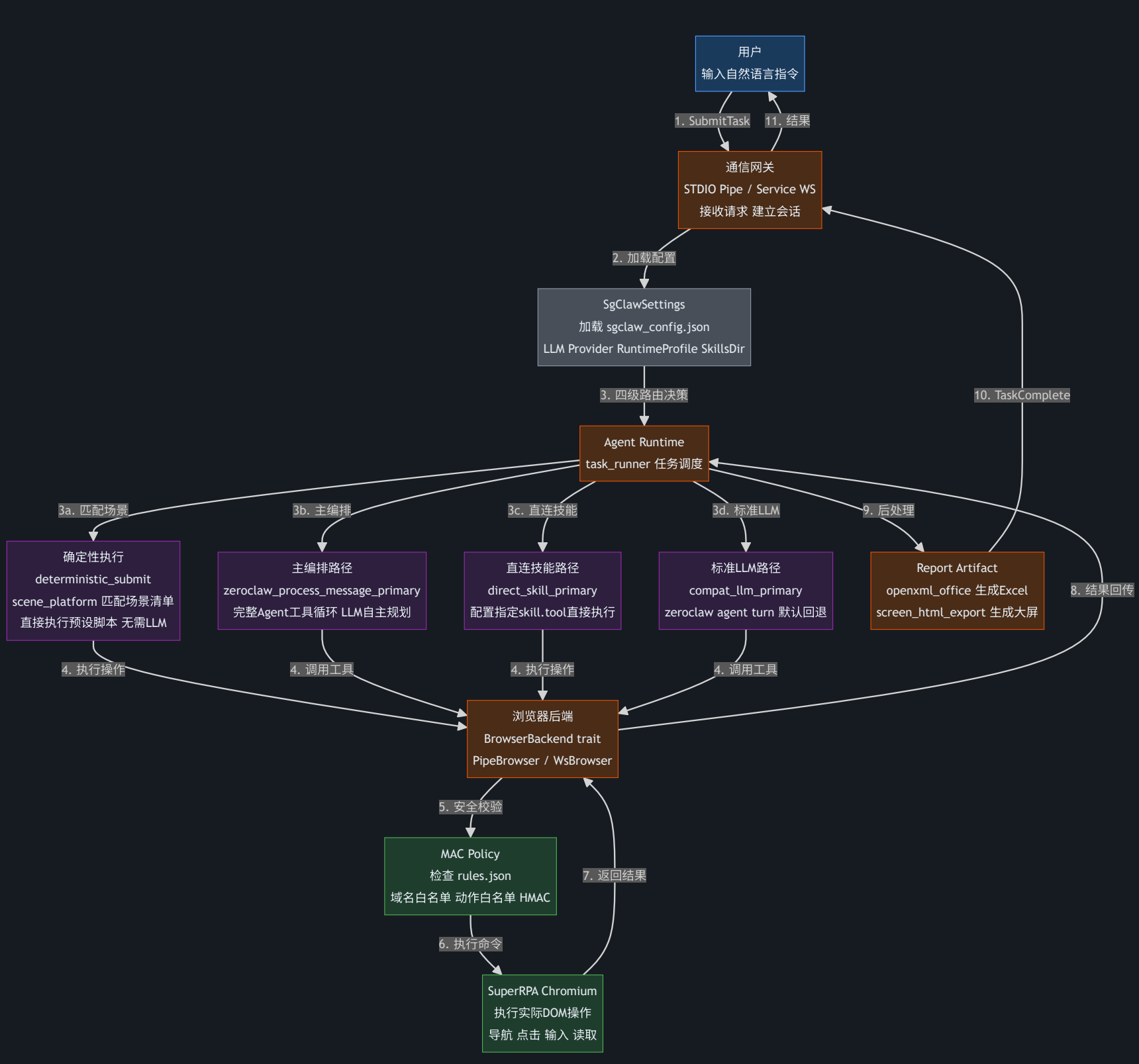
sgClaw 智能浏览器自动化平台

核心组件职责与流转全景图 - 每个组件是什么 做什么 什么时候调用

1 全景概览 - 从用户指令到浏览器执行的完整链路

> 当用户说出"帮我查本月线损率"时，sgClaw 内部多个组件协同工作。以下是**完整的执行链路**，展示每个组件在哪个环节被调用、承担什么职责。



2 核心组件详解 - 职责 调用时机 输入输出

以下是每个核心组件的详细说明。点击卡片可查看**什么时候调用**、**输入什么**、**输出什么**。

内部 通信网关

负责接收用户请求、建立会话、返回最终结果。支持两种模式：STDIO Pipe（默认，与浏览器宿主通过 stdin/stdout JSON Line 通信）和 Service WS（WebSocket 服务模式，接受外部客户端连接）。

