L1 — 系统架构与安全模型层

文档版本: 1.0 **适用项目**: sgClaw (业数融合一平台 AI Agent 底座) **编制日期**: 2026-03-03

1. 全局架构拓扑

1.1 完整架构图

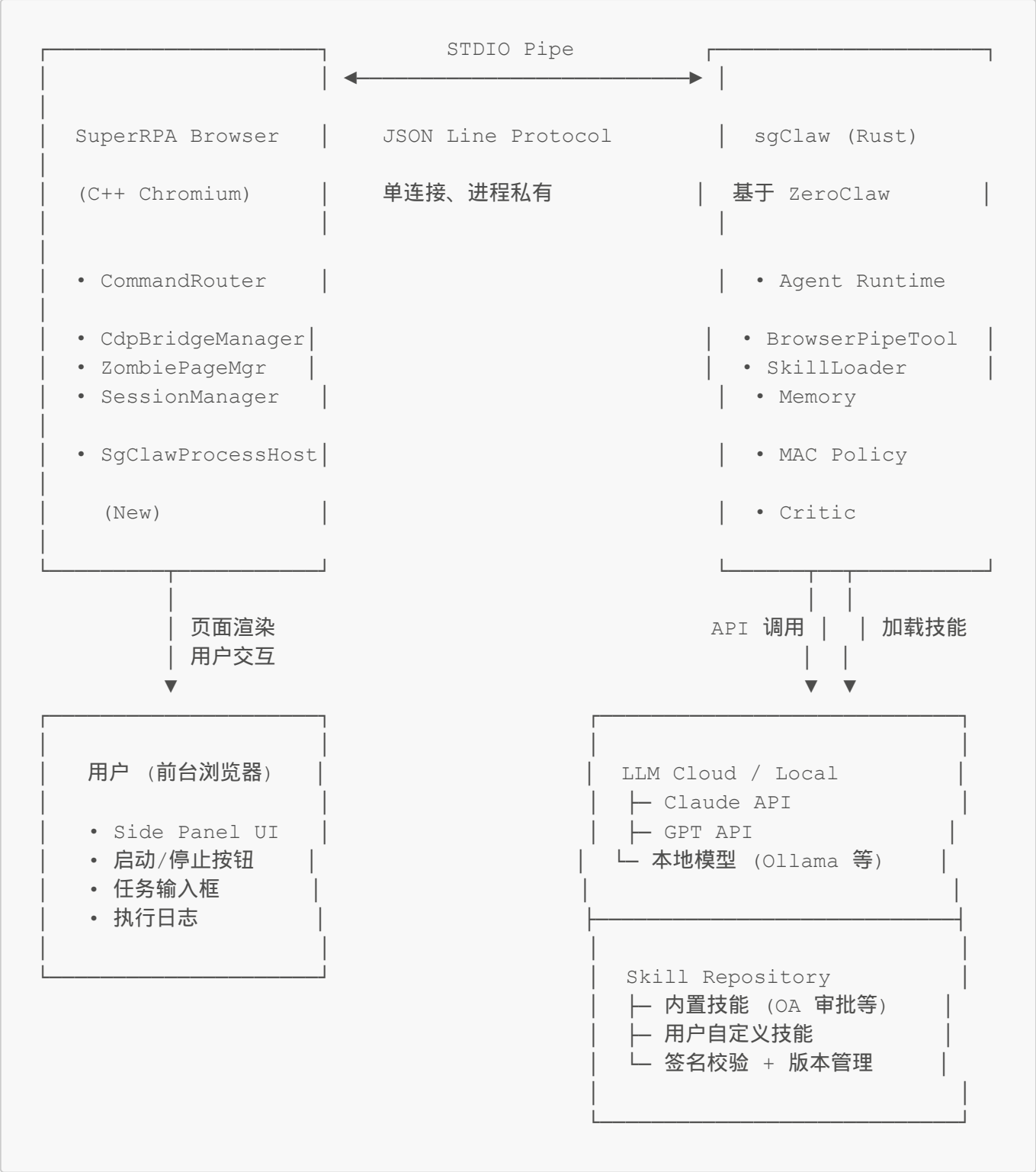
以下是 sgClaw 系统的完整进程与组件拓扑。图中标注了新增组件(New)与已有组件(Existing)， 以便评估改造范围。

```
superrpa-chromium.sh (wrapper, 启动入口)
  │
  ├─ watchdog (background) ──────────────────────────── [Existing, 不变]
  │    └─ node local_service                            [Existing, 不变]
  │         ├─ 本地 HTTP API (端口 9527)
  │         └─ 文件上传 / 下载辅助
  │
  └─ chrome (main process, foreground) ──────────────── [Existing, 不变]
       │
       ├─ SgClawProcessHost ──────────────────────────── [New, Singleton in
Browser Process]
       │    │
       │    ├── STDIO Pipe (stdin/stdout) ──────────→ sgclaw (Rust child
process)
       │    │                                              │
       │    │                                              ├─ Agent Runtime
──────── ZeroClaw ReAct Loop
       │    │                                              │    └─ think → act
→ observe → repeat
       │    │                                              │
       │    │                                              ├─ LLM Provider
──────── Claude / GPT / 本地模型
       │    │                                              │    ├─ streaming
支持
       │    │                                              │    └─ token 用量统
计
       │    │                                              │
       │    │                                              ├─ BrowserPipeTool
─── 自定义 Tool trait 实现
       │    │                                              │    └─ 将 Agent
action 序列化为 pipe command
       │    │                                              │
       │    │                                              ├─ SkillLoader
──────── 加载 JS 业务技能脚本
       │    │                                              │    ├─ 技能签名校验
       │    │                                              │    └─ 沙箱执行环境
       │    │                                              │
       │    │                                              ├─ MAC Policy
```

```
──────── 域 / Action 白名单
        │     │                                                    │     └─ rules.json
配合浏览器侧双重校验
        │     │                                                    │
        │     │                                                    ├─ Critic
──────────── 输出质量评估
        │     │                                                    │     └─ Circuit
Breaker (熔断器)
        │     │                                                    │
        │     │                                                    ├─ Memory
──────────── SQLite + Vector 向量存储
        │     │                                                    │     ├─ 短期对话记忆
        │     │                                                    │     └─ 长期任务知识
库
        │     │                                                    │
        │     │                                                    └─ MCP Client
──────── rmcp (Rust MCP SDK)
        │     │                                                          └─ 连接外部 MCP
Server 获取工具
        │     │
        │     ├─ PipeListener (async read loop)
        │     │     └─ 解析 JSON Line，分发到 MAC 校验
        │     │
        │     ├─ MAC Whitelist Check ──────────────────── [New, ~100 lines
C++]
        │     │     ├─ 校验 action 是否在 pipe 允许列表
        │     │     └─ 校验 expected_domain 是否匹配当前页
        │     │
        │     └─→ CommandRouter ──────────────────────── [Existing, 40+
actions, 不变]
        │           ├─ CdpCommandExecutor → 页面操作
        │           ├─ ZombiePageManager → 后台会话池
        │           └─ SessionManager → 会话状态机
        │
        ├─ CdpBridgeManager ──────────────────────────── [Existing, 页内 JS
SDK, 不变]
        │     └─ 页面 JS → CDP binding → CommandRouter
        │
        ├─ ZombiePageManager ─────────────────────────── [Existing, 后台会话池,
不变]
        │     └─ 最多 5 个 zombie page，自动回收
        │
        ├─ SessionManager ────────────────────────────── [Existing, 会话状态机,
不变]
        │     └─ login → active → expired → re-login
        │
        └─ RpaGlobalStorage ──────────────────────────── [Existing, 跨页存储,
不变]
              └─ key-value 存储，持久化到磁盘
```

