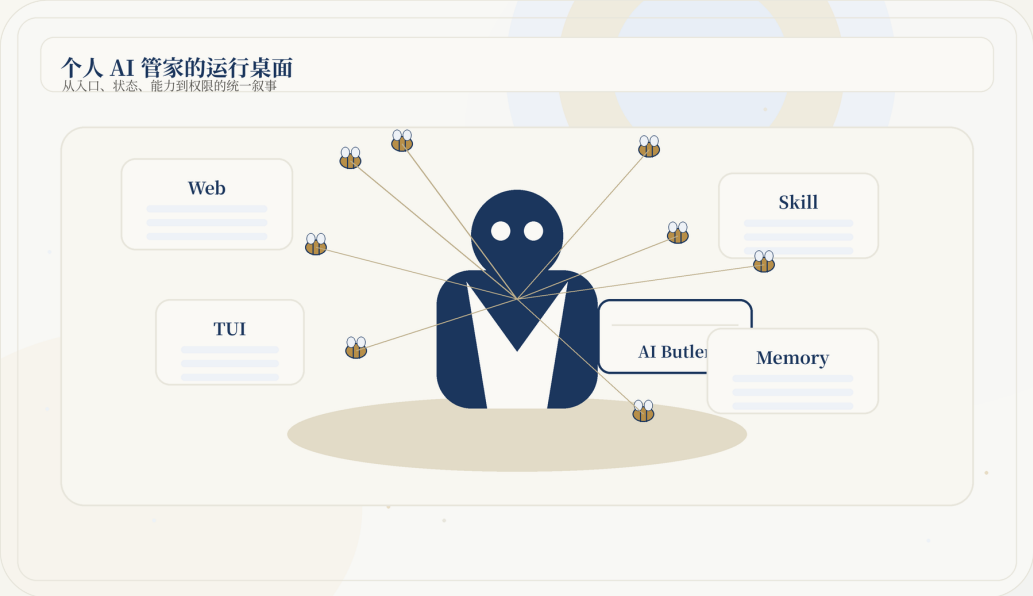
PROJECT FIELD GUIDE

JiuwenSwarm 打造可进化的个人 AI 管家

第四版出版增强：版权页、来源声明与预检报告



基于 AtomGit 仓库 `openJiuwen/jiuwenswarm` develop 分支资料整理
版本视角：2026-05-26 · 第四版出版增强

版权页与出版声明

项目	内容
书名	JiuwenSwarm: 打造可进化的个人 AI 管家
版本	第四版出版增强版
版本日期	2026-05-26
编著整理	OpenAI GPT-5.5 Pro 整理
资料来源	AtomGit 仓库 `openJiuwen/jiuwenswarm` develop 分支公开资料、项目 README、docs 文档与关键入口代码
适用范围	内部分发、技术培训、项目介绍、开源社区传播、出版送审前样稿
出版状态	本版未申请 ISBN/CIP；如需正式出版，应由出版主体完成审校、权利审核和备案流程

来源与授权说明

本书是基于公开仓库资料进行的结构化整理、解释性写作和排版再创作，不是 JiuwenSwarm 项目官方文档的逐字复制，也不代表华为、JiuwenSwarm 项目维护者或相关平台的官方出版物。源项目的代码与文档授权以仓库中的 `LICENSE` 和各文件实际声明为准；本书文字、重绘图示和原创插图的使用授权应由最终发布主体另行确认。

商标与名称声明

书中出现的 JiuwenSwarm、Huawei Cloud、Xiaoyi、小艺、Feishu、Lark、DingTalk、Discord、WhatsApp、WeCom、GitHub、OpenAI、DeepSeek、Anthropic 等名称均归其各自权利人所有。本书仅为技术说明和学习目的引用，不构成任何背书、合作或隶属关系声明。

插图与截图说明

本书中的解释性架构图和美术插图均为本书制作的示意素材，用于帮助理解系统结构和阅读节奏。除非明确标注为官方截图，否则不应被视为 JiuwenSwarm 官方界面、官方架构图或实际产品页面复刻。

免责声明

本书内容依据 2026-05-26 时点可访问的公开资料整理。开源项目迭代较快，命令、路径、配置字段、默认端口和第三方平台规则可能随版本变化而调整。生产部署、商业出版、企业归档或平台上架前，应以仓库最新代码、官方文档、许可证文件和法律/合规审查结果为准。

目录

前言: 为什么这不是一本普通的安装手册

本书不是官方文档的逐字搬运，而是围绕 JiuwenSwarm 的产品主线、系统架构、使用路径和二次开发方法做的一次结构化再创作。第四版在第三版基础上补齐出版声明、来源边界和预检报告；第三版已在第二版基础上调整了章节节奏：把“本章扩展”统一移到章末，改为“章末延伸与实践”，并在原有 22 张解释性图示之外新增 9 张原创美术插图。书中命令和配置应以仓库实际代码与文档为准；正式部署前请再次核对最新版本。

本书阅读路线图。本图为基于仓库文档与代码入口重绘的解释性示意图。

第三版阅读说明：先正文，后延伸

第一版更接近资料导读，第二版把每章扩成教程型结构，但“本章扩展”出现在章首会打断主线。第三版把这些内容统一后移到每章末尾，作为“章末延伸与实践”：先读正文建立概念，再读延伸区理解真实场景、检查表和误区。

阅读时可以把本书分成三条路线。第一条是使用路线：先跑起来，再学会模型配置、模式切换、会话管理、任务规划和记忆清理。第二条是集成路线：从 Web/TUI 扩展到飞书、小艺、Cron、Heartbeat、Browser MCP，再进入权限和多实例。第三条是开发路线：理解工作区、AgentServer/Gateway 分离、E2A 信封、Skill 自进化和分布式 Team。

图示并非装饰，而是“读代码前的地图”。第三版保留原有解释性图示，并增加少量美术插图，用来调节长文节奏：解释图回答“结构怎么运转”，美术图回答“这个模块在产品叙事里像什么”。如果你只想快速使用，可以先看每章图和检查表；如果你要二次开发，再回到文字细节和源码路径。

很多开源 Agent 项目的文档，最后都会落成两类内容: 一类是「怎么 pip install」, 另一类是「怎么配置 API Key」。这当然重要，但远远不足以理解 JiuwenSwarm。

JiuwenSwarm 的核心价值，不在于它多提供了一个聊天页面，而在于它试图把一个个人 AI 助手做成可长期运行、可接入多个入口、可拥有记忆、可通过 Skill 扩展能力、可被权限系统约束、还可以通过调度和心跳主动工作的系统。换句话说，它接近一个「个人 AI 操作系统」雏形。

README 给出的定位是: JiuwenSwarm 是用 Python 构建的智能 AI Agent, “Swarm” 表示多个智能体像蜂群一样协同工作。项目强调生态兼容、与小艺开放平台集成、自托管部署和多平台访问。这个描述很短，但已经包含三层含义:

1. 它不只服务于单一 UI, 而是要成为多通道的 Agent 后端。
2. 它不只响应单轮问题，而是要围绕任务、记忆和技能持续演进。
3. 它不只面向个人玩具场景，而是考虑了权限、工作区、配置隔离、多实例和团队协作。

本书适合三类读者:

- 想把 JiuwenSwarm 跑起来的使用者;
- 想把它接入飞书、小艺、钉钉、Web、TUI 或自定义通道的集成者;
- 想研究可进化 Agent、Skill 系统、记忆系统和多进程网关架构的开发者。