何时调用: 用户发起请求时第一时间响应

输入: SubmitTask 消息（指令 conversationId pageUrl pageTitle）

输出: TaskComplete LogEntry StatusChanged 消息

内部 Agent Runtime 任务调度

run_submit_task() 是任务执行入口。依次执行四级路由决策：① deterministic_submit 确定性场景匹配 ② primary_orchestration 主编排 ③ direct_submit_skill 直连技能 ④ compat_llm_primary 标准LLM回退。

何时调用: SubmitTask 消息到达后

输入: 指令 AgentRuntimeContext BrowserPipeTool

输出: AgentMessage::TaskComplete

内部 场景平台 Scene Platform

扫描 skills/ 目录下的场景清单（scene.toml），解析 deterministic 段落的关键词规则。当用户指令匹配时，构建 DeterministicExecutionPlan（含 target_url org_code period_mode 等执行参数），直接执行预设脚本。

何时调用: 四级路由决策第一步

输入: 用户指令 pageUrl pageTitle skills目录

输出: DeterministicExecutionPlan 或 NotDeterministic

内部 SgClawSettings 配置管理

从 JSON 配置文件或环境变量加载运行时配置：多 Provider 管理（apiKey baseUrl model）、Runtime Profile、SkillsDir、BrowserBackend 类型、OfficeBackend、Service WS 监听地址等。

何时调用: 每次任务提交时加载

输入: sgclaw_config.json 或环境变量

输出: SgClawSettings 结构体

内部 Runtime Engine 运行时引擎

根据 Runtime Profile（BrowserAttached/BrowserHeavy/GeneralAssistant）构建 Tool Policy 白名单，加载技能包，注入 Memory，构建 Agent 实例。同时负责指令增强（附加浏览器合约提示、检测特定任务类型）。

何时调用: 主编排路径和标准LLM路径构建Agent时

核心方法: build_agent() build_instruction()

Profile: BrowserAttached / BrowserHeavy / GeneralAssistant

外部 ZeroClaw Core 智能体核心

位于 third_party/zeroclaw/ 的 vendored Agent 核心库。提供 Agent 构建、Provider 管理、工具循环、Memory 接口、技能加载、Prompt 组装等核心能力。sgClaw 在其基础上叠加安全信封层。

何时调用: 主编排和标准LLM路径中

位置: third_party/zeroclaw/

核心能力: Agent Provider ToolLoop Memory Skills

内部 Browser Backend 浏览器后端

统一的浏览器操作接口（BrowserBackend trait）。两种实现：PipeBrowserBackend（通过 STDIO 与宿主通信）和 WsBrowserBackend（通过 WebSocket 直连 DevTools）。支持 SuperRpa/AgentBrowser/RustNative/ComputerUse 多种后端类型。

何时调用: 需要操作浏览器时

内部 MAC Policy 安全策略

从 resources/rules.json 加载安全规则。三层安全模型：①握手时 HMAC seed 交换和会话密钥派生 ②Rust 侧域名+动作白名单校验 ③宿主侧 HMAC 二次验证。拒绝不在白名单的域名和被禁用的动作。

何时调用: 每次浏览器操作执行前

检查项: 域名白名单 动作类型 HMAC验证

支持操作: navigate click type getText eval select scrollTo 等15种

外部 **SuperRPA Chromium 浏览器宿主**

实际执行 DOM 操作的外部系统。接收 sgClaw 的 Command（含 HMAC），验证后执行 navigate/click/type/getText 等操作，返回 Response（含操作结果 + HMAC）。STDIO 模式下与 sgClaw 进程通过管道通信。