1.2 简化四组件视图

从系统集成的角度，sgClaw 的架构可简化为四个核心组件的交互：



---

2. 技术选型决策记录

2.1 基底框架选型: ZeroClaw

在确定 sgClaw 的 Agent Runtime 底座时，团队对 Rust 生态中主流 Agent 框架进行了系统评估。 评估维度包括：社区活跃度、内存占用、可嵌入性（是否支持替换浏览器控制层）、以及与我们 STDIO Pipe 架构的兼容性。

Framework	Language	Stars	Runtime Memory	Embeddability	Browser Layer Replaceable	备注
ZeroClaw	Rust	17K	~5 MB	High (trait-driven)	**Best**	活跃社区，trait 抽象完善
Rig	Rust	6.2K	Minimal	Best (cargo add)	Full custom	轻量但 Agent loop 需自建
Moltis	Rust	-	~40 MB	Medium	Via MCP only	内存偏高，嵌入需改造
OpenFang	Rust	New	~40 MB	Low (API only)	Difficult	API Server 架构，不适合嵌入
MicroClaw	Rust	152	-	Low (fork needed)	Via MCP only	社区不活跃，需 fork 维护

决策：ZeroClaw

核心理由：

1. **trait-driven 架构**：ZeroClaw 将 Tool、Provider、Memory、Security 全部定义为 trait， 允许我们用自定义的 `BrowserPipeTool` 替换默认的浏览器控制层（如 Playwright/Puppeteer）， 同时复用其 Agent Runtime、Provider 抽象、Memory 系统和安全模块。
2. **内存优势**：~5 MB 的运行时内存占用，在 8 GB 总内存预算内几乎可以忽略不计。 相比 Moltis/OpenFang 的 ~40 MB，差异显著。
3. **ReAct Loop 成熟**：ZeroClaw 内置 think → act → observe 循环，支持 streaming、 multi-turn、tool-use，无需从零构建。
4. **MCP 生态兼容**：内置 rmcp client，可在需要时连接外部 MCP Server 扩展工具集。

2.2 通信协议选型: STDIO Pipe

sgClaw (Rust) 与 SuperRPA Browser (C++) 之间的 IPC 通道是整个架构的关键路径。 团队评估了四种 IPC 机制：

方式	Linux 支持	Windows 支持	安全性	多连接支持	延迟
STDIO Pipe	fd inherit	HANDLE inherit	**最高** (进程私有，无法外部连接)	否	~0.1 ms
Unix Domain Socket	/tmp/sock	不支持	高 (文件权限控制)	是	~0.2 ms
Named Pipe	不支持	\\.\pipe\	高 (DACL)	是	~0.2 ms
TCP localhost	支持	支持	**低** (任意进程可连)	是	~0.5 ms

决策：STDIO Pipe

核心理由：

1. **安全性最高**：STDIO Pipe 是进程私有的文件描述符（Linux）或句柄（Windows），只有父子进程间可以访问。本机其他进程无法连接、监听或注入命令。这从物理层面杜绝了 Pipe Hijack 攻击。
2. **跨平台统一**：Chromium 的 `base::LaunchProcess` 已经抽象了跨平台的管道创建，Linux 使用 `pipe()` + `fork()`/`exec()`，Windows 使用 `CreatePipe()` + `CreateProcess()`。我们不需要编写任何平台特定代码。
3. **单连接足够**：sgClaw 是 Browser 的唯一 Rust 子进程，一条 STDIO Pipe 即可满足全部双向通信需求。不需要多客户端并发连接的能力。
4. **延迟最低**：内核态 pipe buffer（Linux 默认 64 KB）的读写延迟约 0.1 ms，远低于 socket 方案。

2.3 Why Rust

为什么 sgClaw 选择 Rust 而非 C++/Python/Node.js：

1. **内存安全无 GC**：在 8 GB 总内存限制下，GC 语言的内存开销和暂停不可接受。Rust 的零成本抽象确保 ~5 MB 运行时内存，无 GC 停顿。
2. **跨平台编译**：目标平台包括 Linux 银河麒麟 V10 SP1 (x86_64) 和 Windows 10/11。Rust 的 cross-compilation 工具链 (`cross`, `cargo-zigbuild`) 可直接生成两个平台的二进制。
3. **极小二进制体积**：release 构建约 8.8 MB（含 LLM provider、memory、MCP client 全部功能），Windows 约 9 MB。无需额外运行时依赖。
4. **冷启动极快**：< 10 ms 内完成进程初始化和 pipe handshake，用户点击"启动"后几乎无感。
5. **ZeroClaw 生态**：ZeroClaw 本身及其核心依赖（rmcp、tokenizers 等）均为 Rust 实现，选择 Rust 可直接复用，无需 FFI 桥接。