阅读方式建议如下: 第 1 到第 4 章用于建立全局认知；第 5 到第 9 章用于掌握日常使用；第 10 到第 13 章用于理解高级部署和二次开发；最后的附录可以作为速查表。

第一章 项目全景: JiuwenSwarm 到底解决什么问题

JiuwenSwarm 能力地图。本图为基于仓库文档与代码入口重绘的解释性示意图。

1.1 一句话定义

JiuwenSwarm 是一个以 Python 为主体的个人 AI Agent 系统。它把模型调用、会话入口、任务规划、记忆、技能、浏览器自动化、定时调度和工具权限管理组合到同一个运行体系里，让用户可以通过 Web、TUI、飞书、小艺等渠道与一个持续运行的 AI 管家交互。

如果只把它看作聊天机器人，会低估它。更准确的视角是:

> JiuwenSwarm 是一个以 AgentServer 为执行核心、以 Gateway 为通道枢纽、以 Workspace 为长期状态容器、以 Skills 和 Memory 为能力扩展层的个人 AI 管家系统。

1.2 项目的四个关键词

数据主权

README 明确强调自托管部署和数据主权。对个人助理而言，数据主权不是口号，而是基础设施需求。记忆文件、会话文件、技能目录、配置文件和日志都不应该只存在于第三方平台的黑盒里。JiuwenSwarm 把运行态数据放在用户工作区，典型目录是 `~/.jiuwenswarm` 或命名实例自己的工作区。

多入口

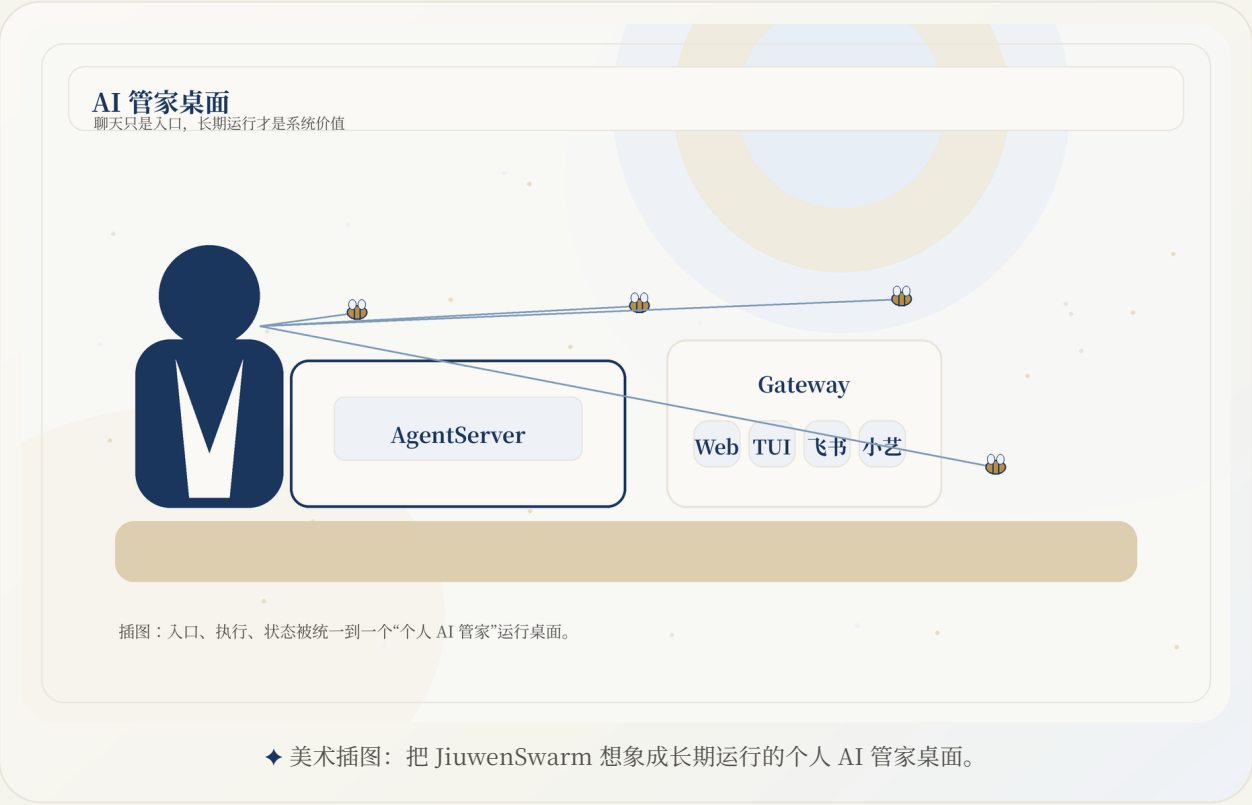
用户可以从 Web 前端、TUI、飞书、小艺、钉钉、WhatsApp、企业微信等入口触发 Agent。多入口的难点不是「多写几个适配器」，而是如何把不同平台的消息、文件、事件和会话语义统一起来。JiuwenSwarm 使用 Gateway 和 E2A 等内部协议来做归一化。

可进化

Skill 不是一次性提示词。Skill 的执行失败、用户纠正和上下文经验可以写入 `evolutions.json`，进一步合并回 `SKILL.md`。这意味着 Skill 有机会成为活文档: 每次出错都成为下一次更稳的素材。

主动性

很多 Agent 项目只能在用户提问时工作。JiuwenSwarm 增加了定时任务、Heartbeat、任务规划和长期记忆，目标是让它在合适的时机主动执行工作。例如每天九点总结待办，或者按心跳读取 `HEARTBEAT.md` 中的活跃任务。



✦ 美术插图：把 JiuwenSwarm 想象成长期运行的个人 AI 管家桌面。

1.3 功能地图

入口层
Web / TUI / Xiaoyi / Feishu / DingTalk / WhatsApp / WeChat / A2A / ACP

网关层
ChannelManager / MessageHandler / Gateway routes / Cron / Heartbeat

执行层
AgentServer / openJiuwen Runner / TeamManager / Browser runtime

状态层
Workspace / Session / Memory / Skills / Config / Logs

能力层
Skills / MCP / Browser tools / Task memory / Coding memory

安全层
Permission engine / allow-ask-deny / external directory / owner scopes

从使用者角度看，最常用的是入口层、配置层和技能层。从开发者角度看，最值得研究的是 Gateway 与 AgentServer 的分离、E2A 协议、Skill 自进化、记忆索引和权限决策。

1.4 本书采用的项目版本视角

本书基于 AtomGit 仓库 `openJiuwen/jiuwenswarm` 的 `develop` 分支资料整理。`pyproject.toml` 中的项目名为 `jiuwenswarm`，版本为 `0.2.0`，Python 版本要求为 `>=3.11,<3.14`，许可证声明为 Apache-2.0。该文件还列出了 ChromaDB、pgvector、FastAPI、uvicorn、websockets、croniter、discord.py、python-telegram-bot、OpenTelemetry 等依赖，说明项目覆盖向量记忆、服务端 API、多通道通信、调度、观测等能力。

章末延伸与实践：把 JiuwenSwarm 看成一个“长期运行的个人 Agent 系统”

理解 JiuwenSwarm 的关键，是不要把它压缩成“一个聊天前端”。聊天只是入口，真正的系统价值在于它把入口、状态、能力和安全放进同一套运行框架。入口负责接收来自 Web、TUI、飞书、小艺等不同平台的消息；网关负责将它们整理成统一的会话和请求；执行层负责调模型、调工具、调 Team；状态层保存记忆、技能、会话产物和配置；安全层则决定工具能否被直接调用。

