日志。

Neko 代理注入建议：

1. 如果 ShellCrash active 但 Google/YouTube 预检失败，前端必须标红。
2. 是否仍注入代理可以配置：
   - `NEKO_PROXY_REQUIRE_E2E=0`：只要求端口。
   - `NEKO_PROXY_REQUIRE_E2E=1`：要求端到端成功。

P0-5：服务控制审计

记录：

1. 时间。
2. 来源 IP。
3. 用户/token 指纹。
4. 动作：service/action。
5. 结果和错误。

文件：

1. `logs/audit.log`
2. 或 SQLite `config/hub_state.sqlite` 增加 audit 表。

9.2 P1：架构拆分

P1-1：拆分 `web.py`

目标结构：

```
src/api/
  app.py
  deps.py
  routes/
    health.py
    services.py
    config.py
    shellcrash.py
    wireguard.py
    android.py
    redroid.py
    live_process.py
    tiktok_replay.py
```

收益：

1. API 责任清晰。
2. 测试可以按模块写。
3. 新增功能不会继续膨胀 `web.py`。

P1-2：拆分 `service_ctl.sh`

目标结构：

```
scripts/linux/service_ctl.sh
scripts/linux/service_ctl/
  common.sh
  shellcrash.sh
  neko.sh
  srs.sh
  android.sh
  webtty.sh
  observability.sh
  docker_modules.sh
```

要求：

1. 保持当前 CLI 兼容：`service_ctl.sh <service> <action>`。
2. 每个模块只导出 status/start/stop/restart/reconfigure。
3. 对危险路径删除增加 realpath 边界检查。
4. 引入 shellcheck。

P1-3：拆分平台抓取

目标结构：

```
src/platforms/
  base.py
  douyin.py
  tiktok.py
  kuaishou.py
  bilibili.py
  twitch.py
  youtube.py
```

统一接口：

```python
class PlatformClient(Protocol):
    async def resolve_room(self, url: str, context: FetchContext) -> RoomInfo: ...
    async def resolve_streams(self, room: RoomInfo, quality: str) -> StreamSet: ...
```

收益：

1. 单个平台修复不影响其他平台。
2. 可以为每个平台单独写 mock 测试。
3. 日志和错误可以标准化。

P1-4：前端状态拆分

建议：

1. `LiveControlApp.tsx` 拆页面组件。
2. 用 hooks 管理 API：
   - `useHubDashboard`
   - `useServices`
   - `useWireGuard`
   - `useNekoLinks`
   - `useAndroidDevices`
3. 用后端 OpenAPI 生成 TypeScript 类型，减少手写接口错配。

9.3 P2：产品化和规模化

P2-1：安装配置向导

新增首次启动向导：

1. 设置主机名。
2. 修改管理员 token。
3. 修改 Neko/FileBrowser/WebTTY/VSCode 密码。
4. 选择 LAN/远控网络模式。
5. 导入 ShellCrash YAML。
6. 导入 WireGuard。
7. 选择业务模式：YouTube、TikTok HDMI、Android 云控、全量。

P2-2：CI/CD

建议流水线：

1. Python：
   - `python -m compileall`
   - `python -m unittest`
   - `ruff`
   - `mypy` 逐步引入
2. Shell：
   - `bash -n`
   - `shellcheck`

3. Frontend：
   - `npm run build`
   - TypeScript check
   - Playwright smoke test
4. Docker：
   - render Neko compose
   - compose config 校验
5. 部署：
   - staging 设备自动部署
   - `/deploy/check`
   - `verify_local_install.py`

P2-3：可观测性

新增统一状态事件：

```
{
  "ts": 1710000000,
  "module": "youtube",
  "instance": "youtube2__abc",
  "state": "streaming",
  "reason": "ffmpeg_alive",
  "metrics": {
    "bitrate_kbps": 4500,
    "fps": 30,
    "dropped_frames": 0
  }
}
```

替代目前大量日志文本推断。

P2-4：设备资源调度

Neko、Redroid、ffmpeg 都吃资源。建议：

1. 硬件探测输出容量建议。
2. YouTube Pro 频道数超过建议时提示降载。
3. 每个 Neko 实例记录内存和 CPU。
4. 推流前检查上行带宽。
5. 低端 ARM 自动降低分辨率、fps、bitrate。

---

10. Neko/Chrome 针对 Google 和 YouTube 的优化建议

当前已经做得较好的点：

1. 使用 Google Chrome 镜像优先于 Chromium。
2. 默认打开 Google 登录到 YouTube 的入口。
3. 保留 profile，不清理 Google 登录态。
4. 不再无条件禁用 Google Sync。
5. Tampermonkey 种子可持久化。
6. ShellCrash 可用时注入代理。
7. 渲染参数可配置，避免黑屏。