---

3. 进程模型与生命周期

3.1 进程层次结构

sgClaw 嵌入 SuperRPA 浏览器的既有进程树中，作为 Chrome 主进程的子进程存在。

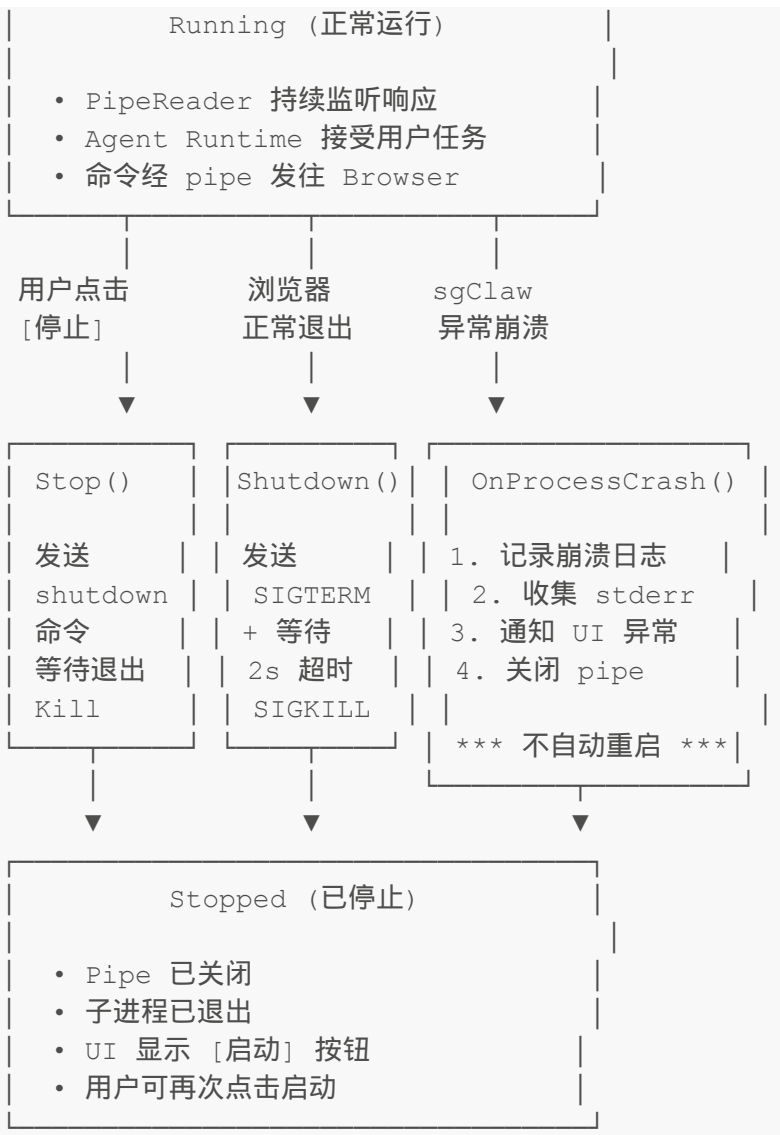
```
superrpa-chromium.sh (wrapper, PID 1001)
  │
  ├─ watchdog (PID 1002, background, 守护进程)
  │    └─ node local_service (PID 1003, HTTP API)
  │         ├─ 监听 127.0.0.1:9527
  │         ├─ 文件上传/下载
  │         └─ 与浏览器通过 HTTP 通信
  │
  └─ chrome (PID 1004, foreground, 用户交互主进程)
       │
       │  当用户在 Side Panel 点击 [启动] 按钮时：
```

```
        │  SgClawProcessHost::Start() 被调用
        │
        └─ sgclaw (PID 1005, child of chrome)
             │
             ├─ stdin  ← chrome stdout (接收命令响应/事件)
             ├─ stdout → chrome stdin  (发送命令请求)
             │
             ├─ Agent Runtime (ReAct loop)
             ├─ LLM Provider (网络出口)
             └─ Memory (SQLite 文件 I/O)
```

关键特征：

- sgclaw 是 chrome 的**直接子进程**，继承 chrome 的 UID/GID 权限
- STDIO Pipe 的文件描述符在 `fork()`/`exec()` 时自动继承，无需额外传递
- 当 chrome 进程退出时，sgclaw 会收到 SIGPIPE/EOF，自行退出
- watchdog 和 local_service 与 sgclaw **无直接通信**，完全独立

3.2 生命周期状态机



关键决策：崩溃不自动重启

当 sgclaw 进程崩溃时（如 panic、OOM、非零退出码），系统**不会**自动重启。 用户必须显式点击 [启动] 按钮才能再次启动。理由：

- 防止恶意 prompt injection 导致的无限崩溃重启循环
- 防止异常状态下的资源耗尽（内存泄漏累积、日志磁盘写满）
- 给用户明确的故障感知，而非静默恢复后行为异常

3.3 内存预算 (8 GB 约束)

目标部署环境为 8 GB 内存的银河麒麟工作站。以下是各组件的内存分配：

组件	内存占用	备注
OS + Desktop (银河麒麟 V10)	~2.0 GB	固定开销
Browser Process (主进程)	~0.2 GB	已有
Foreground Tabs (1-3 tabs)	~0.3-0.9 GB	用户页面
Side Panel (Agent UI)	~0.05 GB	Vue SPA

Zombie Page Pool (max 5)	~0.15-0.25 GB	后台会话
local_service (Node.js)	~0.1 GB	已有
sgClaw (Rust binary)	~0.005 GB	5 MB
LLM context cache (对话历史)	~0.05 GB	内存缓存
SQLite + vector memory	~0.02 GB	磁盘为主
Total	**~3.0-3.6 GB**	
Headroom	**~4.4-5.0 GB**	>50%

