系统前端框架及集成方案（微前端：Wujie，双栈并存）

1. 背景与目标

1.1 背景

- 现有 `front-system`：基于 Vue3 + Vite + Element Plus，已完成登录、首页布局、菜单框架、权限管理配置等平台能力。
- 计划新增 `front-process`：基于 Vue3 + Vite + Ant Design Vue，承载"全过程平台"的业务模块开发。
- 诉求：业务模块页面能够被集成到 `front-system` 的菜单体系中，在同一系统壳内统一展示与导航。
- 约束：不要求子应用独立部署到不同域名（允许同域名不同路径）。

1.2 总体目标

- 双栈并存：`front-system` 使用 Element Plus；`front-process` 使用 Ant Design Vue。
- 平台壳统一：登录、鉴权、菜单、导航、用户态由 `front-system` 统一提供。
- 业务独立迭代：`front-process` 可独立开发、独立构建、独立发布（至少独立产物与版本），但最终可同域名部署。
- 集成稳定：支持深链接/刷新不 404；支持主/子应用通信；支持权限控制与菜单动态加载。

1.3 非目标（明确不做）

- 不在同一个页面/同一个组件树内混用 Element Plus 与 Ant Design Vue 的核心表单/表格/弹窗（避免样式与交互冲突）。
- 不追求主/子应用"运行时共享同一个 Vue 实例"这一类高复杂度共享（除非后续明确要求并引入更强的共享机制）。

2. 总体架构

2.1 角色划分

- Host（主应用）：`front-system`
  - 职责：登录、token 生命周期、用户信息、权限与菜单、系统配置、统一布局（header/sidebar/tabs）、全局路由守卫。
  - 职责：提供微前端容器，负责挂载/卸载子应用。
- SubApp（子应用）：`front-process`
  - 职责：全过程业务模块页面与路由、业务页面内的权限展示（按钮/菜单项可见性）以及业务接口调用。
  - 约束：不自行实现登录与 token 刷新；不与 Host 的存储策略强耦合（通过 Host SDK 读取）。

2.2 路由策略（同域名）

- 约定统一路由前缀：
  - Host 自己页面：`/login`、`/system/**`、`/`（首页等）
  - SubApp 页面前缀：`/process/**`
- Host 在路由命中 `/process/**` 时：
  - 渲染“微前端容器页”（仅保留壳：菜单/页签/面包屑等）
  - 将当前路径转发给子应用（由子应用 router 接管内部视图）

2.3 菜单与权限模型

- 菜单数据由 Host 统一维护（系统权限管理模块）。
- Host 菜单支持两类节点：
  - Host 页面：`type=host`，如 `/system/role`
  - 子应用页面：`type=micro`，如 `/process/order/list`，并包含 `app=process`
- 权限判断：
  - Host：用于决定“菜单是否出现/路由是否允许进入”。
  - SubApp：用于按钮/页面内部功能点可见性（通过 Host SDK 获取权限集合）。

3. 技术选型：Wujie（已选定）

3.1 选型结论

- 本方案明确选用：Wujie（Vue3 + Vite 场景）。
- 目标：在 `front-system`（Element Plus）不大改动的前提下，接入 `front-process`（Ant Design Vue）并进行样式隔离与生命周期治理。

3.2 选型理由

- 接入路径更直接：同域名、路由前缀、容器挂载属于 Wujie 的典型场景。
- 隔离能力更容易落地：可选 Shadow DOM/样式隔离能力，更适配现有 Host 中 Tailwind 与 Element Plus 全局样式覆盖的现状。

3.3 统一要求

- 明确“接入协议”（路由、菜单、鉴权、通信）并版本化。
- 子应用必须在容器内可重复 mount/unmount，不泄露全局事件、定时器、全局样式。

4. 仓库与工程结构建议

4.1 当前仓库结构（建议演进）

建议把前端拆为两个目录（可在同仓库内）：

```
WholeProcessPlatform/
  frontend-system/   # Host：Element Plus
  frontend-process/  # SubApp：Ant Design Vue
  backend/           # Spring Boot
```

如果短期不希望移动目录，也可以先在现有 `frontend/` 基础上复制一份 `frontend-process/`，逐步迁移到标准结构。

4.2 子应用产物与路径约定（同域名）

- Host 发布到：`/`（例如 `https://example.com/`）
- SubApp 发布到：`/process/`（例如 `https://example.com/process/`）
- 子应用构建时必须配置：
  - `base: '/process/'`
  - 资源路径、路由刷新策略与后端 rewrite 一致

5. 接入协议（核心：可迭代）

5.1 子应用 Manifest（推荐）

Host 通过一个标准 manifest 获取子应用信息，避免硬编码：
`GET /process/manifest.json`（同域名静态文件或接口均可）

示例：

```
{
  "name": "process",
  "version": "1.0.0",
  "entry": "/process/",
  "routeBase": "/process",
  "routes": [
    { "path": "/process/order/list", "title": "订单列表", "perm": "process:order:list" }
  ]
}
```

Host 使用方式：

- 菜单配置模块把 `perm/path/title/app` 写入系统菜单表。
- 或者 Host 启动时拉取 manifest，生成“可选菜单项”供管理员配置。

5.2 Host SDK（推荐：共享通信与鉴权）

提供一个统一 SDK（同仓库内共享文件或独立包），子应用只能通过 SDK 获取宿主能力。

接口建议（最小闭环）：

- `getToken(): string | null`
- `getUser(): { id: string; username: string; roles: string[]; perms: string[] } | null`
- `hasPerm(code: string): boolean`
- `onAuthChanged(cb): () => void`
- `navigate(path: string): void`（由 Host 控制主路由）
- `getAppConfig(): Record<string, any>`