何时调用: BrowserBackend 发送命令时

通信协议: STDIO JSON Line 或 WebSocket

3 LLM 大模型工作全流程 - 从语义识别到任务规划

当用户指令无法匹配已知技能时，LLM 大模型开始工作。以下是**大模型从理解用户意图到生成可执行计划的完整过程**。

1 语义识别 - 理解用户说了什么

LLM 接收用户自然语言指令，识别用户的**真实意图**。例如"帮我查本月线损率" → 识别为"查询线损率数据"。

2 场景匹配 - 判断是否为已知场景

结合 Memory（记忆模块）中存储的历史任务记录，判断该指令是否与已有技能匹配。如果匹配，转交快速通道执行。

3 任务拆解 - 将大目标分解为小步骤

如果是新场景，LLM 将用户目标拆解为具体的、可操作的步骤序列。例如：打开系统 → 选择月份 → 点击查询 → 读取数据 → 导出Excel。

4 工具选择 - 决定用什么能力完成任务

LLM 根据步骤需求，从可用工具库中选择合适的工具。例如：需要打开网页选择"导航工具"，需要点击按钮选择"点击工具"，需要读取数据选择"读取工具"。

5 参数填充 - 确定每个工具的具体参数

LLM 为每个工具填充具体参数。例如点击工具需要知道"点击哪个按钮"，导航工具需要知道"打开哪个URL"。这些参数从用户指令和上下文中提取。

6 执行计划生成 - 输出可执行的JSON/结构化指令

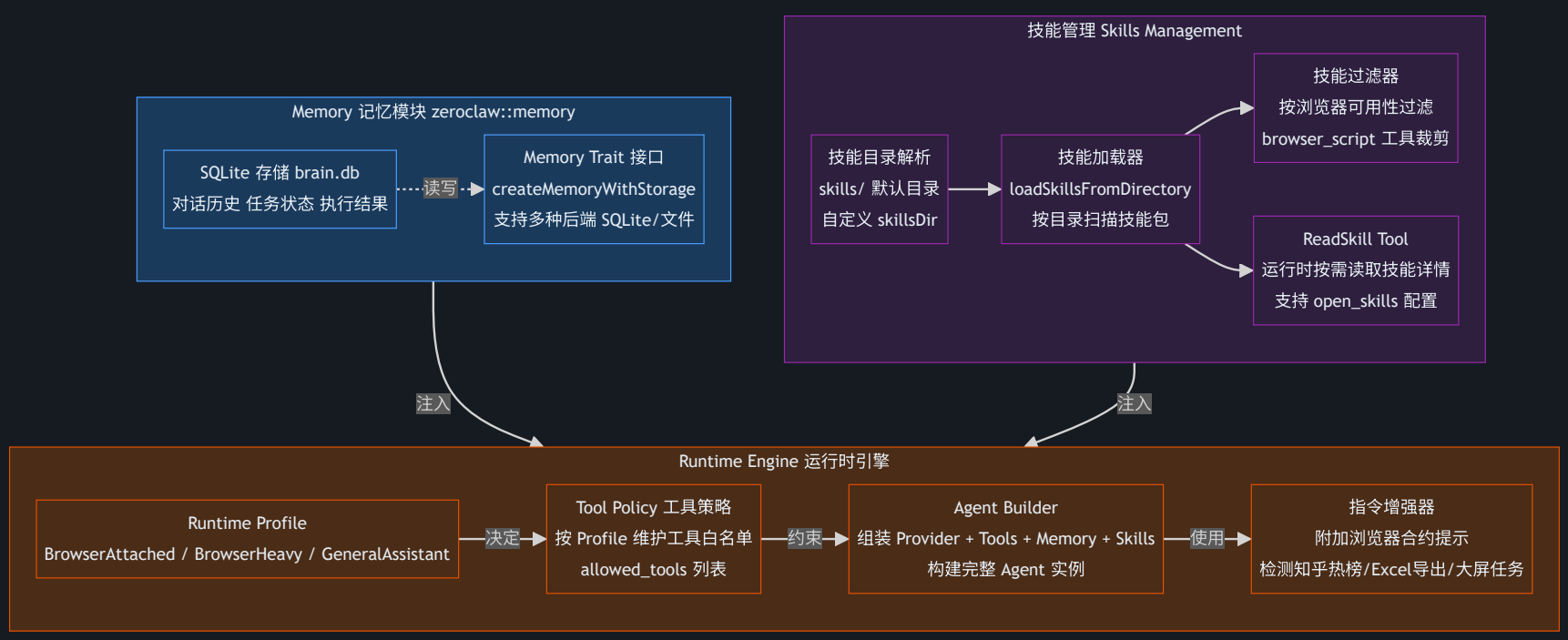
LLM 将拆解的步骤、选择的工具、填充的参数整合为**结构化的执行计划**，交由工具执行引擎依次执行。

