sgClaw 项目现有优势与下一步计划汇报稿

一、项目现有优势

和传统 openclaw 类自动化项目相比，本项目的优势不只是“能做页面操作”，而是已经具备了向企业级、长期可用方向演进的基础。简单说，传统方案更像“能跑起来的自动化脚本集合”，本项目更像“带安全边界、带统一控制、带长期演进能力的智能执行底座”。

而且根据后续已经落地的提交来看，本项目的优势已经不再只是架构上的“方向正确”，而是开始体现为一批已经交付的现实能力，包括：

- 运行时策略已经从写死逻辑转向配置驱动。
- 已支持 planner-first 的先规划后执行模式。
- 已支持技能包驱动的浏览器脚本执行。
- 已支持面向具体任务的 Excel 导出和屏幕展示导出能力。
- 已具备版本级日志、技能版本日志和真实验收记录。

这意味着本项目已经从“原型底座”进一步进入“可验证、可扩展、可交付”的阶段。

1. 从“脚本工具”升级为“统一执行底座”

传统项目通常是一个场景一套脚本，一个系统一套规则，能解决眼前问题，但难以复用、难以管理、难以持续演进。页面一改，脚本就要改；系统一多，维护成本就会快速上升。

本项目已经把任务接入、运行时控制、浏览器执行、日志回传、结果回传放进统一框架里。这样后续不管是新增能力、扩展场景，还是替换模型、替换策略，都不需要推倒重来，而是在同一个底座上持续增强。对业务侧来说，这意味着项目价值不再停留在“做通一个流程”，而是逐步沉淀为可以支撑更多业务的公共能力。

2. 安全设计更完整，更适合企业内网

这是本项目最突出的优势。传统自动化方案最大的问题，是一旦拿到页面操作能力，就容易变成“脚本想点什么就点什么”，安全边界不清楚，风险很难控制。本项目当前已经不是这种模式，而是把浏览器能力放进了严格受控的执行边界里。

从汇报角度，可以把当前安全设计概括为以下几组能力：

3 层安全机制

第一层是启动门禁。浏览器宿主和运行时必须先完成握手，版本不一致、超时、顺序错误，系统都不会进入正式运行状态。

第二层是运行时策略校验。每次真正要执行页面动作前，都会先检查动作是否合法、目标页面是否在允许范围内。

第三层是宿主二次校验。就算运行时已经发出了命令，浏览器宿主仍然会再做一次本地校验，防止异常命令真正落地。

这 3 层叠加起来，形成了“不是模型想做什么就做什么，而是每一步都要过关”的安全控制方式。

6 项协议硬约束

当前协议层已经明确了 6 项刚性要求：

1. 传输格式固定，不能随意乱发消息。
2. 编码方式固定，避免解析异常。
3. 单条消息大小有限制，防止异常数据冲击系统。
4. 序列号必须严格递增，不能重复、不能乱序。
5. 每条关键命令都必须带安全字段。
6. 一次请求只能对应一次响应，不能混乱回包。

这说明系统不是“发个命令过去试试看”，而是每条消息都有严格规则，更适合企业环境中的稳定运行。

2 类白名单

本项目当前至少有两类核心白名单同时生效。

第一类是域名白名单。只有允许的业务域名才可以被操作，不是浏览器里所有页面都能碰。

第二类是动作白名单。只有允许的动作类型才能执行，不是脚本写得出来就一定能跑。

白名单机制的意义在于，系统把“哪些页面能动、哪些动作能做”提前规定清楚，而不是把决定权完全交给模型或脚本。

1 类显式黑名单

除了白名单，本项目还明确保留了显式阻断项。也就是说，不只是“没允许的不行”，而是“高风险动作被直接明确禁止”。

这在企业场景下非常重要，因为有些能力不是“暂时不用”，而是“原则上就不能开放”。有了黑名单，系统在设计上就能提前规避高风险能力外溢。

5 个默认允许动作

当前默认真正开放给运行时执行的动作共有 5 个：

1. 点击
2. 输入
3. 页面跳转
4. 文本读取
5. 受控脚本执行

这里最重要的一点不是“多了一个动作”，而是这个新增能力并没有破坏安全边界。它不是把任意页面脚本能力全部放开，而是在现有受控协议和校验链路内，给技能包提供了一种更强但仍然可控的执行方式。

