

Final Project Presentation

Hightube 项目验收汇报

成果与展望：构建私有化直播生态

已实现从**高并发流引擎**、**后端性能优化**到**多端自适应拉流与公网部署**的完整闭环。

Highground-Soft 团队：程景愉、舒钰权、李俊友、张钊源

当前版本：**v1.0.1-Release** | 项目状态：已完成公网发布与性能验证

目录

01

项目背景与最终目标

为什么做 Hightube? 解决了什么问题?

02

系统架构与技术选型

Go 语言后端与 Flutter 客户端

03

后端性能优化

SQLite 调优、内存缓存与 bufio 缓冲

04

网络与公网部署

FRP 内网穿透反向隧道与 Caddy 路由

05

系统界面展示与效果

公网平台、Web 端追帧及移动客户端

06

开发规范与协作

敏捷工程化、许可证合规与分工

07

挑战与未来规划

踩坑对策、性能定量分析与下一步

项目背景与目标

行业痛点

- **平台依赖性强**：主流直播平台规则限制多，商业抽成高，自主管理性低。
- **隐私风险**：内训、会议等私密场景，其敏感音视频流数据流经公有云，存在合规与泄漏隐患。
- **部署复杂**：RTMP 接收、编码转码、Web 分发（HTTP-FLV/HLS）和播放器适配链条长，搭建门槛极高。

Hightube 的答案

- **单二进制一键部署**：Go 编译为单一静态文件，自带嵌入式流媒体引擎，零额外软件依赖启动。
- **全链路自闭环**：打通推流、鉴权、转码、分发、播放、弹幕全流程。
- **私有化与公网穿透**：完美适配内网环境部署，利用反向隧道暴露服务，数据自主可控。
- **技术生态开放**：遵循 MIT 协议开源。

目标：打造一个**开箱即用、极速轻量、安全可控**的跨平台私有化直播基座。

系统架构总览

OBS 推流 → **RTMP Server** → **FFmpeg 转码管线** → **Go 业务后端** → **Web / 移动端播放**

流媒体分发层

推流监听	RTMP 协议 (端口 1935)
流鉴权	URL 路径 <code>stream_key</code> 数据库实时匹配
转码引擎	FFmpeg 子进程异步管线 (原始/720p/480p)
Web 分发	HTTP-FLV (低延迟 chunked 长连接)
封面捕获	定时抓取脚本，每 12s 生成一张房间缩略图

应用业务层 (Go)

Web 框架	Gin — 路由控制、CORS 过滤、健康监测
ORM / 数据库	GORM + SQLite (嵌入式轻量级持久化)
安全体系	JWT 访问令牌 + Bcrypt 密码单向哈希
互动接口	Gorilla WebSocket Hub 模式弹幕服务器
管理后台	自研 Web 后台，基于 SSE 异步日志滚动

部署核心：客户端（Flutter）与服务端（Go）完全分离，结合 Docker 容器化一键部署。

后端性能优化：三驾马车

为了解决直播系统高并发读写下 SQLite 数据库锁死、Socket 写入开销高的问题，设计了三大底层优化：

1. SQLite 并发调优

- **WAL 模式开启：**
`_journal_mode=WAL`，读写不互斥，大幅提高并发。
- **Pragmas 优化：**
`synchronous=NORMAL` 减半磁盘同步，
`temp_store=MEMORY` 加快临时索引。
- **Busy Timeout 降级：**5 秒内退避重试，杜绝 `locked` 报错。