这类架构的好处是复用。你为 Web 配好的 Skill，不应该只能在 Web 用；你在飞书里沉淀的任务记忆，也不应该和 TUI 割裂；你给浏览器自动化设置的权限，也不应该因为换了入口就失效。JiuwenSwarm 的设计方向就是把这些能力向中间收拢，让不同入口共享同一套 Agent 能力。

但这种设计也带来复杂度。第一次使用时，用户会接触模型配置、工作区、通道、权限、心跳、任务、Skill 等多个概念。如果没有地图，很容易把“启动失败”“模型失败”“前端失败”“通道失败”“权限拒绝”混在一起。本章的建议是：先按层排查。入口层看连接，网关层看路由，执行层看模型和工具，状态层看工作区，安全层看权限。

真实场景：个人助理与企业入口并存

假设你本地通过 TUI 让 Agent 维护项目文档，同时在飞书群里让它作为数字分身处理与自己相关的提醒。此时两个入口共享同一个 Agent 认知，但场景风险不同：TUI 可以让你逐步确认文件操作，群聊数字分身不能在每一步都问你。因此飞书群聊必须配更明确的 owner scope 和工具白名单。这就是为什么多入口系统一定要有统一权限层，而不是每个入口各自临时判断。

章末检查表

- 先确认你要使用 JiuwenSwarm 的哪一层能力：聊天、工具、记忆、Skill、通道、团队，还是调度。
- 不要把所有问题都归咎于模型；Agent 系统故障常常来自配置、路径、端口、权限或通道。
- 建议先用 Web/TUI 建立最小闭环，再接入飞书、小艺和 Cron 等外部入口。

第二章 五分钟启动: 从零跑起一个 JiuwenSwarm

首次启动闭环。本图为基于仓库文档与代码入口重绘的解释性示意图。

2.1 环境要求

快速上手文档给出的基础要求很直接:

依赖	建议版本	用途
操作系统	Windows 10/11, macOS 10.15+, Linux	主流桌面和服务器环境
Python	>=3.11, <3.14	运行后端与 Agent
Node.js	18.x 或更高	Web 前端构建或开发
Git	最新稳定版	源码安装与开发

如果只是 pip 安装和使用已发布包，Node.js 并不总是第一时间暴露在用户面前；但如果你从源码安装、构建 Web 前端或调试浏览器自动化，Node.js 会成为必要依赖。

2.2 pip 安装路径

对大多数用户，建议从 pip 安装开始。先创建虚拟环境，再安装包:

```
python -m venv jiuwenswarm-env
source jiuwenswarm-env/bin/activate
pip install jiuwenswarm
```

Windows 激活虚拟环境时使用:

```
jiuwenswarm-env\Scripts\activate
```

首次运行通常分两步:

```
jiuwenswarm-init
jiuwenswarm-start
```

`jiuwenswarm-init` 负责初始化用户工作区、复制配置模板、写入语言偏好和初始化 Agent 模板。`jiuwenswarm-start` 则负责启动服务。

2.3 启动后应该看到什么

正常启动后，后端 API、Web 服务和 Gateway 会以各自端口运行。文档中的典型访问地址是:

```
http://localhost:5173
```

打开后进入 Web UI, 你首先应该完成模型配置，而不是直接期待聊天成功。安装完成不等于可用；模型 API、Provider 和 Key 才是 Agent 能否工作的关键。

2.4 模型配置的最小集合

最小可用配置包含四个字段:

字段	环境变量	说明
`model_name` 或 `model`	`MODEL_NAME`	模型 ID, 如 `deepseek-chat`, `gpt-4o`
`api_base`	`API_BASE`	模型服务 Base URL, 不要手动追加 `/chat/completions`
`api_key`	`API_KEY`	模型服务密钥
`model_provider`	`MODEL_PROVIDER`	Provider 类型, 如 OpenAI, DeepSeek, SiliconFlow

配置页面提供 Test 按钮。测试失败时先检查三件事: API Key 是否正确，Base URL 是否可访问，模型名和 Provider 是否匹配。

2.5 TUI 启动路径

如果你偏好终端界面，还可以安装并启动 TUI:

```
pip install jiuwenswarm-tui
jiuwenswarm-tui
```

TUI 适合开发者和重度命令行用户。它支持工作区路径命令、上下文压缩命令等只在 TUI 侧生效的控制命令。

2.6 源码安装路径

源码安装适合二次开发者。总体流程是:

```
git clone https://gitcode.com/openJiuwen/jiuwenswarm.git
cd jiuwenswarm
uv venv
uv pip install -e .
```

源码安装要特别注意前端构建。安装指南强调: editable install 不会自动把前端 `dist` 带入用户工作区；如果没有构建和复制前端产物，启动可能会失败，报出 `dist directory not found` 一类错误。

典型流程:

```
cd jiuwenswarm/channels/web
npm install
npm run build
cp -r dist ~/.jiuwenswarm/channels/web/frontend/dist
```

实际路径可能随仓库结构调整而变化，部署时以当前文档和目录为准。

2.7 首次启动检查表

01 Python 版本在 3.11 到 3.13 范围内。

02 虚拟环境已激活， `pip install jiuwenswarm` 成功。

03 执行过 `jiuwenswarm-init` , 工作区已创建。

04 `~/.jiuwenswarm/config/.env` 或 Web UI 中已经配置模型。

05 `jiuwenswarm-start` 后可以访问 Web UI。

06 模型 Test 通过，再进入 Chat 页面提问。

章末延伸与实践：首次启动的“最小闭环”

很多安装问题不是因为命令错，而是因为没有把“安装”“初始化”“配置模型”“启动服务”“验证入口”看成一个闭环。`pip install jiuwenswarm` 只把程序装进环境；`jiuwenswarm-init` 才会把配置模板、工作区和 Agent 模板放到用户数据目录；`jiuwenswarm-start` 只是启动进程；模型配置成功后，Agent 才能真正回答问题。

最稳的做法是一次只验证一个层级。第一步验证 Python 版本和虚拟环境，避免把系统 Python、conda、uv 环境混在一起。第二步执行 `jiuwenswarm-init`，确认用户目录下出现 `.jiuwenswarm` 工作区。第三步启动服务，确认 Web 页面可以打开。第四步进入配置页填写模型，并使用 Test 按钮验证模型。第五步才开始提问。任何一步失败，都不要跳到后面。

源码安装更容易踩坑，因为前端构建产物不一定存在。editable install 的优点是便于改代码，缺点是你要自己处理前端 `dist`、Node 依赖和复制路径。若你的目标是试用，而不是开发，优先 pip 安装；若你要改通道、改权限或调试前端，再切源码路径。

建议的启动验收

验收点	通过标准	失败时优先看哪里
Python 环境	`python --version` 在 3.11-3.13	虚拟环境是否激活
包安装	能找到 `jiuwenswarm-init` 命令	pip 安装日志和 PATH
工作区	用户目录出现 `.jiuwenswarm/config`	初始化是否被取消或权限不足
服务启动	Web 页面可打开	端口冲突、日志、前端产物
模型测试	Test 成功返回	api_base、api_key、provider、model

常见误区

不要把 `api_base` 写成完整的 `/chat/completions` 地址，很多 Provider 适配会自动拼接路径。不要一开始就开多个实例、多个通道和多个模型；首次验证只保留一个默认模型和一个 Web 入口。不要在源码安装时忽略前端构建，因为后端成功启动并不代表 Web UI 资源一定存在。