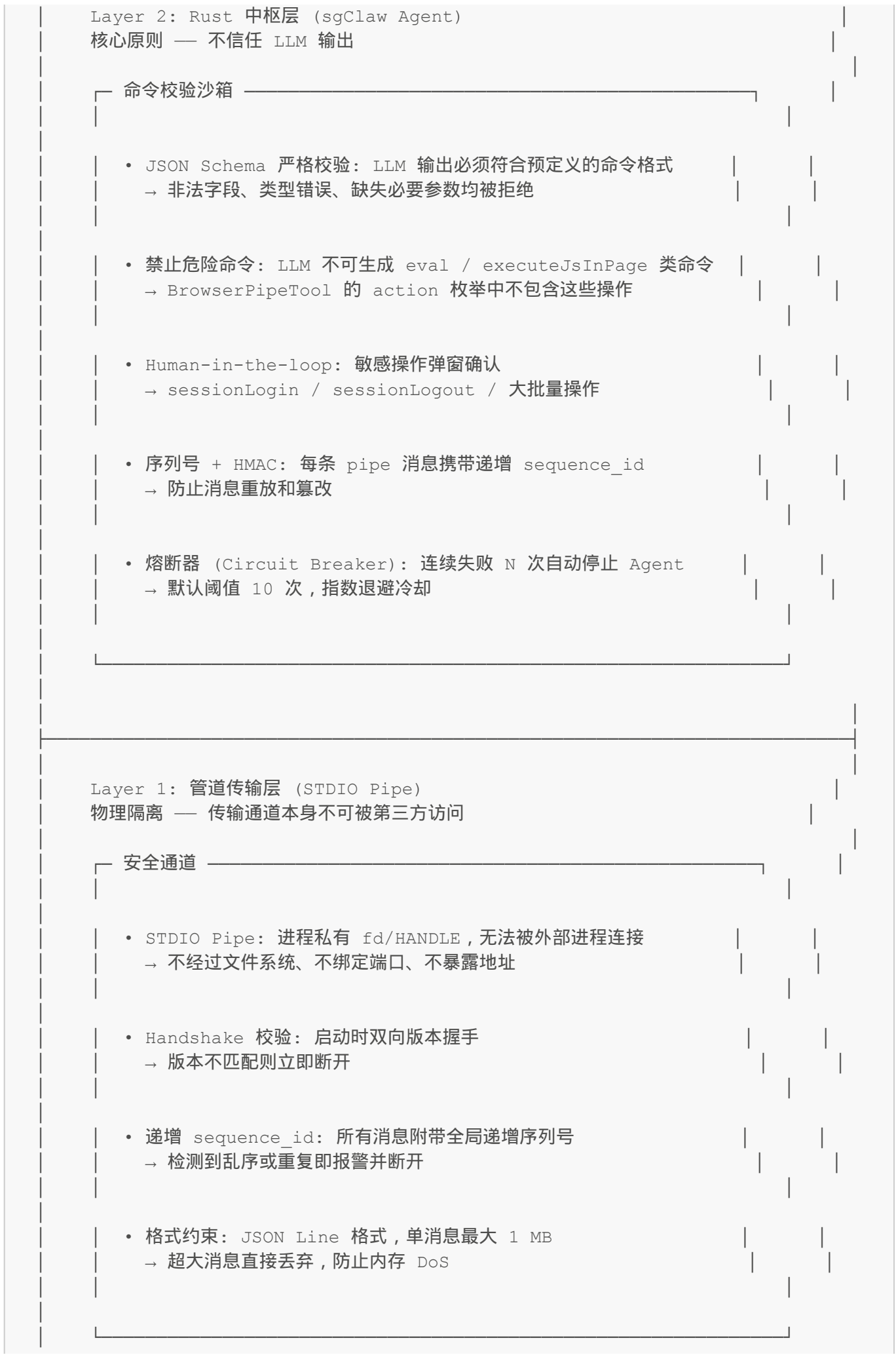
sgClaw 的 ~5 MB 内存占用仅为总预算的 0.06%，对系统几乎零负担。 即使在极端场景下（3 个前台标签 + 5 个 zombie page + LLM 长对话）， 总内存使用也不超过 4 GB，保留超过 50% 的余量供操作系统和突发需求使用。

---

4. 安全架构

4.1 三层纵深防御模型

sgClaw 的安全架构采用纵深防御（Defense in Depth）策略，从内到外设置三道独立的安全层。 任何单一层被突破时，上层仍能有效阻止攻击。


Layer 2: Rust 中枢层 (sgClaw Agent)
核心原则 —— 不信任 LLM 输出
命令校验沙箱
• JSON Schema 严格校验：LLM 输出必须符合预定义的命令格式
→ 非法字段、类型错误、缺失必要参数均被拒绝
• 禁止危险命令：LLM 不可生成 eval / executeJsInPage 类命令
→ BrowserPipeTool 的 action 枚举中不包含这些操作
• Human-in-the-loop：敏感操作弹窗确认
→ sessionLogin / sessionLogout / 大批量操作
• 序列号 + HMAC：每条 pipe 消息携带递增 sequence_id
→ 防止消息重放和篡改
• 熔断器 (Circuit Breaker)：连续失败 N 次自动停止 Agent
→ 默认阈值 10 次，指数退避冷却
Layer 1: 管道传输层 (STDIO Pipe)
物理隔离 —— 传输通道本身不可被第三方访问
安全通道
• STDIO Pipe：进程私有 fd/HANDLE，无法被外部进程连接
→ 不经过文件系统、不绑定端口、不暴露地址
• Handshake 校验：启动时双向版本握手
→ 版本不匹配则立即断开
• 递增 sequence_id：所有消息附带全局递增序列号
→ 检测到乱序或重复即报警并断开
• 格式约束：JSON Line 格式，单消息最大 1 MB
→ 超大消息直接丢弃，防止内存 DoS

```
│                                                                                    │
└────────────────────────────────────────────────────────────────────────────────────┘
```

4.2 各层防御详解

Layer 1 — 管道传输层

本层的设计目标是确保 sgClaw 与 Browser 之间的通信通道从物理层面不可被第三方窃听或注入。

防御对象：

- 本机恶意进程尝试连接或嗅探 IPC 通道
- 中间人攻击（消息篡改、注入伪造命令）
- 消息重放（录制合法命令序列后重新发送）

实现机制：

- STDIO Pipe 使用操作系统内核的匿名管道，文件描述符仅在父子进程间继承，`/proc/[pid]/fd/` 对其他用户不可见（取决于 procfs 挂载选项，银河麒麟默认安全）
- 启动时的 handshake 消息包含协议版本号，防止二进制版本不匹配导致的解析错误
- 每条消息的 `sequence_id` 全局递增，接收方校验连续性，检测到跳号或重复立即断开 pipe

Layer 2 — Rust 中枢层

本层的核心假设是 **LLM 的输出不可信**。无论是 prompt injection 还是 hallucination， LLM 可能生成任何格式、任何内容的输出。sgClaw 必须在将命令发往 Browser 之前进行严格校验。

防御对象：

- Prompt injection：恶意网页内容被 LLM 读取后，诱导其生成危险操作
- Hallucination：LLM 生成不存在的 action 或格式错误的参数
- 无限循环：LLM 陷入 retry 死循环，持续消耗资源

实现机制：

- `BrowserPipeTool` 定义了一个封闭的 action 枚举（enum），LLM 只能选择预定义的安全操作
- 每个 action 都有对应的 JSON Schema，参数类型、范围、必填项均有严格约束
- Critic 模块在 Agent 每步 observe 后评估输出质量，异常时触发终止
- Circuit Breaker 在连续 10 次操作失败后自动停止 Agent loop，需用户手动恢复
- 敏感操作（涉及登录、批量删除等）通过 Side Panel UI 弹窗请求用户确认

Layer 3 — 浏览器内核层

本层是安全架构的最后防线。即使 sgClaw 的 Rust 层被完全绕过（理论上几乎不可能）， Browser 的 C++ 代码仍会独立执行 MAC 检查。

防御对象：

- 已突破 Layer 2 的命令（假设 sgClaw 被完全控制）
- 未授权域上的操作（sgClaw 试图操作不在白名单中的网站）
- 凭证窃取（任何试图通过 pipe 读取存储密码的操作）

实现机制：

- `rules.json` 白名单由管理员配置，列出允许 Agent 操作的域名（如 `oa.example.com`）
- 每条 pipe 命令到达 Browser 后，先校验 `expected_domain` 是否与当前页面的实际域名一致
- `credential_store`（Linux Keyring / Windows Credential Manager）的 API 在 pipe handler 中 没有任何暴露点——登录凭证只能由 SessionManager 在 Browser 内部使用
- 速率限制器独立于 sgClaw 的 Circuit Breaker，即使后者被禁用，Browser 侧的限速仍然生效