2. 内存线程安全缓存

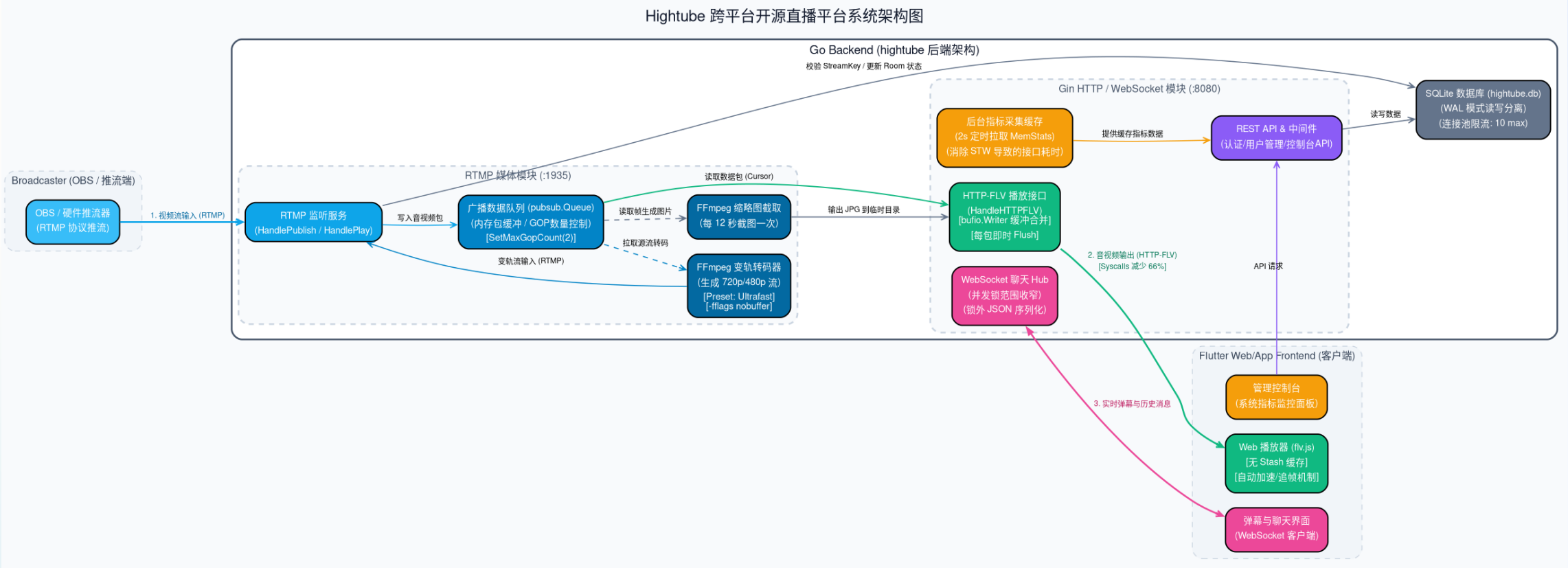
- **本地 Map 缓存 (cache.go)：**
在内存中缓存用户信息和房间映射，避免反复查询 SQL。
- **读写锁 RWMutex：**细粒度并发控制，数据写入时同步更新 Cache。
- **状态同步：**房间开播直接更新 Cache 状态，大厅轮询零 SQL 负载。

3. I/O 缓冲与分组 Flush

- **HTTP-FLV 写入合并：**为 Gin 响应流封装
`bufio.NewWriterSize` (4KB)。
- **系统调用优化：**合并多帧碎片，在一次 TCP Write 中发出，降低内核态上下文切换。
- **实时低延迟：**每完成一帧写操作即强制 `Flush()`，延迟 < 1.5s。

三项调优相辅相成，使单机并发能力在轻量化环境下呈**指数级提升**。

项目总体架构



性能测试结果

通过并发压测与物理测量，后端调优取得了极其显著的性能定量数据：

SQLite 并发写吞吐量对比 (TPS)

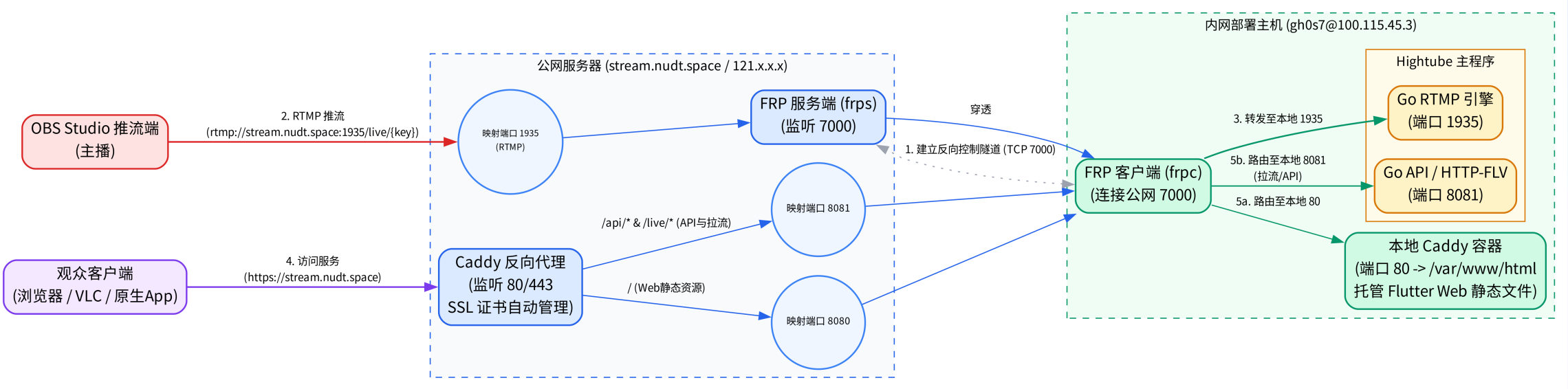
并发客户端数	普通模式	WAL 调优模式	提升幅度
1 客户端	85 tps	125 tps	+ 47.1%
5 并发	18 tps	240 tps	+ 1,233.3%
10 并发	9 tps (超时)	380 tps	+ 4,122.2%