第三章 工作区: Agent 长期状态放在哪里

工作区结构示意。本图为基于仓库文档与代码入口重绘的解释性示意图。

3.1 工作区是 JiuwenSwarm 的大脑文件夹

JiuwenSwarm 的运行状态不是凭空存在的。它会落到工作区中。文档将 workspace 描述为 Agent 记忆、技能、会话数据和可配置心跳任务的运行目录。源码模式和 wheel 安装模式的初始位置不同，但安装模式下常见路径是:

```
~/.jiuwenswarm/workspace/
```

在新的布局中，Agent 工作区通常位于:

```
~/.jiuwenswarm/agent/workspace/
```

不同版本之间可能存在历史路径迁移，因此升级前应该备份旧目录。

3.2 工作区布局

一个典型工作区可以理解为以下结构:

```
workspace/
├── AGENT.md
├── HEARTBEAT.md
├── IDENTITY.md
├── SOUL.md
├── USER.md
├── MEMORY.md
├── memory/
│   ├── YYYY-MM-DD.md
│   ├── messages.json
│   └── memory.db
├── skills/
│   ├── _marketplace/
│   └── <skill-name>/
└── session/
    ├── sess_<id>/
    │   ├── todo.md
    │   └── outputs...
    └── heartbeat_<id>/
```

这里最重要的是五类文件:

- `SOUL.md` : 系统提示人格。
- `MEMORY.md` : 长期记忆。
- `USER.md` : 用户画像。
- `HEARTBEAT.md` : 心跳时可执行的主动任务。
- `skills/<skill-name>/SKILL.md` : Skill 的行为说明。

3.3 为什么 Markdown 是好选择

JiuwenSwarm 把很多长期状态放在 Markdown 文件里，而不是只放数据库。这有三个好处:

1. 可读: 用户可以直接打开、审计和修改。
2. 可迁移: 备份和版本管理更简单。
3. 可参与推理: Agent 可以把文件内容作为上下文读取。

数据库仍然重要。比如记忆检索需要 ChromaDB、SQLite FTS、向量索引等结构化能力。但源头内容保留为 Markdown, 更符合个人知识库和可解释 Agent 的方向。

3.4 会话与任务产物

每个会话会有独立目录。任务规划工具会把 todo 写入 `workspace/session/{session_id}/todo.md` , 这样长任务可以被拆解、插入、完成和查询。对于需要持续数小时甚至数天的任务，文件化的会话状态比单纯依赖模型上下文稳定得多。

3.5 初始化脚本做了什么

`init_workspace.py` 的注释说明了初始化脚本的职责: 默认根目录为 `~/.jiuwenswarm`，支持 `JIUWENSWARM_DATA_DIR` 覆盖；运行时会询问语言偏好，复制 `config.yaml`，`builtin_rules.yaml`，`.env.template` 和 Agent 模板；还支持 `--name` 创建命名实例。

因此， `jiuwenswarm-init` 不只是建目录，它决定了你的 Agent 后续把配置、记忆、技能和会话文件放在哪里。对生产环境来说，这一步应该纳入部署脚本，而不是手工随意执行。

章末延伸与实践：为什么工作区比“安装目录”更重要

安装目录保存程序，工作区保存你的 Agent 生命史。配置、记忆、Skill、会话、任务、日志和心跳文件都围绕工作区展开。换一台机器迁移 JiuwenSwarm 时，真正需要重点保护的不是 Python 包，而是工作区中的用户数据。升级时也是一样：包可以重装，记忆和自定义 Skill 丢了就很难恢复。

工作区最好按“事实源”和“派生产物”区分。`config.yaml`、`.env`、`MEMORY.md`、`USER.md`、`SKILL.md`、`HEARTBEAT.md` 属于事实源；日志、向量索引、压缩消息、会话临时文件更像派生产物。备份策略应优先覆盖事实源，再根据需要保留派生产物。尤其是 `agent/skills` 与 `agent/memory`，这是长期使用后最有价值的部分。

另一个容易忽视的问题是路径迁移。文档和代码中可能出现旧路径、新路径和兼容路径。遇到“找不到记忆”或“Skill 没加载”时，不要只看一个目录，要检查当前版本实际使用的 workspace 根路径，以及是否存在历史目录残留。升级前用 `tree` 或文件管理器拍一份目录快照，会让回滚容易很多。

工作区管理建议

- 把 `.env` 和 `config.yaml` 当成敏感配置文件，不要直接上传到公共仓库。
- 自定义 Skill 建议单独保留 Git 仓库，工作区中安装的是运行副本。
- 记忆文件建议定期备份，但包含隐私内容时要加密保存。
- 多实例时不要共用同一个 workspace，除非你非常确定状态隔离不重要。

排查口诀

如果 Agent “忘了东西”，看 memory；如果“不会做以前会做的事”，看 skills；如果“同一入口会话错乱”，看 session；如果“启动行为变化”，看 config；如果“明明配置了但不生效”，看当前进程加载的是哪个 workspace 和哪个 `.env`。

第四章 进程架构: AgentServer 与 Gateway 的分离

AgentServer 与 Gateway 拆分架构。本图为基于仓库文档与代码入口重绘的解释性示意图。

4.1 为什么要拆进程

JiuwenSwarm 不是所有东西都塞进一个 Flask 或 FastAPI 服务。代码中有三个关键入口:

- `jiuwenswarm-app`: 一键编排 AgentServer 与 Gateway 两个子进程。
- `jiuwenswarm-agentserver`: 单独启动 Agent 执行侧。
- `jiuwenswarm-gateway`: 单独启动通道接入侧。

这种拆分的价值在于: Agent 执行和通道接入是两类不同负载。AgentServer 更关注模型、工具、Team、扩展和运行时; Gateway 更关注 WebSocket、通道消息、心跳、定时任务和路由。

4.2 AgentServer 的职责

`app_agentserver.py` 的注释很清楚: 该进程启动 Agent runtime 和 AgentWebSocketServer, Gateway 应该单独启动并连接到这个 WebSocket 服务。它们共享同一个用户工作区。

AgentServer 启动过程中会:

1. 确认工作区存在，必要时初始化或迁移。
2. 加载 `.env`。

3. 初始化扩展系统。
4. 启动 Agent WebSocket Server。
5. 对分布式 Team 场景启动 teammate bootstrap daemon。

这说明 AgentServer 是执行心脏，不负责面对每个外部聊天平台的细节。

4.3 Gateway 的职责

`app_gateway.py` 的顶部注释把 Gateway 的职责概括为四件事:

- 启动 Gateway MessageHandler 与 ChannelManager。
- 启动 WebChannel WebSocket server。
- 启动 Heartbeat service。
- 启动 Cron scheduler service。

同时，Gateway 会连接本地或远程 AgentServer WebSocket 端点。这使它天然适合做多通道入口、路由和协议转换。

4.4 一键启动做了什么

`app.py` 会先启动 AgentServer 子进程，短暂等待后再启动 Gateway 子进程。若启动 Gateway 失败，会终止 AgentServer。退出时也会统一终止两个子进程。

可以把它理解为:

```
jiuwenswarm-app
  ├── python -m jiuwenswarm.server.app_agentserver
  └── python -m jiuwenswarm.gateway.app_gateway
```

这是一种开发和本地部署都比较友好的结构: 用户可以用一个命令启动，也可以在高级部署中分别管理两个进程。

4.5 架构示意

4.6 架构带来的部署选择

你可以有三种部署方式:

部署方式	适用场景	特点
一键本地启动	个人使用、演示	简单，默认路径和端口即可
分进程启动	开发、调试、服务管理	AgentServer 与 Gateway 可单独观察日志
多机或多实例	团队、生产隔离、通道聚合	需要规划端口、工作区、数据库和权限

章末延伸与实践：分离 Gateway 与 AgentServer 的意义

Gateway 与 AgentServer 的分离，是 JiuwenSwarm 走向多通道和可扩展部署的基础。Gateway 关注“消息从哪里来、属于哪个会话、是否是控制命令、应该送到哪里”；AgentServer 关注“拿到标准化请求后如何执行模型、工具、记忆、Skill 和 Team”。如果这两件事混在一个入口里，每新增一个通道都会把执行逻辑和平台适配绑在一起。

拆分后，系统可以更容易支持远程 AgentServer、多个通道共用一个执行核心、以及在 Gateway 侧增加 Cron/Heartbeat 等主动触发。比如飞书消息、小艺消息和 Web 消息都可以由 Gateway 归一

化；AgentServer 不需要知道它们来自哪里，只需要处理统一字段中的文本、文件、会话、模式和参数。

这也解释了为什么很多故障要先分层定位。Web 页面打不开，不一定是 AgentServer 问题；模型不响应，不一定是 Gateway 问题；Slash 命令不生效，可能是 Gateway 拦截逻辑而不是 Agent prompt；定时任务触发但没结果，可能是 Gateway 调度成功但 AgentServer 连接失败。

日志排查顺序

1. 看启动日志：AgentServer 是否 ready，Gateway 是否成功连接 AgentServer。
2. 看入口日志：Web/IM/TUI 消息是否到达 Gateway。
3. 看转发日志：Gateway 是否生成正确 session_id、channel_id 和 req_method。
4. 看执行日志：AgentServer 是否调用模型、工具、记忆和 Skill。
5. 看权限日志：工具是否被 deny 或 ask 阶段阻塞。

架构取舍

单进程部署更易试用，拆分部署更适合生产。个人用户可以先用 `jiuwenswarm-start` 一键启动；集成者可以分别启动 AgentServer 和 Gateway，以便将 Gateway 放在靠近通道的位置，把 AgentServer 放在更安全、更稳定的执行环境中。

第五章 配置系统: 从默认模型到多模态能力

配置系统地图。本图为基于仓库文档与代码入口重绘的解释性示意图。

5.1 配置入口

Web UI 的 Configuration 页面用于配置模型、Embedding、第三方服务、自进化、上下文压缩和工具权限等。文档强调模型配置是必需项，其他配置大多是可选增强。

配置的思路可以归纳为一句话:

> 先让默认文本模型稳定可用，再按需求打开记忆检索、多模态、搜索、GitHub、SkillNet、浏览器和权限策略。

5.2 默认模型

默认模型承担最核心的任务: 多轮对话、任务规划、工具调用和一般推理。它必须支持函数调用或等价的工具调用能力，否则 Agent 的工具生态难以稳定工作。

配置例子:

```
api_base: https://api.openai.com/v1
api_key: sk-your-key
model: gpt-4o
model_provider: OpenAI
```

对 OpenAI-compatible API, 最容易出错的是 `api_base`: 写到 `/v1` 即可，不要追加完整的 chat completions 路径。

5.3 多模型与别名

配置文档支持 Model List, 每个模型可以有 `model_name`, `alias`, `api_base`, `api_key`, `model_provider`, `temperature` 等字段。`alias` 的意义很大: 它让用户能用较稳定的名字切换模型，而不用记住底层服务的具体模型 ID。

建议为不同用途设置别名:

别名	用途	配置建议
`daily`	日常对话	低成本、高响应速度模型
`reason`	复杂规划	更强推理模型
`vision`	图片理解	配置视觉模型
`code`	编程任务	工具调用和代码能力稳定的模型

5.4 Embedding 配置

Embedding 是记忆系统的关键。未配置时可以有 mock 或基础检索；配置真实 Embedding 后，记忆召回会更准确。

典型配置项:

```
embed_api_base: https://api.siliconflow.cn/v1
embed_api_key: sk-your-key
embed_model: BAAI/bge-large-zh-v1.5
```

中文用户建议优先选择中文优化过的 embedding 模型，因为记忆内容很可能同时包含中文任务、代码片段、项目名称和口语化表达。

5.5 多模态模型

配置文档把模型类型拆成默认文本模型、视频模型、音频模型、视觉模型和图像生成模型。多模态配置不应一次性全开。更推荐按照真实任务逐个打开:

- 需要截图、表格识别、图片问答时配置 Vision。

- 需要语音转写时配置 Audio。
- 需要会议录像或视频理解时配置 Video。
- 需要出图时配置 Image Generation。

不要把多模态能力视作装饰。对个人 AI 管家而言，图片、音频和文件往往是任务入口，例如「把这张发票录入表格」「分析这段会议录音」「把这张白板图变成执行计划」。

插图：配置不只是表单，而是决定 Agent 能力边界的控制台。

✦ 美术插图：配置面板决定模型、记忆、通道和权限的组合方式。

5.6 第三方服务配置

JiuwenSwarm 的配置文档列出了 Jina、Bocha、Serper、Perplexity、GitHub、TeamSkillsHub 等可选服务。它们不是系统启动的硬依赖，但会影响搜索、网页读取、GitHub 访问、SkillNet 等高级功能。

建议按以下顺序配置:

1. 模型 API。
2. Embedding API。
3. 搜索服务。
4. GitHub Token。
5. Skill 市场相关 Token。
6. 浏览器和 MCP。

5.7 自进化、压缩和权限开关

三个值得单独理解的开关:

开关	默认倾向	作用
`evolution.enabled`	默认关闭	允许系统记录 Skill 改进建议
`context_engine.enabled`	默认开启	长对话时压缩和卸载上下文
`permissions.enabled`	文档中默认关闭	启用工具调用权限检查

生产环境尤其建议打开权限系统，并为高危工具配置 ask 或 deny。个人本地环境也不应长期使用完全放开的工具策略。

章末延伸与实践：配置不是表单，而是能力开关系统

JiuwenSwarm 的配置分为“必须有”和“按场景启用”两类。默认模型是必须有的，因为没有它，Agent 无法完成对话、规划和工具调用。多模态模型、Embedding、外接记忆、搜索服务、GitHub Token、Browser MCP、通道凭证、权限规则和遥测则按场景开启。把所有配置一次性填满，反而会增加故障面。

模型配置要特别注意 provider 适配。不同模型服务可能兼容 OpenAI API，但 Provider 字段仍会影响请求格式、工具调用格式和错误处理。API Base 也要使用服务根地址，不要把具体接口路径写进去。配置多个模型时，alias 很有用：它让你在 UI 或命令中用稳定名称切换，而不必记住完整模型 ID。

Embedding 配置决定记忆检索质量。没有 Embedding Key 时，系统仍可能使用基础检索或 mock provider，但语义召回会受影响。若你打算长期使用记忆系统，Embedding 不应被视为可有可无的装饰，而是记忆“能不能找回来”的核心依赖。

建议配置顺序

阶段	配置项	目标
最小可用	默认模型四要素	能聊天、能规划、能调用基础工具
记忆增强	Embedding 与 memory.engine	能跨会话检索事实和经验
工具增强	MCP、Browser、搜索服务	能操作网页和外部工具
通道接入	Feishu/Xiaoyi/DingTalk 等	能从工作入口触发 Agent
生产防护	permissions、owner_scopes、日志	限制风险并便于审计