进一步建议：

1. 把 `NEKO_START_URL` 改成可按实例配置：
   - 实例 1：Google 登录。
   - 实例 2：YouTube Studio。
   - Pro 频道 N：对应频道 Studio Live 控制室。
2. 每个 Neko 实例独立 cookie/profile，不共享账号。
3. 对 Chrome profile 做定期备份，尤其 `Default/Cookies`、`Login Data`、`Preferences`、`Local Storage`。
4. 避免过度自动化 YouTube Studio 页面，Tampermonkey 只做"关闭弹窗/恢复布局/辅助点击"，不要做违反平台规则的行为。
5. 增加 YouTube 页面状态识别：
   - 是否登录。
   - 是否进入 Studio。
   - 是否存在直播控制室。
   - 是否 stream key 已显示。
   - 是否弹出安全验证。
6. 使用 CDP 抓取 console error 和 network error，帮助判断黑屏、代理失败、TLS 失败。
7. 对 Google/YouTube 网络走 ShellCrash，局域网和 `host.docker.internal` 走 bypass。
8. 如果 ShellCrash 节点延迟高，前端显示推荐切换节点，而不是让用户在 Neko 黑屏里猜。

---

11. 风险清单

风险	等级	说明	建议
默认密码	高	多个服务默认 `12345678`	首次启动强制改密
Web 控制系统服务	高	API 可触发 sudo/systemd/Docker	增加鉴权、审计、allowlist
Shell 脚本过大	高	`common.sh` 和 `service_ctl.sh` 难维护	模块化拆分
Neko 黑屏	高	容器 healthy 不代表 Chrome 可用	CDP 真实健康检查
ShellCrash 端口可用但节点不可用	高	端口监听不等于 Google 可访问	端到端代理预检
多频道资源过载	中高	Neko + ffmpeg + Redroid 会抢资源	容量模型和限流
配置分散	中高	env、ini、json、SQLite、Shell 默认值混杂	配置 schema 化
大文件维护难	中高	多个文件超过 1700 行	分模块重构
编码乱码	中	README/docs/注释存在显示乱码	统一 UTF-8，修复文档
部署脚本过重	中	部署、构建、权限、systemd 混在一个工具	拆分阶段、支持回滚
前后端类型错配	中	TS 类型手写	OpenAPI 生成类型
平台接口变动	中	多平台直播抓取依赖网页/API	平台插件化和监控

12. 推荐实施路线图

第 1 阶段：稳定线上可用性，1 到 2 周

1. 完成 API 鉴权和默认密码治理。
2. 给 ShellCrash 增加端到端代理健康检查。
3. 给 Neko 增加 Chrome/CDP 真实健康检查。

4. 给 WireGuard/Neko 远控切换加失败回滚。
5. 增加服务操作审计日志。
6. 修复 README/docs 编码显示问题。

第 2 阶段：拆分高风险大文件，2 到 4 周

1. 拆 `web.py` router。
2. 拆 `service_ctl.sh` 子模块。
3. 前端 `LiveControlApp` 拆 view + hook。
4. 把 YouTube Pro/Neko/WireGuard 的后端类型文档化。
5. 引入 shellcheck、ruff、前端类型检查。

第 3 阶段：业务流程产品化，4 到 8 周

1. 首次安装向导。
2. YouTube Pro 频道和 Neko 实例稳定映射。
3. 直播前 checklist：URL、key、ffmpeg、代理、Neko、带宽。
4. Android/Redroid 诊断页。
5. 部署脚本支持 staging、回滚、dry-run。
6. 设备资源模型和容量建议。

第 4 阶段：规模化交付，长期

1. 设备镜像化。
2. 多设备集中管理。
3. 远端升级通道。
4. 日志和指标集中采集。
5. 密钥管理和审计。
6. 模块市场：Neko、SRS、Android、ShellCrash、WireGuard、YouTube Pro 可单独启停安装。

---

13. 最优架构目标

长期目标应是：

```
d2ypp2
  core-api          FastAPI 控制平面
  live-business     直播源抓取与推流
  service-manager   systemd/Docker 服务控制
  network-manager   ShellCrash/WireGuard/代理检测
  browser-manager   Neko/Chrome/profile/CDP
  android-manager   ADB/Redroid/Scrcpy
  web-console        产品前端
  deploy             安装、升级、验收、回滚
```

每个 manager 对外提供稳定接口，前端只调用 API，不理解 Shell/Docker 细节。Shell 脚本只做系统操作，不包含业务判断。配置中心是唯一配置来源，服务控制根据配置生成运行环境。

---

14. 结语

该项目已经具备较强的实际交付能力：能部署、能运行、能远控、能管理浏览器、能控制安卓、能转推 YouTube，也有相当数量测试。真正需要投入的是工程边界、权限安全、真实健康检查和模块化。

优先级建议非常明确：

1. 先保线上稳定：Neko、ShellCrash、WireGuard、YouTube 推流要端到端可验证。
2. 再保安全：默认密码、控制接口、sudo、文件写入要收敛。
3. 再保可维护：大文件拆分、配置 schema、前后端类型生成。
4. 最后做规模化：多设备、自动升级、集中监控、设备镜像。

只要按这个顺序推进，`d2ypp2` 可以从“强力脚本集合”升级成“可交付、可运维、可扩展的无人直播设备平台”。