这看起来不如一些传统方案“动作数量多”，但它的价值恰恰在于边界非常清楚。先把最稳定、最可控、最容易审计的核心动作做好，再逐步扩展，比一开始把大量高风险动作全部开放更稳。

7 个默认允许域名

当前规则里默认允许的域名是有限集合，而不是浏览器里的所有网页都能碰。这样做非常符合企业内网环境的实际需求。对于办公系统、ERP、OA 等场景，大家真正需要的不是“全网自动化”，而是“在明确范围内可控地自动化”。

1 套 HMAC 签名机制

所有关键命令不是明文裸发，而是带签名校验。可以简单理解成，每条关键操作都会带“防伪标记”。

这样做的价值是，命令在链路中不容易被伪造、篡改或错误复用，整体安全性远高于普通脚本直接调用页面接口的模式。

1 套序列号机制

每条命令都有严格递增的序列号，而且一个序列号只能对应一个结果。

这让系统能够清楚知道“这条结果到底对应哪一次操作”，避免串包、乱序、错配等问题，提升稳定性和可追溯性。

3 重脚本执行约束

后续提交里新增了技能包驱动的浏览器脚本能力，但这部分并不是“把页面执行彻底放开”，而是在现有安全边界内增加了一层受控能力。

可以把它理解为 3 重约束：

1. 脚本必须来自技能包内的受管路径，不能越界读取技能目录之外的文件。
2. 执行时必须声明目标域名，不能脱离页面上下文随意运行。
3. 脚本仍然通过现有浏览器 pipe 和动作白名单执行，而不是绕过宿主直接落地。

这类设计很关键，因为它说明项目在增强能力的同时，仍然坚持“新增能力必须留在安全边界里”，而不是为了方便把安全口子越开越大。

5 类错误处理策略

系统不是失败了就“直接崩”，而是把错误分成不同类型处理。

- 有的错误不允许重试，直接失败。
- 有的错误可以限次重试。
- 有的错误需要等待配置或人工确认。
- 有的错误会触发熔断。
- 所有失败都要求结构化返回，便于定位问题。

这比传统脚本“报错了就人工重跑”的方式要成熟得多。

1 个熔断阈值

同一动作如果连续失败超过阈值，系统会主动停止继续尝试并通知界面，而不是无限重复。

这能有效避免错误状态下反复点击、反复提交、反复操作，减少业务风险和误操作成本。

7 项联调验收标准

当前项目已经把联调成功的标准写清楚了，包括：

1. 握手成功率要求
2. 版本不匹配的失败处理
3. 序列号异常场景处理
4. 超大消息拦截
5. 核心动作成功率要求
6. 结构化错误返回要求
7. 日志全链路贯通能力

这说明项目不是“靠经验凑合能跑”，而是已经开始形成可以复制、可以验收、可以交付的工程标准。

3. 浏览器只是执行面，不再定义整个系统

传统 openclaw 类项目常见问题是浏览器能力太强，最后整个系统都围着页面脚本转，浏览器脚本几乎变成了系统本体。

本项目已经明确把浏览器定义为“受保护的特权执行面”，而不是整个 runtime 本体。这意味着以后就算扩展到别的工具面、别的执行面，也不需要推翻现有架构，系统的演进空间更大，整体结构也更清楚。

4. 运行时能力已经从“写死逻辑”升级为“配置驱动”

这一点是后续提交中非常重要的进展。传统项目经常把模型、策略、模式、环境差异写死在代码里，导致后续一改就牵动整体。

本项目现在已经把一批关键决策收进运行时配置，包括：

- 使用哪个模型提供方
- 当前激活哪个 provider
- 使用什么 planner 模式
- 采用哪种 runtime profile
- 浏览器能力走哪种 backend
- Office 导出走哪种 backend
- skills 从哪个目录加载