7 循环迭代 - 根据执行结果动态调整

如果某一步执行失败或结果不符合预期，LLM 会收到反馈，重新规划后续步骤。例如页面打不开则尝试备用URL，元素找不到则换选择器。

4 Memory 技能管理 与 Runtime Engine - 运行时核心引擎

sgClaw 的运行时核心由三大引擎协同工作：**Memory（记忆模块）**负责持久化存储对话历史与任务状态，**技能管理系统**负责加载和注入技能包到 Agent，**Runtime Engine**负责根据 Runtime Profile 构建完整的 Agent 运行环境（工具策略 + 技能加载 + 指令增强）。



内部 Memory 记忆模块

职责:基于 SQLite（brain.db）持久化存储对话历史、任务状态和执行结果。通过 zeroclaw::memory::Memory trait 提供统一接口，支持多种存储后端。

何时调用: Agent 构建时创建 每次 LLM 调用前后读写

调用者: Runtime Engine（build_agent 方法）

存储路径: workspace/memory/brain.db

内部 技能管理系统

职责:从 skills/ 目录（或自定义 skillsDir）扫描加载技能包，按浏览器是否可用过滤 browser_script 工具，通过 ReadSkill Tool 让 Agent 按需读取技能详情。支持 open_skills 独立技能目录配置。

何时调用: 每次 Agent 构建时加载技能列表

调用者: Runtime Engine（load_skills_for_surface）

技能来源: workspace/skills/ 或 skillsDir 配置

内部 Runtime Engine

职责:运行时核心编排器。根据 Runtime Profile 决定工具白名单，加载技能，注入 Memory，构建 Agent 实例。同时负责指令增强（附加浏览器合约提示、检测特定任务类型如知乎热榜/Excel导出/大屏展示）。

何时调用: 每次任务提交时 构建 Agent 前

核心方法: build_agent() build_instruction()

Profile: BrowserAttached / BrowserHeavy / GeneralAssistant

5 任务路由 - 四种执行路径决策树

任务提交到 sgClaw 后，**Agent Runtime** 按优先级依次判断走哪条执行路径。这不是简单的"快速/AI"二选一，而是**四级决策树**。

1 确定性场景匹配 - deterministic_submit

通过 scene_platform 模块扫描 skills/ 目录下的场景清单（scene.toml），匹配指令关键词、URL、页面标题。匹配成功则构建 DeterministicExecutionPlan，直接执行场景预设的浏览器脚本，**无需 LLM 参与**。典型场景：线损查询、报表导出等固定流程。

2 主编排路径 - zeroclaw_process_message_primary

当 Runtime Profile 启用浏览器工具（browser_surface_enabled）且 orchestration::should_use_primary 判定走主编排时，调用 zeroclaw 的 process_message 完整 Agent 循环。LLM 可以调用所有允许的工具（浏览器操作、技能工具等），支持多轮工具调用和动态规划。

3 直连技能路径 - direct_skill_primary

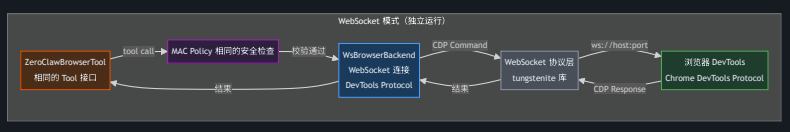
当配置中设置了 directSubmitSkill（格式：skillName.toolName），绕过正常 Agent 循环，直接执行指定的技能工具。适用于需要固定流程但又不适合确定性场景的中间态。