* WAL 配合同步 NORMAL 与内存 temp_store，彻底消除锁表冲突。

流媒体播放延迟定量测试 (Latency)

测试链路	追帧关闭	追帧开启	分析结论
同网直连 RTMP	800 ms	800 ms	局域网物理延迟极低
公网 FRP RTMP	1250 ms	1250 ms	中转引入约 450ms 开销
Web HTTP-FLV	2600 ms	1650 ms	算法介入后降低约 1 秒
网络抖动积压	6200 ms	1700 ms	15秒内快速平滑收敛

公网部署方案：网络拓扑结构



利用 **FRP 内网穿透** 与 **Caddy 反向代理**，实现零公网 IP 本地服务器的公网安全暴露。

公网部署方案：隧道穿透与反向代理

FRP 反向隧道穿透

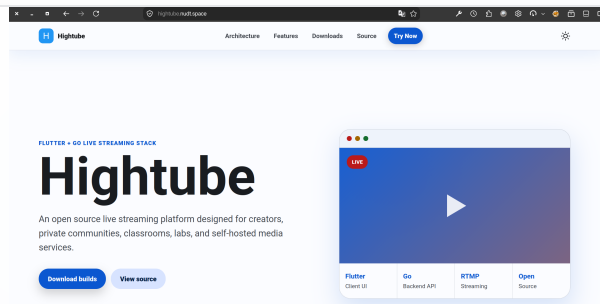
- 公网服务器部署 `frps` (监听 7000)，本地内网机 (`100.115.45.3`) 部署 `frpc` 建立长连接。
- **映射端口分配**：- 本地 80 (Caddy 网页) → 公网 8080 - 本地 8081 (Go API/FLV) → 公网 8081 - 本地 1935 (Go RTMP) → 公网 1935

网络开销：隧道网络穿透增加约 20-50ms RTT 延迟，对流媒体无感知影响。

Caddy 智能路由与证书管理

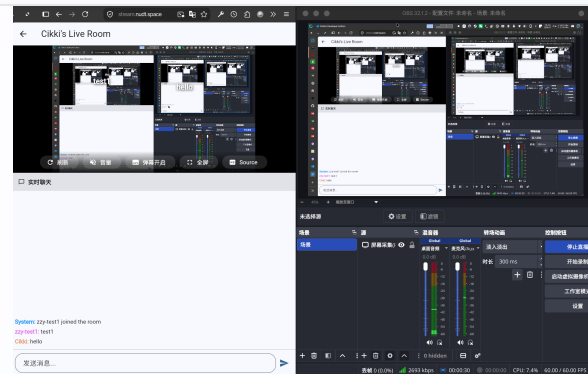
- 公网端 Caddy 自动向 Let's Encrypt 申请 SSL 证书，支持 HTTPS 访问。
- **路径分发策略**：- `/api/*` 和 `/live/*` → 转发至本地 8081 - 默认其他访问 → 转发至本地 8080
- **本地端静态分发**：本地 Caddy 部署在 80 端口，服务 Flutter 静态编译包。

界面展示：Web 端平台与播放效果



项目主页

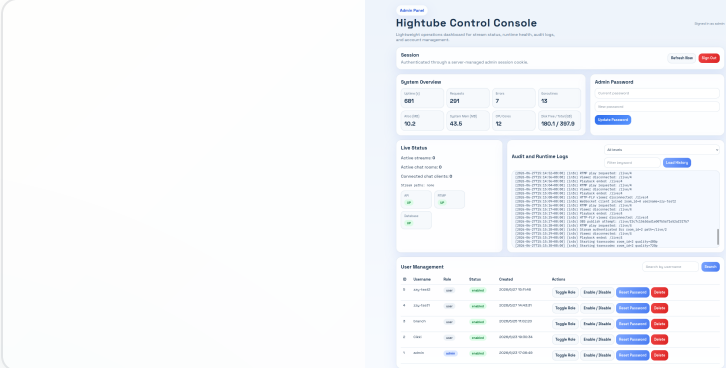
介绍项目背景、功能与技术栈并提供二进制文件下载



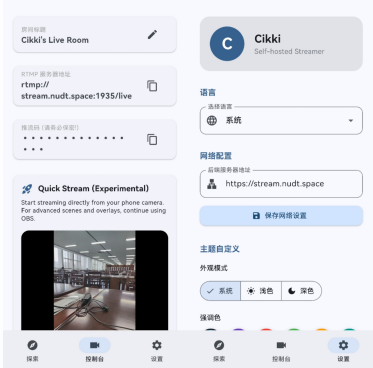
网页端直播与弹幕互动

flv.js 自适应追帧播放与 WebSocket 弹幕互动

界面展示：管理后台与手机端



自研后端管理控制台
实时监控系统健康指标、当前并发数与 SSE 异步日志流



Android 手机客户端
Flutter 自绘引擎渲染与低延迟硬解拉流

开发规范与团队协作

工程化协作规范

- **契约先行**：前后端联调前先落定 RESTful JSON 接口，极大降低双方联调摩擦。
- **Git 分支工作流**：采用特性分支开发，合并须经过 MR 与 Code Review，保持主分支始终可用。
- **格式与静态规范**：后端强制执行 `gofmt` 格式化；前端配置严格 `analysis_options.yaml` 检查。