从汇报口径上，可以把它概括为：

1 套统一 runtime config
3 种 runtime profile
多项可切换运行策略

这意味着系统不再只是“代码怎么写就怎么跑”，而是开始进入“按环境、按任务、按场景灵活切换”的阶段，更适合企业实际落地。

5. 已形成“先规划、再执行、再产出结果”的闭环能力

传统自动化项目往往是一上来就直接操作页面，缺少中间过程的可解释性，也不利于后续审计和治理。

本项目后续提交已经进一步加强了 planner-first 模式，也就是在真正执行之前，先给出计划，再按计划执行，再输出结果。对业务和管理层来说，这样的价值非常直接：

- 更容易理解系统准备做什么
- 更容易检查执行过程是否偏离目标
- 更容易把计划、执行、结果串成闭环

同时，本项目已经不是只有一个简单浏览器工具，而是开始形成更清晰的能力分工，例如：

- `superrpa_browser` 负责受控浏览器操作
- `openxml_office` 负责结果导出
- `screen_html_export` 负责展示类产物导出

这说明项目正在从“一个浏览器操作入口”走向“围绕业务结果组织工具链”的阶段。

6. 技能体系已经开始从“提示词描述”走向“可执行能力包”

这是本项目相对传统 openclaw 类项目非常重要的一个现实优势。很多传统项目里的“技能”更多只是提示词模板，真正落地时还是回到页面脚本堆叠。

本项目后续提交已经支持技能包驱动的浏览器脚本执行。简单理解，就是一个技能不再只是“告诉模型怎么做”，而是可以带着确定的脚本能力一起交付。这样做有几个明显好处：

1. 能力更稳定
   关键步骤不必完全依赖模型自由发挥，而是可以由打包好的脚本完成。

2. 可复用性更强
   同一个技能包可以在相似场景中重复使用，不必每次都重新组织页面操作。

3. 更适合沉淀企业资产
   后续很多高价值流程，都可以逐步从“提示词经验”沉淀成“可复用技能包”。

这意味着项目已经开始从“智能执行框架”走向“智能执行框架 + 可复用技能资产”的模式。

7. 前端只负责展示，不掌握执行权

传统项目里，前端、脚本、执行逻辑经常混在一起，最后变成“界面里藏了很多业务决策”。这种方式短期看开发快，长期看风险大、维护成本高。

本项目已经把前端限制为展示层，只负责展示状态、日志、计划和结果，不负责决定是否执行、如何切换模型、如何绕过安全边界。这样一来，系统结构更清楚，后续维护和升级时也更不容易失控。

此外，后续提交已经支持外部 frontend bundle 优先、内置资源兜底的装载方式。这意味着后续改界面、改展示逻辑，不必每次都重编浏览器宿主，研发效率和交付效率都会更高。

8. 配置能力更强，更适合业务落地

传统项目往往把很多关键逻辑写死在脚本里，修改一次就要重新改代码。这样不仅效率低，而且很容易因为局部修改牵动整体。

本项目已经开始把运行时配置、模型配置、策略配置从代码里抽出来，让宿主、运行时、前端之间的责任更清楚。这意味着未来业务调整、模型切换、策略升级都可以更平滑，而不是每次都进行大规模改造。

同时，后续提交还进一步加强了 source checkout 启动包装和 rules 同步能力，这对开发团队来说很重要。它意味着项目不仅适合做成二进制交付，也更适合在源码态持续联调和快速迭代。

9. 更适合做长期资产沉淀

传统自动化方案常见的问题，是做完一个流程后，价值基本也就结束了，经验很难积累成资产。

本项目不一样，它的方向是把执行能力、规则、安全边界、日志能力以及后续的元素识别能力，逐步沉淀成可复用资产。对企业来说，这种价值远高于“今天跑通一个流程”，因为它决定了未来是不是能够越做越快、越做越稳、越做越便宜。