配置维护建议

不要把所有密钥直接写入 `config.yaml`，优先用 `.env` 和环境变量占位。团队部署时，不同实例应使用不同配置根目录，避免 dev/prod API Key 混用。修改权限规则后，要用低风险工具先试跑，例如 `pwd`、`ls`、只读文件访问，再逐步开放写入和 shell 操作。

第六章 模式系统: Agent, Code, Team

运行模式选择矩阵。本图为基于仓库文档与代码入口重绘的解释性示意图。

6.1 为什么需要模式

同一个 Agent 不应在所有场景下使用同一种行为策略。日常问答、长任务规划、代码执行、多人协同，对工具、记忆和权限的需求都不同。JiuwenSwarm 用模式系统把这些差异显式化。

6.2 模式概览

文档列出的主要模式如下:

模式	代码	典型用途
Agent Plan	`agent.plan`	默认模式，强调推理、规划和主动记忆
Agent Fast	`agent.fast`	快速响应，使用被动记忆
Code Plan	`code.plan`	编程任务的规划阶段
Code Normal	`code.normal`	编程任务的执行阶段
Team	`team`	多智能体协作

6.3 切换命令

在支持的通道中可以使用:

```
/mode agent
/mode code
/mode team
/mode agent.plan
/mode agent.fast
/mode code.plan
/mode code.normal
```

还可以使用 `/switch` 在同类模式中切换子模式:

```
/switch plan
/switch fast
/switch normal
```

6.4 模式背后的差异

模式的本质差异不是名字，而是三件事:

1. 工具集合不同。
2. 记忆策略不同。
3. 安全 rails 不同。

例如 Code 模式会启用 Coding Memory, 并配置 FileSystemRail、SkillUseRail、LspRail 等 rails。Agent Plan 模式则更偏向主动记忆和任务拆解。

6.5 如何选模式

你的任务	推荐模式	原因
写一封邮件、总结文本、安排日程	`agent.fast` 或 `agent.plan`	工具压力低，重在理解和表达
复杂任务拆解、跨多步执行	`agent.plan`	需要 todo 和上下文管理
修改代码、调试错误、生成脚本	`code.normal`	需要文件和代码上下文
先做技术方案再执行	`code.plan` 后切 `code.normal`	分离规划和落地
多角色协作、拆任务给成员	`team`	需要 TeamManager 和共享状态

6.6 每个通道可以有默认模式

配置中可以为通道设置默认模式，例如 Web 通道默认 `agent.plan`。这很适合多入口场景: Web 入口做规划，TUI 入口做开发，飞书入口做轻量沟通，小艺入口做移动端问答。

章末延伸与实践：模式决定“同一个请求会被怎样执行”

模式系统的价值在于让同一个 Agent 根据任务类型改变行为边界。`agent.plan` 适合复杂任务，因为它倾向于先拆解、再执行，并主动使用记忆。`agent.fast` 适合快速问答和轻量任务，因为它减少主动记忆和过度规划。`code.plan` 适合做代码方案和风险分析，`code.normal` 更适合真正修改文件、运行命令和使用 Coding Memory。`team` 则把单 Agent 扩展成 Leader 与成员协作。

不要把模式当成“回答风格”开关。它会影响工具集合、记忆策略、安全 rails 和迭代方式。例如 Code 模式会引入文件系统安全、Skill 使用约束和 LSP 辅助；Team 模式则需要 Team 配置、成员身份、存储和生命周期配合。错误模式下执行任务，可能导致 Agent 要么太保守、要么过度行动。

模式选择例子

- “帮我解释这个概念”适合 `agent.fast`。
- “帮我规划一个三周迁移方案”适合 `agent.plan`。
- “分析这个仓库的改造方案，先不要动代码”适合 `code.plan`。
- “修复这个 bug 并跑测试”适合 `code.normal`。
- “把一个大型项目分成多个子任务并并行推进”适合 `team`。

模式切换注意事项

模式切换通常会取消当前运行任务或影响后续消息的处理方式。长任务执行中不要频繁切换模式；如果要探索另一种方案，优先使用 `/branch` 分叉会话，而不是在原会话里反复切换。通道默认模式也应按入口设定：Web 可以默认 `agent.plan`，TUI 开发场景可以更偏 Code，群聊入口则要更保守。

第七章 Slash 命令: 控制会话、模式和上下文

Slash 命令解析边界。本图为基于仓库文档与代码入口重绘的解释性示意图。

7.1 命令不进入 Agent

CLI 文档强调: 特殊前缀命令由 Gateway 的 MessageHandler 解析，不会被发送给 Agent。这一点很关键，因为 `/mode`、`/new_session` 等命令是控制平面操作，而不是普通聊天内容。

支持控制命令的 IM 通道包括飞书、小艺、钉钉、WhatsApp 和微信等。

7.2 新会话

```
/new_session
```

它会为当前 channel 生成新的 session_id, 并取消当前会话中的运行任务。适合以下场景:

- 换话题，不想让旧上下文干扰。
- 任务已经跑偏，需要重新开始。
- 当前会话包含临时敏感内容。

7.3 分支会话

```
/branch
/branch fix login issue
```

分支会话适合探索不同方案。它不是简单清空上下文，而是从当前会话 fork 出新会话。对于代码修复、方案比较、文案 A/B 版本，这个能力很有用。

7.4 回滚会话

```
/rewind 3
/rewind confirm 3
/rewind cancel
```

回滚会删除指定轮次及其之后的对话记录。文档提醒: 回滚不影响已经手工编辑的文件，也不撤销 bash 命令造成的外部副作用。因此它是会话层回滚，不是系统级事务回滚。

7.5 主动压缩

TUI 支持:

```
/compact
```

它会主动触发上下文压缩。返回值可能是 `busy`，`compressed` 或 `noop`。这对超长会话很有用，尤其是工具输出堆积、日志太长、对话已经接近模型上下文窗口时。

7.6 Skill 列表

```
/skills list
```

用于查看可用技能。日常使用中，建议在安装新 Skill 后先列出一次，确认它已被系统识别。

章末延伸与实践：Slash 命令是“控制面”，不是普通聊天内容

Slash 命令的作用是绕开自然语言不确定性，用确定的语法修改会话、模式、模型、MCP、记忆和上下文。它们不应该全部进入 Agent 对话，否则 `/new_session` 这样的命令可能会被模型当作普通文本解释，而不是真正新建会话。

因此，Slash 命令存在解析边界。TUI 有自己的本地命令，比如清屏、换主题、查看状态和工作区授权；Gateway 处理通道控制命令，比如新会话、切模式、分支、回退；部分命令会进入 Agent 或后端能力，比如 Skill 管理、MCP 管理、上下文压缩、Cron 管理。理解解析位置，才能知道命令失败时应该看前端、Gateway 还是 AgentServer。

实战建议

先掌握四个高频命令：`/new_session` 用于清空当前入口的会话上下文，`/mode` 用于切换 Agent/Code/Team，`/branch` 用于从当前对话分叉探索，`/compact` 用于在长上下文中压缩历史。对于开发者，再进一步掌握 `/mcp`、`/model`、`/memory`、`/diff` 和 `/init`。

命令治理风险

文档中已经指出，Slash 命令如果在 Gateway、IM pipeline、TUI 多处重复定义，容易出现语义漂移。二次开发时不要在新通道里私自复制一套命令列表；应复用统一注册表或至少把解析规则集中维护。否则同一个 `/mode code` 在 Web 与飞书里的行为可能不一致，用户很难理解。