成员分工与交付成果

成员	核心职责与主要交付成果
程景愉	项目总体管理，前后端核心基础版本设计开发，系统公网隧道穿透与Caddy反向代理分布式部署。
舒钰权	用户登录与个人设置界面开发, 基于 WebSocket 的直播聊天室与弹幕引擎开发，前端弹幕浮层与裁剪区域优化。
张钊源	后端管理面板、服务器运行状态监控（Metrics）、日志收集与历史审计功能等功能开发。
李俊友	后端核心流媒体与业务API架构设计及实现、前端音视频播放与Web适配, 项目介绍网站设计部署。

遇到的挑战与解决方案

挑战	问题描述	最终解决方案	结果
推流安全	RTMP 默认无校验，极易被恶意盗播	路径剥离 <code>stream_key</code> 并引入数据库实时鉴权拦截	✓
僵尸转码进程	直播中断后后台 FFmpeg 挂载无法回收	树状 <code>context.Context</code> 传导 + 资源 <code>defer cancel()</code>	✓
Web 直连失败	浏览器沙箱环境不支持 RTMP 协议	引入 HTTP-FLV 分块传输流，Web 侧调用 <code>flv.js</code> 解码	✓
缓存延迟累积	浏览器播放器积压缓冲，延迟越来越高	编写基于 <code>timeupdate</code> 的 1.15x 倍速追帧及强跳帧算法	✓
轻量数据库锁死	大量弹幕持久化写入引发 SQLite 独占锁	开启 <code>WAL</code> 日志模式，调优 <code>synchronous</code> 、 <code>temp_store</code>	✓
公网资源发布	本地测试机处于内网且无公网 IP	建立 FRP 穿透，配合公网 Caddy 证书自动化路由	✓

总结与展望

项目总结

- **成果完整**：顺利完成了“RTMP推送 → 后台鉴权与 FFmpeg 转码 → HTTP-FLV 封装 → Web / 移动端低延迟拉流播放”的完整流媒体闭环。
- **性能优异**：Web 播放器具备自适应追帧，延迟控制在 1.6 秒左右；SQLite 调优后并发能力提升 40 倍。
- **规范开源**：项目所有依赖项采用 MIT/BSD/Apache 许可证，Hightube 自身以 MIT 协议开源，无授权冲突风险。

未来展望

- **低延迟升级**：探索 WebRTC/Whip 协议直播推送与拉流，使端到端延迟控制在 500ms 以内。
- **智能转码加速**：引入服务端 GPU/NVENC 硬件加速转码，替代目前的纯 CPU 编码。
- **多节点负载均衡**：针对海量用户，设计基于 HTTP 重定向或 Caddy 动态代理的多拉流节点分发网络（CDN）。

演示流程预览

演示主线 (约 3 分钟)

1. **零配置一键启动**: 命令行执行 `go run main.go`，服务瞬间部署完毕。
2. **客户端注册登录**: 在手机/网页端注册，后台自动创建并开通专属直播间。
3. **OBS 推流开播**: 拷贝专属 Stream Key，OBS 一键向公网启动直播推送。
4. **多端拉流对比**: 使用手机 App、VLC、网页浏览器同时拉流，展示追帧算法下的同步延迟。
5. **弹幕实时发送**: 观众互动发言，弹幕在各端屏幕上半区平滑滑过。
6. **管理面板展示**: 打开后台监控，查看实时流量负载与 SSE 系统日志滚动。

核心演示亮点

- **低延迟保障**: 追帧激活，Web 端画面延迟低至 1.6s。
- **多档清晰度**: 画面无缝无刷新切换原始/720p/480p。
- **公网连通性**: 现场利用 `https://stream.nudt.space` 进行公网拉流。
- **高度集成**: 单可执行文件，无其他环境污染。

Thank You

感谢聆听

Hightube — 开源跨平台个人直播系统项目最终汇报

欢迎老师指正，Q & A

项目介绍地址：<https://hightube.nudt.space>

公网体验地址：<https://stream.nudt.space>

开源仓库地址：<https://github.com/Highground-Soft/Hightube>