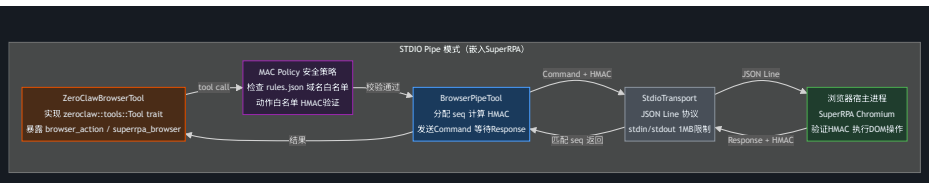
4 标准 LLM 路径 - compat_llm_primary

以上三条路都不通时的默认回退。创建标准 zeroclaw Agent turn，LLM 根据指令自主决定使用哪些工具。这是最灵活但也最慢的路径。

6 浏览器执行全过程 - 从sgClaw到SuperRPA浏览器的命令传输

sgClaw 有两种浏览器后端模式：**STDIO Pipe 模式**（sgClaw 进程通过 stdin/stdout 与浏览器宿主通信）和 **WebSocket 模式**（直接连接浏览器 DevTools WebSocket）。安全校验在两种模式下都由 MAC Policy 层负责。





内部 ZeroClawBrowserTool

职责:实现 zeroclaw::tools::Tool trait，将 BrowserBackend 适配为 LLM 可调用的工具。暴露两个工具名：browser_action（传统别名）和 superrpa_browser（SuperRPA 专用，优先使用）。

何时调用: LLM 决定操作浏览器时

文件位置: compat/browser_tool_adapter.rs

内部 MAC Policy 安全策略

职责:从 resources/rules.json 加载安全规则。三层安全检查：①握手时 HMAC seed 交换 ②Rust 侧域名+动作白名单校验 ③宿主侧 HMAC 二次验证。拒绝不在白名单的域名和被禁用的动作。

何时调用: 每次浏览器工具调用前

规则文件: resources/rules.json

内部 BrowserBackend 浏览器后端

职责:统一的浏览器操作接口（BrowserBackend trait）。两种实现：PipeBrowserBackend（通过 StdioTransport 与宿主通信）和 WsBrowserBackend（通过 WebSocket 直连 DevTools）。由 BrowserBackend 配置决定使用哪种。

后端类型: SuperRpa / AgentBrowser / RustNative / ComputerUse / Auto

文件位置: browser/pipe_backend.rs browser/ws_backend.rs

内部 BrowserPipeTool

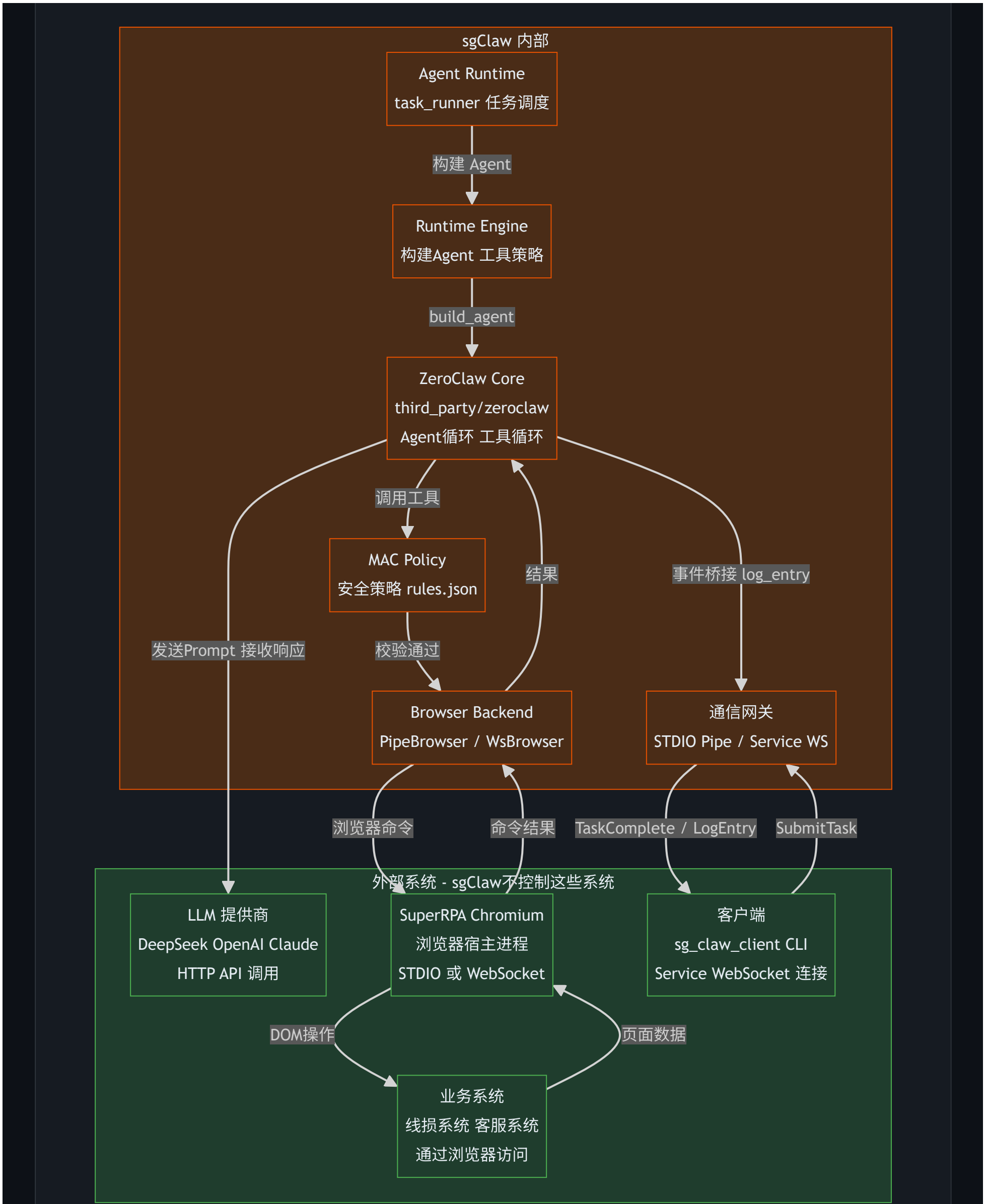
职责:STDIO Pipe 模式下的特权浏览器工具。为每个命令分配单调递增 seq，使用派生会话密钥计算 HMAC，发送 Command 消息后阻塞等待匹配的 Response，支持超时。