现在这件事已经开始有现实支撑了。因为项目不只是在“能操作页面”，还已经能把技能、脚本、导出流程、运行时策略和日志标准逐步固化下来。后续再推进“混合自愈选择器”和元素指纹库时，这些都会自然成为资产沉淀的基础层。

10. 可观测性更强，已经开始具备运行级审计基础

传统项目常常只在失败时打印一段日志，出了问题很难知道系统到底做了什么。

本项目后续提交已经补上了一批很关键的运行级日志信息，包括：

- runtime 版本
- 协议版本
- 配置来源
- skills 目录解析结果
- runtime profile
- skills prompt mode
- 已加载技能及版本号
- 当前执行模式

这类能力的价值非常直接：它让系统开始具备“说清楚自己是怎么运行的”的能力。对研发、测试、验收和后续审计来说，这都是非常重要的基础。

11. 已经形成“真实验收”而不是“概念演示”

后续提交里，项目已经留下了更完整的验收记录，而不是停留在文档层面的能力描述。以知乎热榜 Excel 导出为例，当前已经形成真实验收结果，包括：

- 真实 provider 模式运行
- 实时热榜数据采集
- 结构化结果导出
- Excel 文件生成
- 验收打分

这说明项目已经不是“理论上可以做到”，而是已经在真实任务链路中证明“能够跑通、能够输出结果、能够形成验收记录”。

对外汇报时，这一点很重要，因为它代表项目已经从“能力设想”走向“能力验证”。

12. 工程化基础更好

本项目已经不是单纯的验证页面或原型，而是以运行时内核、协议、规则和测试为主的工程结构。这说明项目更接近“可持续建设的产品内核”，而不只是“一个能演示的自动化效果”。

从目前仓库状态看，已经有 20 多个顶层测试文件，覆盖协议、握手、runtime、配置、兼容层、导出工具、技能执行和验收评分等多个方面。这说明项目已经在往“可持续交付、可持续验证”的方向走，而不是停留在临时性脚本工程。

从长期看，这种工程化能力决定了项目能不能真正进入生产环境，能不能被更多团队协同使用，能不能在后续持续扩展能力。

13. 一句话总结现有优势

如果用非技术语言概括，本项目当前最大的优势可以总结为：

不是“更会点网页”，而是“已经具备了企业级智能执行系统该有的安全边界、控制能力、真实交付能力、稳定基础和长期演进空间”。

二、下一步计划

下一阶段的重点，不是继续堆脚本，而是进一步解决“页面一变就失效”的老问题，同时把项目能力从“能执行任务”继续提升为“能持续积累企业级自动化资产”。

1. 研发“混合自愈选择器”（Hybrid Self-Healing Selector）

在内网环境下，逐步摆脱对单一 XPath 的依赖，建立企业级元素指纹库，让系统在页面变化后依然能更稳地找到目标元素。

2. 定义元素指纹数据结构（JSON）

给每个可操作元素建立一份“数字档案”，核心字段包括：

- 语义文本
- A11y Role
- 相对空间位置
- 属性哈希
- 兜底 XPath
- 视觉切图（Base64 小图）

这样系统找元素时，不再只靠一条路径，而是像“多特征识别”。

3. 推进“影子录制”（Shadow Recording）机制

在现有传统 RPA 正常运行时，于底层开启影子模式。当旧脚本通过 XPath 成功命中元素并完成操作时，后台静默抓取该元素的完整指纹并写入本地数据库。

通过这种方式，在不额外增加大量人工录制成本的前提下，持续沉淀高价值元素资产库。

4. 开发穿透层能力

利用定制 Chromium 的底层权限，解决 `iframe` 和闭合 `Shadow DOM` 这类复杂页面结构下的定位难题，为后续自愈选择器提供更强支撑。

三、预期结果

通过下一阶段建设，本项目将从“能执行任务”进一步升级为：

- 在复杂企业页面中更稳
- 对页面变化更不敏感
- 更容易持续积累高价值资产
- 更适合在企业环境中长期推广使用

从业务视角看，项目价值也会从“完成单个流程自动化”进一步升级为“建设企业级智能执行底座”。