使用检查表

- 不确定当前会话是否污染时，先 `/new_session` 或 `/branch`。
- 不确定当前模式时，先查看状态或明确 `/mode agent.plan`。
- 长任务上下文变大时，用 `/compact`，而不是直接清空所有记忆。
- 涉及文件改动时，用 `/diff` 或版本控制确认变更。

第八章 任务规划: 让长任务不中途丢失

任务规划状态机。本图为基于仓库文档与代码入口重绘的解释性示意图。

8.1 长任务的真实问题

真实工作不是单轮问答。用户可能先要求整理 12 月发票，执行中途又追加 1 月发票，再要求合并成一份汇总邮件。普通聊天式 Agent 很容易在这种任务中丢目标、重复执行或忘记中间状态。

JiuwenSwarm 的任务规划机制用结构化 todo 工具来解决这个问题。

8.2 TodoToolkit

任务规划文档列出五个核心工具:

工具	作用
`todo_create`	创建初始任务列表，已有列表时会失败
`todo_insert`	在指定位置插入任务，没有列表时可创建
`todo_complete`	标记任务完成，并可记录结果
`todo_remove`	删除任务并重新编号
`todo_list`	查看全部任务和状态

任务状态包括 `waiting`，`running`，`completed`，`cancelled`。

8.3 为什么任务写入 Markdown

Todo 被存储在:

```
workspace/session/{session_id}/todo.md
```

这意味着任务状态不是只存在模型记忆里。它可以被读取、检查、持久化，并按会话隔离。对长任务来说，这比让模型「自己记住」可靠得多。

8.4 一个典型工作流

```
用户：帮我整理这个项目的发布准备工作。
Agent：创建 todo 列表。
Agent：完成第一项，记录结果。
用户：先插入一项：检查安全权限配置。
Agent：todo_insert 插入新任务。
Agent：继续执行并逐项完成。
用户：现在列一下状态。
Agent：todo_list 返回进度。
```

这种结构降低了两个风险: 一是任务目标丢失，二是执行过程不可观察。

8.5 使用建议

- 对超过三步的任务，优先开启任务规划。
- 让 Agent 在关键节点输出 todo 状态。
- 插入新任务时说清楚优先级。
- 不要把 todo 当成最终报告；它是执行状态，不是总结文档。

章末延伸与实践：任务规划解决的是“长任务失忆”问题

长任务最常见的失败不是模型完全不会做，而是在执行过程中丢目标、丢约束、丢中间结果。用户插入新需求、工具返回大量内容、上下文接近上限、步骤顺序变化，都会让 Agent 偏离原计划。任务规划通过 `todo.md` 把计划显式写下来，让进度变成可读、可改、可恢复的状态。

`todo_create` 适合一开始创建任务列表； `todo_insert` 适合用户临时插入优先级更高的事项； `todo_complete` 不只是打勾，还应该写入结果摘要； `todo_remove` 用于取消不再需要的步骤； `todo_list` 用于用户随时检查当前状态。关键是：每个完成项都要有“完成结果”，否则 todo 只是一串空洞动作。

例子：报告生成任务

用户要求“基于仓库写一本电子书”，合理的任务规划不是直接写，而是拆成：读取 README 与文档目录、抽取主题地图、确定读者和章节、收集配图素材、撰写章节、生成 PDF/EPUB、渲染检查、

修复版式、交付打包。若用户中途说“章节太薄、缺配图”，就应该插入“扩写每章”和“补图”任务，而不是推倒重来。

任务规划质量标准

- 每个任务粒度应能在一次工具调用或一小段执行中完成。
- 任务名称要表达结果，而不是动作，例如“完成配置检查表”优于“继续写”。
- 完成结果要记录关键产物路径、判断依据和未解决问题。
- 用户新增需求时用插入，不要简单覆盖原计划。

常见误区

不要把 todo 当成给用户看的装饰。任务规划真正的价值在于让 Agent 自己也能读回当前计划，并在上下文变长后仍然知道下一步是什么。也不要把所有琐事都拆成 todo；过细会让管理成本超过收益。

第九章 记忆系统: 从会话记忆到经验沉淀

记忆写入与检索管线。本图为基于仓库文档与代码入口重绘的解释性示意图。

9.1 记忆的四个层次

JiuwenSwarm 的记忆不是单一概念。至少可以分为四层:

1. 内置长期记忆: 保存偏好、稳定事实、每日记录。
2. 外部记忆: 对接 OpenJiuwen LTM、Mem0、OpenViking 或插件。
3. 任务记忆: 保存任务经验，避免重复犯错。
4. Coding Memory: Code 模式下专用的代码上下文和工程经验。

9.2 内置记忆文件

内置记忆常见布局:

```
{workspace_dir}/memory
├── MEMORY.md
├── USER.md
└── YYYY-MM-DD.md
```

`MEMORY.md` 适合放长期决策、偏好和稳定事实。`USER.md` 适合放用户画像。每日 Markdown 文件适合放当天运行上下文。

9.3 检索: BM25 与向量混合

文档说明内置记忆默认可以使用 BM25 全文检索；配置 Embedding 后可结合向量和 BM25 做混合召回。技术栈中还出现 SQLite FTS5、sqlite-vec、embedding cache 等组件。

这说明 JiuwenSwarm 的记忆设计不是简单的「把历史对话全部塞回提示词」。它更接近知识库检索:

```
Query -> 关键词检索 + 向量检索 -> 合并排序 -> 读取相关片段 -> 放入上下文
```

✦ 美术插图：记忆不是堆积，而是持续种植、索引、修剪和召回。

9.4 Dreaming: 睡眠期记忆巩固

Dreaming 是很有想象力的设计。它在空闲时间扫描过去会话，用 LLM 抽取值得长期保留的内容，再写入持久记忆。Agent 模式和 Code 模式共享思路，但输出目标不同:

模式	抽取目标	输出
Agent	用户偏好、背景、兴趣	`DREAMING.md`
Code	调试根因、API 边界、设计决策	`coding_memory/consolidated_{hash}.md`

与实时记忆相比，Dreaming 的优势是「事后反思」。实时记忆很容易受当下对话噪音影响；睡眠期巩固可以从完整会话中提炼更稳定的经验。

9.5 Task Memory

Task Memory 是经验系统。它提供三类工具:

- `experience_retrieve`：在任务开始前检索相关经验。
- `experience_learn`：在任务结束前记录关键发现。
- `experience_clear`：清空经验数据，只应在用户明确要求时使用。

推荐模式是「任务开始前检索，任务结束前学习」。这让 Agent 不只是完成任务，还能形成工程经验。

9.6 Coding Memory

Coding Memory 在 Code 模式中自动启用，提供:

- `coding_memory_read`
- `coding_memory_write`
- `coding_memory_edit`

它适合记录架构约定、Bug 根因、调试路径、API 特性和项目风格。和一般记忆不同，Coding Memory 的内容天然更结构化，也更依赖代码上下文。

9.7 Team Memory

在 Team 模式下，记忆又分为个人记忆和团队记忆。持久团队会在每轮结束后由 Leader 派生 extractor agent, 从任务记录和团队消息中抽取值得保留的信息，写入 `TEAM_MEMORY.md`。常见标签包括 `[decision]`，`[lesson]`，`[member]`，`[context]`。

