下面把"在你把 **Frida 注入功能**真正打通之前，你都做了哪些铺垫工作"梳理成**时间线 + 证据位**。这些基础把数据流、可视化、模板体系、PID/进程探测、以及安全与稳定性都先搭好了，最后 Frida 只是在这套骨架上"落一子就活"。

---

1) 把"流量进平台"的底座打牢（mitmproxy → Flask）

- **写了 mitmproxy 插件**：把每个请求的 `url/method/headers/body` 采集后**放入队列**，由后台 **worker** 以**限速 + 重试**的方式 POST 给后端，避免高并发/弱网把后端打爆。支持 `MITM_POST_INTERVAL / MITM_MAX_RETRIES / MITM_REQUEST_TIMEOUT` 环境变量调参。
- **做了简单敏感信息检测**：在插件里对 body+headers 做手机号/邮箱/Base64 token 的正则检出，命中会在前端标记"敏感字段"。
- **后端接收并入库**：Flask 落地 SQLite 表 `sessions`，保存每条会话（含敏感命中），作为你后面"点会话→一键注入"的上下文来源。

小白总结版：先把 HTTPS 流量**稳稳推送**到后端并存起来。

---

2) 建前端"可视化 + 操控台"，形成实战闭环

- **会话列表/详情 UI**：左侧列表实时刷新、右侧详情展示 header/body，方便你"选一条流量"后直接做动态分析。
- **实时 Console**（绿色终端）与控制按钮（清空/复制/放大），准备好**承接 Frida 的 stdout**。
- **WebSocket 通道**：前端连上 Socket.IO，自动刷新列表、接收后端推的 `session_new` / `sessions_cleared` / 未来的 `frida_log`。

小白总结版：把 **"看数据 + 看日志"** 的界面提前搭好。

---

3) 后端"中枢能力"：存、查、导出、广播

- **API 整理**：`/api/sessions`（增/查）、`/api/sessions/export`（CSV/JSON 导出）、`/api/clear_sessions`（清空），并通过 SocketIO 广播前端刷新。
- **敏感字段正则（后端兜底）**：即使来源端检出失败，后端也做一次轻量检出，显示在列表"敏感字段"列。

小白总结版：后端把"数据服务端的一切"先打通。

---

4) 为 Frida 注入做"脚本与运行"的骨架

- **模板仓库**：在后端集中维护 `FRIDA_TEMPLATES`，包含 `okhttp_log_url / requestbody_dump / ssl_pinning_bypass / sharedprefs_dump / runtime_probe` 等，后续扩展也只需加模板。
- **安全渲染器**：实现 `safe_format`，容错填充 `{param}`，缺参不崩，保证模板可复用。
- **"生成脚本"接口**：`/api/generate_frida` 可把模板或基于会话自动选的模板（有 body → 抓 `RequestBody`，否则抓 URL）渲染出来供你下载/排查。

- **消息包装约定**：统一要求模板里把 `console.log` 走 `send({__frida_console:true, args:[...]})`，后端再**转发成** `frida_log` 到前端 Console——这就是你后来能看到 `[WRAP]` / `[bypass]` / `[SharedPreferences]` 的基础。

小白总结版：把 **"模板库 + 渲染 + 日志协议"** 全部先定好了。

---

5) 进程探测 & 选择（防止"附错进程/Java 不可用"）

- **Probe** 接口：`/api/probe` 调 `frida-ps -Uai`，把和包名匹配的进程列出来供你判断。
- **主进程优选器**：`pick_exact_pid_from_ps` 从 ps 输出里尽量选"**恰好等于包名**"那行（规避 `:remote/:push`），为后续注入选到正确 PID。

小白总结版：先把"该附哪个进程"**搞清楚**，后面注入才稳。

---

6) 前端把"运行入口"铺好（按钮、下拉、Spawn、自定义）

- **会话→运行**：右上角"脚本/运行/Spawn"，点"运行"会 POST `/api/run_frida`，把当前会话上下文 + 包名 + 是否 Spawn 传给后端。
- **模板专区**：在"Memory / Frida Templates"里内置 `ssl_pinning_bypass / sharedprefs_dump / dump_class_string_fields / search_jwt_in_static_strings`，点击直接走 `/api/run_frida`。
- **自定义脚本**：独立 Tab 粘贴 JS，勾选 **Spawn**，走 `/api/run_frida_custom`。页面还提供"一键填充示例脚本"。

小白总结版：**按钮都接上**了：会话运行 / 模板运行 / 自定义注入。

---

7) 注入链路的双实现准备（环境兼容）

- **优先 Frida Python API**：`import frida` 成功就 `device.attach/spawn + session.create_script + script.on('message', ...)`。
- **回退 CLI**：否则就起 `subprocess.Popen(['frida', ...])` 读取 stdout/stderr 转发给前端，保证"无论环境如何都能跑"。（代码里已做"可选导入 + fallback"设计，并在 Console 显示 `[frida_started] pid=... script=...` 一类状态。）

小白总结版：不管有没有装 Python 版 Frida，都有路可走。

---

8) 安全与运维层面的先手

- **ADMIN_TOKEN** 保护敏感 API（注入/清空/导出等），前端会询问 Token 并加到 `X-ADMIN-TOKEN` 头。
- **统一日志**：`server_debug.log` 记录后端关键事件，方便你排障。
- **README 与一键使用**：给出"启动 mitm / 启动后端 / 安装证书 / 打开前端 / 一键 Hook"的使用路径，确保演示时走不歪。

小白总结版：把安全与"可演示、可排障"也先安排上了。

结论（给导师的一句话）

> 在实现 Frida 注入前，我已经把 **数据采集（mitm）→ 后端存取/广播（Flask+SocketIO）→ 前端可视化与 Console → 模板库/渲染 → 进程探测与 PID 选择 → 注入运行入口（模板/自定义/Spawn）→ 安全与日志** 这一整条链路搭好。**Frida 注入**只是顺着这条链路接入，因此一接通就能**一键运行 + 实时回显**