何时调用: Pipe 模式下每次浏览器操作

文件位置: pipe/browser_tool.rs

7 外部系统关系图 - sgClaw与谁交互

> sgClaw 不是孤立运行的，它与多个**外部系统**协同工作。以下是sgClaw与外部系统的交互关系。



8 完整生命周期 - 一个任务从出生到结束

以一个真实场景为例：**"帮我查本月线损率并导出Excel"**，展示sgClaw从接收指令到返回结果的完整生命周期。

1 通信网关接收指令

浏览器宿主进程通过 STDIO（JSON Line 协议）发送 SubmitTask 消息。sgClaw 创建会话，解析指令、page_url、page_title、conversation_id。

2 加载配置SgClawSettings

从 sgclaw_config.json 或环境变量加载配置：LLM provider（apiKey/baseUrl/model）、runtimeProfile、skillsDir、directSubmitSkill 等。

3 确定性场景匹配deterministic_submit

扫描 skills/ 目录下的场景清单（scene.toml），发现指令包含"线损率"、"本月"关键词，匹配到"线损查询"场景。构建 DeterministicExecutionPlan（含 target_url、org_code、period_mode 等参数）。

4 MAC Policy安全校验

检查目标域名是否在 rules.json 白名单中 → 通过。检查操作类型（navigate、click、getText）是否在动作白名单中 → 通过。

5 BrowserPipeTool执行浏览器命令

为每个命令分配单调递增 seq，使用派生会话密钥计算 HMAC。通过 StdioTransport 发送 Command 消息给浏览器宿主。执行：导航到线损系统 → 选择月份 → 点击查询 → 读取表格数据。

6 SuperRPA Chromium执行DOM操作

浏览器宿主接收 Command，验证 HMAC，执行实际 DOM 操作（导航、选择下拉框、点击按钮、读取表格内容），返回 Response（含操作结果 + HMAC）。

7 Report Artifact后处理

将浏览器返回的表格数据解析为结构化格式。根据场景的 postprocess 配置，使用 openxml_office 工具生成 .xlsx 文件。生成结果包含本地文件路径。

8 通信网关返回结果

通过 StdioTransport 发送 TaskComplete 消息给浏览器宿主，包含 success=true 和执行摘要（含生成的 .xlsx 文件路径）。浏览器宿主提示用户下载完成。