---

5. 威胁模型与对抗策略

5.1 威胁分类总表

#	威胁名称	描述	影响	防御层	对抗策略
T1	Pipe Hijack	本机攻击者劫持 STDIO Pipe	完全控制浏览器操作	Layer 1	STDIO 进程私有；fd 不经过文件系统；procfs 受限
T2	LLM Prompt Injection	恶意页面内容注入 LLM prompt	执行未授权操作	Layer 2	JSON Schema 校验；Action 枚举白名单；禁止 eval
T3	Skill Poisoning	注入恶意 Skill 脚本	持久化后门	Layer 2 + 3	Skill 签名校验；沙箱执行；MAC 域名限制
T4	Credential Leakage	通过 pipe 读取浏览器存储的密码	账户泄露	Layer 3	credential_store API 不暴露给 pipe；凭证仅内部使用
T5	Replay Attack	录制合法命令后重放	重复执行敏感操作	Layer 2	sequence_id 递增校验 + HMAC 签名
T6	Retry Storm	LLM 无限重试导致资源耗尽	CPU/内存/网络耗尽	Layer 2	Circuit Breaker (阈值 10)；指数退避；最大重试限制

5.2 各威胁详细分析

T1: Pipe Hijack（管道劫持）

攻击路径：同一台机器上的恶意进程尝试读取/写入 sgClaw 与 Browser 之间的 STDIO Pipe。

分析：STDIO Pipe 基于操作系统匿名管道实现，没有文件系统路径，没有网络端口。 攻击者需要获取目标进程的 fd（Linux）或 HANDLE（Windows），这在标准安全配置下 需要 root 权限或 `ptrace` 能力。银河麒麟 V10 默认启用 `kernel.yama.ptrace_scope=1`， 非父进程无法 ptrace。

残余风险：root 用户可以访问任何进程的 fd。但在本部署场景中，root 被视为可信实体。

T2: LLM Prompt Injection（提示注入）

攻击路径：用户访问的恶意网页中包含精心构造的文本，当 sgClaw 读取页面内容并发送给 LLM 时， 这些文本被 LLM 误解为指令，导致 Agent 执行非预期操作。

分析：这是 AI Agent 面临的最常见威胁。sgClaw 的防御不依赖于 LLM 本身的抗注入能力 （这是不可靠的），而是在 Rust 层进行硬编码限制：

- LLM 只能输出预定义的 action 枚举值，任何不在枚举中的操作被直接拒绝
- 最危险的 `eval`、`executeJsInPage` 不在枚举中，从根本上不可能被 LLM 触发
- `expected_domain` 字段要求 LLM 明确声明目标域名，Browser 侧校验实际域名是否匹配

T3: Skill Poisoning（技能投毒）

攻击路径：攻击者将恶意 JavaScript Skill 注入到 Skill Repository 中， 当 sgClaw 加载该 Skill 时执行恶意代码。

分析：Skill 文件在加载前经过以下校验链：

1. 签名校验：每个 Skill 文件必须附带由构建系统生成的数字签名
2. 沙箱执行：Skill 代码在受限的 JavaScript 沙箱中运行，无法访问文件系统或网络
3. MAC 限制：即使 Skill 生成了恶意命令，Browser 侧的域白名单仍会阻止未授权操作

T4: Credential Leakage（凭证泄漏）

攻击路径：通过 pipe 发送特制命令，尝试读取 Browser 中存储的登录密码。

分析：`credential_store` 使用 Linux Keyring（银河麒麟）或 Windows Credential Manager 存储加密凭证。其 C++ API 仅在 `SessionManager::DoLogin()` 内部调用， 没有任何 CommandRouter action 会暴露凭证数据。pipe handler 的代码路径中 完全不存在读取凭证的逻辑。

T5: Replay Attack（重放攻击）

攻击路径：攻击者（假设已突破 Layer 1）录制合法的 pipe 命令序列，稍后重放。

分析：每条消息携带递增的 `sequence_id` 和基于共享密钥的 HMAC。 接收方维护已处理的最大 sequence_id，拒绝任何小于或等于该值的消息。 HMAC 密钥在每次 sgclaw 启动时通过 handshake 动态协商，重启后旧命令的 HMAC 无效。

T6: Retry Storm（重试风暴）

攻击路径：LLM 因 hallucination 或 prompt injection 陷入无限循环， 持续生成失败命令并重试。

分析：sgClaw 内置两级防护：

1. Agent Runtime 层：单次任务最大步数限制（默认 50 步），超出自动终止
2. Circuit Breaker：连续 10 次操作失败后，断路器打开，停止所有 pipe 通信， 需要用户手动恢复。冷却期间采用指数退避（1s → 2s → 4s → ... → 30s max）

---

6. 通信架构

6.1 Pipe 协议概览

sgClaw 与 Browser 之间使用 JSON Line 格式通过 STDIO Pipe 通信。 每条消息占一行（以 `\n` 结尾），单消息最大 1 MB。

消息类型一览：

```
sgClaw → Browser:
  ├─ command   （请求执行浏览器操作）
```

```
  └─ ack       (确认收到事件)

Browser → sgClaw:
  ├─ response (命令执行结果)
  ├─ event    (主动推送的页面事件)
  └─ init_ack (handshake 响应)
```

Request 格式（sgClaw → Browser）：

```
{
  "seq": 1,
  "type": "command",
  "action": "click",
  "params": {
    "selector": "#submit-btn",
    "wait_after": 1000
  },
  "security": {
    "expected_domain": "oa.example.com",
    "hmac": "a3f8c2..."
  }
}
```

字段说明：

- `seq`: 递增序列号，全局唯一，用于匹配 response 和防重放
- `type`: 消息类型，`command` 表示操作请求
- `action`: 操作名称，必须在允许列表中
- `params`: 操作参数，每个 action 有独立的 JSON Schema
- `security.expected_domain`: 期望的目标域名，Browser 侧校验
- `security.hmac`: 消息完整性签名

Response 格式（Browser → sgClaw）：

```
{
  "seq": 1,
  "type": "response",
  "success": true,
  "data": {
    "clicked": true,
    "element_text": "提交"
  },
  "aom_snapshot": [
    {"role": "button", "name": "提交", "bounds": [100, 200, 80, 30]}
  ],
  "timing": {
    "queue_ms": 2,
    "exec_ms": 45
```

```
  }
}
```

字段说明：

- `seq`: 与请求对应的序列号
- `success`: 操作是否成功
- `data`: 操作结果数据，格式因 action 而异
- `aom_snapshot`: 操作后的 AOM（Accessibility Object Model）快照， 供 Agent 的 observe 阶段使用，理解页面当前状态
- `timing`: 性能计时信息