通信方式建议：

- 同窗口同域名：优先使用自定义事件（`window.dispatchEvent(new CustomEvent(...))`）或轻量事件总线。
- 避免子应用直接读写 Host 的 pinia/vuex store（强耦合）。

5.3 鉴权约定

- token 统一由 Host 管理：登录、刷新、退出、失效重登。
- 子应用请求后端 API 的 token 获取：
  - `Authorization: Bearer <token>` 由子应用拦截器从 Host SDK 注入。
- 子应用不得自行实现登录页面与 token 刷新逻辑（避免双重刷新与状态打架）。

6. 样式隔离与 UI 共存策略（关键风险治理）

6.1 原则

- Host（Element Plus）与 SubApp（AntD）在同一 DOM 上下文时，必须最小化全局样式污染。
- 禁止在子应用里全局覆盖宿主样式（也不建议在宿主覆盖子应用）。

6.2 治理手段

- 选择 wujie 时：
  - 优先开启样式隔离能力（根据实际接入方式选择 Shadow DOM 或样式沙箱）。
- Tailwind 风险：
  - Host 当前使用 Tailwind 全局注入时，会影响子应用的基础样式（尤其 preflight/base）。
  - 建议后续收敛：对子应用范围做隔离，或在 Host 中通过约定限制 tailwind base 的影响范围。
- 弹层与 z-index：
  - AntD 与 Element Plus 都有 modal/message/notification。
  - 约定：子应用弹层仅覆盖子应用容器；宿主弹层覆盖全局；避免跨应用调用对方 UI 弹层。

7. 本地开发联调模式（Dev）

7.1 端口约定（示例）

- backend：`8093`
- Host（front-system）：`5173`
- SubApp（front-process）：`5174`

7.2 代理与同域策略

由于“不要求独立域名”，本地开发仍建议通过代理实现“看起来同域”：

- Host dev server 代理：
  - `/api` → `http://localhost:8093`
  - `/process/` → `http://localhost:5174/`（子应用 dev server）

这样 Host 在开发态加载子应用时仍以 `/process/` 作为 entry，最大程度贴近生产路径。

7.3 Demo 联调（本仓库）

- Host（system）菜单项：`/process/antd-demo`
- 子应用（process）入口：`/process/`（由 Host 通过 Wujie 挂载）
- 本地启动顺序：
  - 启动 `frontend-process`：`pnpm dev`（端口 5174）
  - 启动 `front-system`（当前仓库的 `frontend`）：`pnpm dev`（端口 5173）
  - 在 Host 菜单中进入“AntD Demo”页面进行验证

8. 生产部署方案（同域名）

8.1 Nginx（推荐）

- Host 静态文件：`/` 指向 host dist
- SubApp 静态文件：`/process/` 指向 subapp dist
- 后端接口：`/api/` 反向代理到 Spring Boot
- 路由刷新：对 `/` 和 `/process/` 分别做 fallback 到对应的 `index.html`

8.2 Spring Boot 静态资源（可行但需约束）

如果用 Spring Boot 统一承载静态资源：

- Host 构建产物放入 `backend/src/main/resources/static/`
- SubApp 构建产物放入 `backend/src/main/resources/static/process/`
- 需确保：
  - Host 与 SubApp 的资源路径不冲突
  - Spring MVC 对前端路由做 rewrite（尤其 `/process/**` 刷新）

9. 迭代计划（按里程碑）

M1：微前端骨架（1 条链路跑通）

- Host 增加微前端容器页 `/process/**`
- SubApp 实现最小页面（例如 `/process/health`）
- Host 菜单可配置一个子应用菜单项并打开该页面
- Host SDK：实现 `getToken/getUser/hasPerm/navigate`

M2：权限与菜单联动

- 菜单管理支持 `type=micro` 与 `app=process`
- 子应用按 `perm` 做按钮级展示
- 深链接与刷新校验通过（直接打开 `/process/**`）

M3：工程治理

- 样式隔离策略固化（shadow dom / css sandbox）
- 错误边界：子应用加载失败时 Host 给出可恢复提示
- 性能优化：按路由懒加载子应用、预加载关键模块

10. 风险清单与应对

10.1 路由刷新 404

- 风险：/process/** 刷新时服务端不 rewrite 到 index.html。
- 应对：Nginx 或 Spring Boot rewrite 规则必须补齐，并对 Host/SubApp 各自生效。

10.2 鉴权状态不一致

- 风险：子应用自行刷新 token 或自行维护登录态。
- 应对：只允许 Host 管理 token；子应用通过 SDK 读取并订阅变更。

10.3 样式污染与弹层冲突

- 风险：Tailwind base、全局 reset、z-index。
- 应对：隔离策略 + 约定弹层边界 + 视觉回归检查。

10.4 版本错配

- 风险：Host 菜单/协议升级，子应用未同步。
- 应对：manifest 版本化；Host 对协议做兼容或强制版本一致发布。

11. 最小验收标准（可作为迭代验收）

- 登录成功后，Host 菜单可进入 `/process/**` 页面并正常展示子应用内容。

- 浏览器刷新 `/process/**` 不 404。
- 子应用通过 Host SDK 获取 token 调用后端 API 成功。
- Host 退出登录后，子应用自动感知并回到登录页（由 Host 控制导航）。
- 子应用异常（加载失败/运行时错误）不会导致 Host 白屏，Host 能提示并支持重试。