Event 格式（Browser → sgClaw，主动推送）：

```
{
  "type": "event",
  "event": "page_navigated",
  "data": {
    "url": "https://oa.example.com/approval/list",
    "title": "审批列表 - OA系统",
    "domain": "oa.example.com"
  },
  "timestamp": 1709452800000
}
```

支持的事件类型：

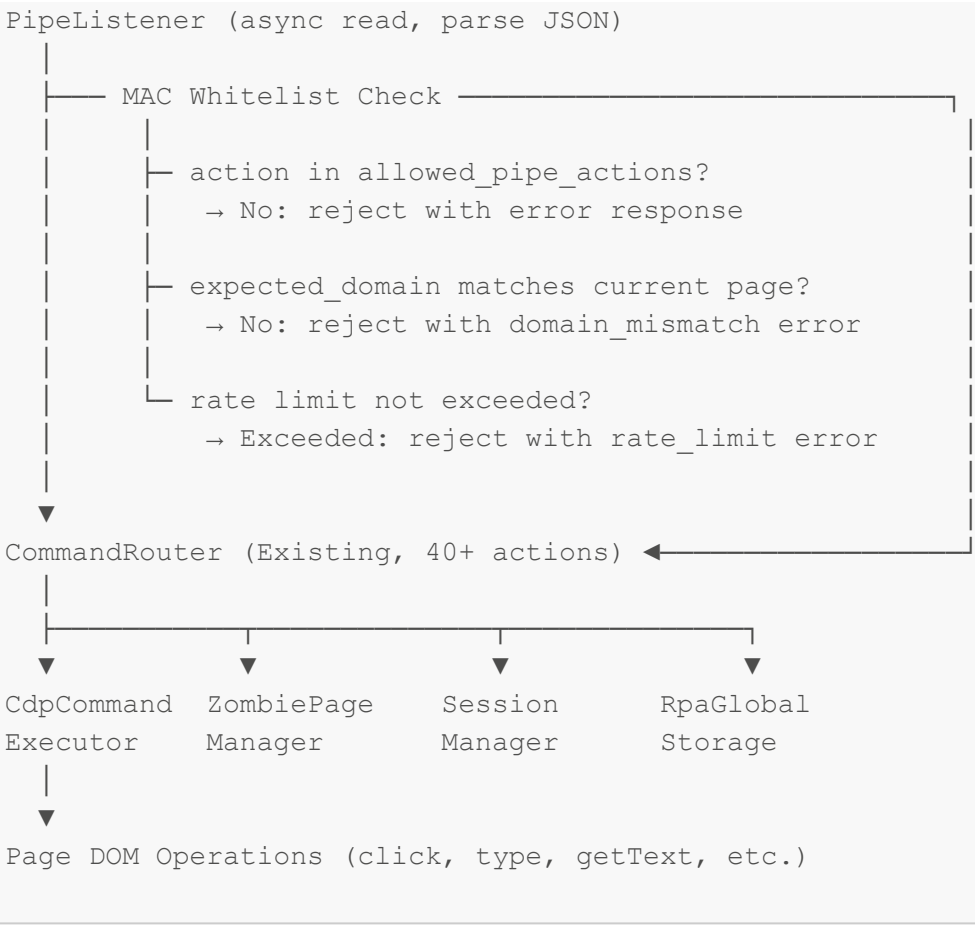
- `page_navigated`: 页面导航完成
- `page_loaded`: 页面 DOMContentLoaded
- `dialog_appeared`: 浏览器对话框弹出（alert/confirm/prompt）
- `session_expired`: SessionManager 检测到会话过期
- `zombie_ready`: Zombie page 初始化完成

6.2 与 CommandRouter 的对接

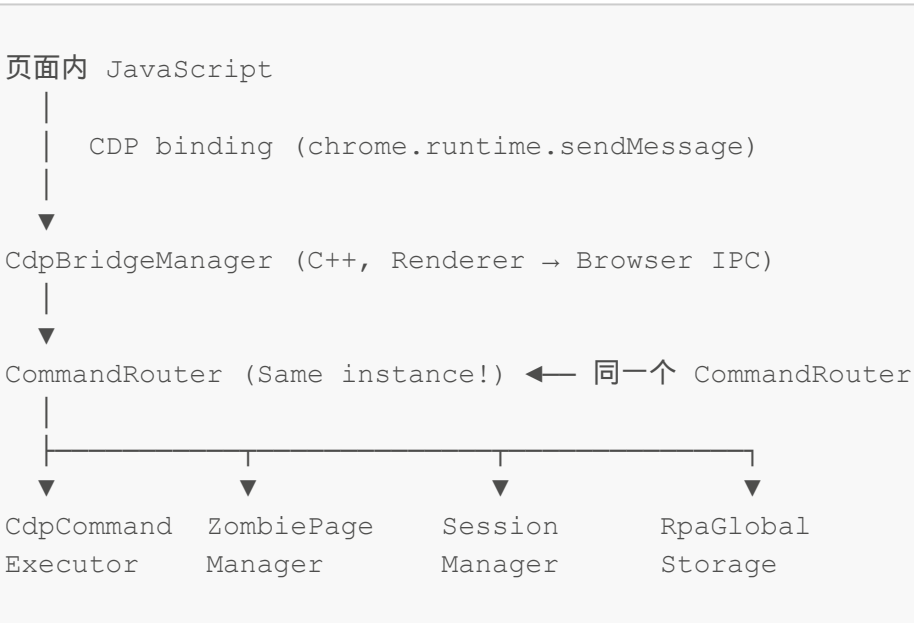
sgClaw 的命令最终汇入 SuperRPA 已有的 CommandRouter，这是一个统一的命令分发器， 支持 40+ 种浏览器操作。sgClaw 作为新增的命令来源，与已有的 JS SDK 并行存在。

sgClaw Pipe 路径（新增）：

```
sgClaw (Rust)
  │
  │  JSON Line over STDIO Pipe
  │
  ▼
SgClawProcessHost (C++, Browser Process)
  │
  ▼
```



已有 JS SDK 路径（不变）：



两条路径汇聚于同一个 CommandRouter 实例。sgClaw 只是新增了一个命令来源， 不改变 CommandRouter 的任何现有逻辑。区别仅在于：

- JS SDK 路径：无 MAC whitelist check（页面 JS 已通过 CSP 和 rules.json 限制）
- Pipe 路径：增加 MAC whitelist check（因为 sgClaw 被视为外部进程，需额外校验）

6.3 Pipe 允许的 Action 子集

并非所有 CommandRouter 支持的 40+ 个 action 都允许通过 pipe 调用。 出于安全考虑，pipe 来源的命令被限制在以下安全子集中：

允许通过 Pipe 调用（无需确认）：

Action	描述	备注
click	点击元素	需 expected_domain 匹配
type	输入文本	需 expected_domain 匹配
navigate	导航到 URL	URL 必须在域白名单内
getText	获取元素文本	只读操作，安全
getHtml	获取元素 HTML	只读操作，安全
waitForSelector	等待元素出现	只读操作，安全
pageScreenshot	页面截图	返回 base64，用于 Agent observe
storageSet	写 RpaGlobalStorage	key 前缀限制为 sgclaw.*
storageGet	读 RpaGlobalStorage	key 前缀限制为 sgclaw.*
zombieSpawn	创建 zombie page	受池大小限制（max 5）
zombieKill	销毁 zombie page	只能销毁自己创建的 zombie
select	下拉框选择	需 expected_domain 匹配
scrollTo	滚动页面	需 expected_domain 匹配
getAomSnapshot	获取 AOM 快照	只读操作，核心 observe 能力

禁止通过 Pipe 调用（硬编码拒绝）：

Action	描述	禁止原因
eval	执行任意 JS	最高危操作，可绕过一切安全限制
executeJsInPage	在页面执行 JS	等同于 eval，prompt injection 核心攻击面
registerJsFunction	注册 JS 回调	可植入持久化代码
setRequestInterceptor	拦截网络请求	可窃取请求中的 token/cookie
exportCookies	导出所有 cookie	可窃取会话凭证

需要用户确认的操作（Human-in-the-loop）：

Action	描述	确认原因
sessionLogin	执行登录流程	涉及凭证使用，需用户知情
sessionLogout	登出会话	可能中断用户正在进行的工作
clearStorage	清空存储	不可逆操作

7. 跨平台策略

7.1 STDIO Pipe 跨平台实现

STDIO Pipe 的跨平台实现依赖 Chromium 已有的 `base::LaunchProcess` 抽象层， sgClaw 团队无需编写平台特定的管道代码。

Linux 实现路径：

```
Browser Process (C++)                          sgClaw (Rust)
──────────────────────                         ──────────────
1. pipe(fds)   → fds[0](读), fds[1](写)
2. pipe(fds2) → fds2[0](读), fds2[1](写)
3. fork()
4. 子进程:
   dup2(fds[0], STDIN_FILENO)
   dup2(fds2[1], STDOUT_FILENO)
   exec("sgclaw")  ──────────────────→       main()
5. 父进程:                                    std::io::stdin()  → 读取命令
   write(fds[1], ...)  ──────────────→         std::io::stdout() → 写回响应
   read(fds2[0], ...)  ←──────────────         BufReader + BufWriter
```

Chromium 的 `base::LaunchProcess` 封装了上述 `pipe()` + `fork()` + `exec()` 流程， 通过 `LaunchOptions::fds_to_remap` 配置 fd 映射。

Windows 实现路径：

```
Browser Process (C++)                          sgClaw (Rust)
──────────────────────                         ──────────────
1. CreatePipe(&hRead1, &hWrite1, ...)
2. CreatePipe(&hRead2, &hWrite2, ...)
3. SetHandleInformation(hRead1, INHERIT)
4. SetHandleInformation(hWrite2, INHERIT)
5. CreateProcess("sgclaw.exe",
     STARTUPINFO {
       hStdInput  = hRead1,                  main()
       hStdOutput = hWrite2                    std::io::stdin()  → ReadFile
     })                                        std::io::stdout() →
WriteFile
6. WriteFile(hWrite1, ...) ──────────→
   ReadFile(hRead2, ...)   ←──────────
```

Chromium 的 `base::LaunchProcess` 在 Windows 下使用 `CreateProcess` + `STARTUPINFO` 实现句柄继承，与 Linux 的 fd 继承语义一致。

Rust 侧统一代码：

无论 Linux 还是 Windows，sgclaw 的 Rust 代码完全一致：

```
// sgclaw 端无需区分平台
let reader = BufReader::new(std::io::stdin());
let writer = BufWriter::new(std::io::stdout());

for line in reader.lines() {
    let msg: PipeMessage = serde_json::from_str(&line?)?;
    let response = handle_message(msg).await?;
    writeln!(writer, "{}", serde_json::to_string(&response)?)?;
    writer.flush()?;
}
```

7.2 目标平台

平台	操作系统	CPU 架构	状态	备注
Primary	银河麒麟 V10 SP1	x86_64	目标部署环境	政企客户主要使用平台
Secondary	Windows 10 / 11	x86_64	开发 + 部分部署	开发团队日常使用 + 部分客户
Future	银河麒麟 V10 SP1	aarch64	规划中	国产 ARM 服务器/终端

银河麒麟 V10 SP1 特殊注意事项：

- 基于 Ubuntu 内核，但定制了安全策略（SELinux/AppArmor 变体）
- `procfs` 默认配置限制跨进程 fd 访问，有利于 Pipe Hijack 防御
- `kernel.yama.ptrace_scope=1`，非父进程无法 ptrace，增强安全性
- glibc 版本需确认兼容（通常 >= 2.31），Rust musl 静态链接可规避

7.3 sgClaw 二进制分发

sgClaw 编译为单一静态链接的 Rust 二进制，无外部运行时依赖。

```
sgclaw 二进制产物
├─ Linux:   sgclaw      (~8.8 MB, musl 静态链接, strip)
│           target: x86_64-unknown-linux-musl
│           无 glibc 依赖，银河麒麟直接运行
│
└─ Windows: sgclaw.exe  (~9.0 MB, MSVC 链接, strip)
            target: x86_64-pc-windows-msvc
            无额外 DLL 依赖
```

构建命令：

```
# Linux (musl 静态链接，确保银河麒麟兼容)
cargo build --release --target x86_64-unknown-linux-musl
strip target/x86_64-unknown-linux-musl/release/sgclaw
```

```
# Windows (交叉编译)
cargo build --release --target x86_64-pc-windows-msvc
```

部署方式：将 `sgclaw` / `sgclaw.exe` 放置在 SuperRPA 浏览器安装目录下：

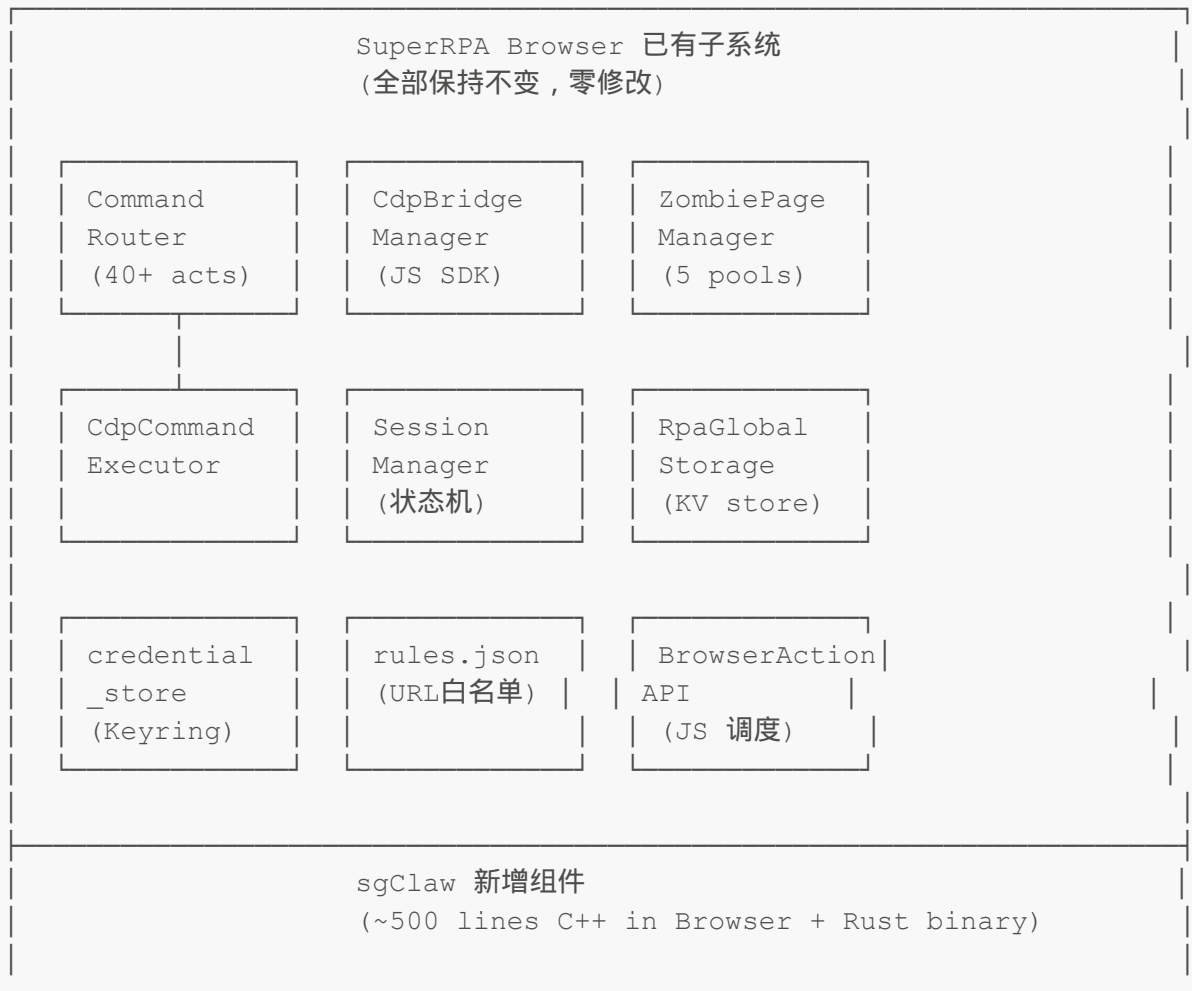
```
superrpa-browser/
├─ chrome                    (浏览器主程序)
├─ superrpa-chromium.sh      (启动脚本)
├─ sgclaw                    (AI Agent, 新增)
├─ sgclaw-skills/            (技能目录, 新增)
│   ├─ oa-approval.js
│   └─ ...
└─ resources/
    └─ rules.json            (域白名单, 已有)
```

---

8. 与 SuperRPA 已有子系统的关系

8.1 复用与新增总览

sgClaw 的核心设计原则之一是**最小侵入**——最大限度复用 SuperRPA 浏览器的已有基础设施，将新增代码量控制在最低水平。



8.2 复用子系统详细说明

子系统	代码位置	sgClaw 如何复用	是否修改
CommandRouter	browser/rpa/command/	sgClaw 通过 pipe → PipeListener → CommandRouter	不修改
CdpBridgeManager	browser/rpa/cdp/	独立运行，JS SDK 路径不受影响	不修改
ZombiePageManager	browser/rpa/zombie/	sgClaw 通过 zombieSpawn/zombieKill action 使用	不修改
SessionManager	browser/rpa/session/	sgClaw 可触发 sessionLogin (需确认)	不修改
RpaGlobalStorage	browser/rpa/storage/	sgClaw 通过 storageSet/storageGet 读写	不修改
BrowserAction API	browser/rpa/action/	已有 JS SDK 继续使用，与 sgClaw 独立	不修改
credential_store	browser/rpa/cred/	仅 SessionManager 内部使用，sgClaw 无法访问	不修改

子系统	代码位置	sgClaw 如何复用	是否修改
rules.json	resources/	sgClaw 的 MAC Policy 也读取此文件进行域校验	不修改

8.3 新增代码量估算

组件	语言	估计行数	位置
SgClawProcessHost	C++	~200-300	browser/rpa/sgclaw/
PipeListener	C++	~150	browser/rpa/sgclaw/
MAC Whitelist Check (Pipe)	C++	~100	browser/rpa/sgclaw/
FunctionsUI handlers	C++	~50	browser/rpa/functions_ui/
Side Panel UI controls	Vue	~100	resources/side_panel/
Browser 侧合计		**~500-600**	
sgclaw binary	Rust	~3000-5000	独立仓库 sgclaw/

8.4 影响评估

对已有功能的影响：零。

具体论证：

1. **CommandRouter 不变**：sgClaw 只是新增了一个命令来源（pipe），CommandRouter 本身的 dispatch 逻辑、action 处理函数、错误处理均不修改。
2. **JS SDK 不变**：CdpBridgeManager 独立于 SgClawProcessHost，两者通过不同的 IPC 路径 到达同一个 CommandRouter。JS SDK 的用户不会感知到 sgClaw 的存在。
3. **会话管理不变**：SessionManager 的状态机（login → active → expired → re-login） 完全不变。sgClaw 只能通过 CommandRouter 触发 sessionLogin action（需用户确认）， 不直接操作 SessionManager 内部状态。
4. **存储隔离**：sgClaw 通过 RpaGlobalStorage 的 key 被限制在 `sgclaw.*` 命名空间下， 不会与已有的 JS SDK 存储的 key 冲突。
5. **二进制独立**：sgclaw 是独立二进制，不链接 Chromium 的任何库。 即使 sgclaw 构建失败或缺失，浏览器仍然正常工作——只是 Side Panel 的 [启动] 按钮 会报错"sgclaw binary not found"。

文档结束。本文档为 sgClaw L1 层架构设计参考，后续 L2（详细设计）、L3（实现规范） 将在此基础上展开。