这让团队协作不只是多 Agent 并行执行，而是能形成组织记忆。

9.8 记忆使用原则

少写事实，多写稳定事实。 临时状态应留在会话，长期偏好才进长期记忆。

先检索，再执行。 对重复性任务，先查经验记忆可以避免重踩坑。

定期备份。 记忆文件是个人数据资产，升级前务必备份。

为团队隔离。 不同团队、不同成员要有清晰路径和索引隔离。

章末延伸与实践：记忆系统要解决“该记什么”和“何时读回”

记忆不是把聊天记录永久保存。好的 Agent 记忆应该只保存对未来任务有价值的事实、偏好、决策和经验。比如“用户喜欢中文技术教程”“项目使用 Python 3.12”“某个接口超时要重试”值得记；一次临时口误、一次无关闲聊、密钥和身份证号不应该记。

JiuwenSwarm 的内置记忆采用 Markdown 文件作为事实源，索引层负责检索。这个设计很重要：Markdown 可读、可备份、可审查；索引可以重建。相比把所有东西塞进不可读数据库，文本事实源更符合个人数据主权。Embedding 和 BM25 的混合检索，则让 Agent 不必每次全量读取记忆。

外接记忆提供更强的扩展能力。`memory.engine` 可以选择 builtin、external、both 或 none；外接 provider 可选 OpenJiuwen、Mem0、OpenViking 或插件。生产中要根据隐私、成本和召回质量选择。个人隐私强的场景更适合本地内置记忆；需要多设备和云端事实抽取时，再考虑外接 provider。

记忆写入策略

类型	建议目标	示例
稳定偏好	USER.md 或长期记忆	用户偏好中文、喜欢表格总结
项目事实	MEMORY.md	项目主分支、部署方式、依赖版本
当日过程	YYYY-MM-DD.md	今天完成了配置迁移
代码经验	coding_memory	某测试 flaky 的根因
任务经验	task memory	生成电子书时先渲染检查再交付

安全建议

启用 forbidden memory 规则或等价约束，明确禁止记住密码、API Key、Token、身份证号等敏感信息。群聊数字分身开启记忆时尤其要谨慎，因为群聊内容可能包含他人信息，不能默认都进入长期记忆。

第十章 Skill 系统: 把能力封装成可安装模块

Skill 包结构。本图为基于仓库文档与代码入口重绘的解释性示意图。

Purpose

将仓库文档、代码入口和使用路径整理成可交付 PDF/EPUB/Markdown。

Trigger conditions

用户提供 GitHub/GitCode 仓库链接，并要求写电子书、教程、白皮书或手册。

Workflow

1. 读取 README、安装文档、配置文档和关键架构文档。
2. 建立章节大纲和事实来源表。
3. 补足必要配图：架构图、流程图、配置图、权限图。
4. 生成 PDF/EPUB/Markdown，并渲染检查。

Verification

检查目录、页码、图片、代码块、来源列表和下载文件。

Skill 设计取舍

Skill 越具体，触发越准确，但复用范围越窄；Skill 越泛化，复用范围越广，但容易行为不稳定。建议把高频、明确、有固定输出格式的任务做成 Skill；把一次性探索任务留给普通 Agent。涉及危险工具的 Skill 要把禁止操作写在最前面，例如不得删除原始文件、不得上传密钥、不得绕过权限。

10.1 什么是 Skill

Skill 文档把 Skill 定义为扩展 JiuwenSwarm 特定能力的模块，可以理解为可安装、可管理、可复用的能力包。它类似手机 App 对手机能力的扩展：Agent 有基础对话、文件、搜索和代码能力；Skill 则把某一类复杂流程封装起来。

一个典型 Skill 目录至少包含 `SKILL.md`，也可能包含 `references/`, `scripts/`, `prompts/` 等。

10.2 Skill 解决什么问题

没有 Skill 时，用户需要一步步告诉 Agent 怎么做。比如创建 PR、处理评审意见、生成 PPT、调用平台 API，都需要大量人工指令。Skill 的价值在于把流程、约束、工具调用顺序、失败处理和输出格式固化为可复用单元。

一个成熟 Skill 至少应该说明:

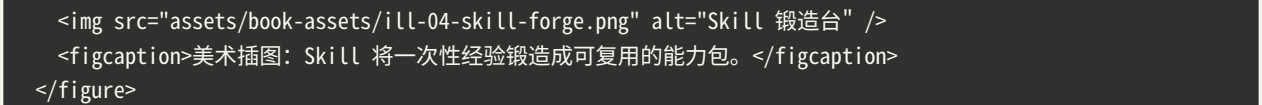
- 何时触发。
- 需要哪些输入。
- 可以调用哪些工具。
- 执行步骤是什么。
- 如何验证结果。
- 失败时如何处理。
- 最终输出格式是什么。

10.3 Skill 来源

JiuwenSwarm 支持多种 Skill 来源:

来源	说明
内置 Skill	随产品发布的核心技能资源
SkillNet	基于 GitHub 技能仓库的在线搜索与安装
ClawHub	OpenClaw 生态的 Skill 商店
Marketplace	第三方或社区源
本地导入	用户自写或调试中的 Skill

安全上要记住一点：Skill 可能涉及文件修改、命令执行和外部服务调用。安装前应检查来源、描述和允许工具。

<figure class="art-figure">
  
  <figcaption>美术插图：Skill 将一次性经验锻造成可复用的能力包。</figcaption>
</figure>

10.4 本地 Skill 结构

最小结构:

```text
my-skill/
 ├── SKILL.md
 ├── references/
 └── scripts/
```

`SKILL.md` 是核心。它不是普通说明书，而是 Agent 执行时要读取的操作规范。写得越具体，行为越稳定。

10.5 好 Skill 的写法

一个可维护 Skill 可以按以下结构编写:

```
---
name: example-skill
description: when to use this skill
---

## 章末延伸与实践：Skill 是“可复用工作流”，不是提示词片段

一个好的 Skill 应该让 Agent 少猜、少临场发挥、多复用确定流程。`SKILL.md` 至少要说明什么时候触发、输入需要什么、执行步骤是什么、可以使用哪些工具、哪些事情禁止做、如何验证结果、失败时如何降级。只写“帮用户生成报告”这种描述，几乎等于没有 Skill。

Skill 的目录结构也反映了工程化思路。`references/` 放长资料，避免把所有背景塞进主说明；`scripts/` 放可重复执行的脚本，避免模型每次重写；`assets/` 放模板和示例；`evolutions.json` 保存运行中积累的改进。这样 Skill 才能从“提示词”变成“能力包”。

### 好 Skill 的写法模板

```markdown
---
name: repo-ebook-writer
description: 当用户要求基于一个开源仓库生成教程型电子书时使用
---

# Purpose

# Trigger conditions

# Inputs

# Workflow

# Verification

# Failure handling

# Output format
```

尤其要写清楚「不要做什么」。很多 Agent 失败不是因为不会做，而是因为边界不清。例如处理文件时要说明不要删除原文件，生成报告时要说明不要编造数据源。
```

10.6 Skill 管理

Web UI 中的 Skills 页面可以安装、查看、搜索、导入和卸载 Skill。文档还提到可以查看 Skill experience, 也就是自进化记录。这意味着 Skill 不只是静态包，也可以积累实际运行中的经验。

10.7 Skill 与模式的关系

Skill 是能力层，模式是行为策略层。两者叠加才决定 Agent 的实际行为。比如同一个 Git PR Skill, 在 Agent Plan 模式下可能先解释计划，在 Code Normal 模式下可能直接修改文件并提交。配置时应同时考虑 Skill 和 mode。

第十一章 Skill 自进化: 让错误变成下一次的经验

Skill 自进化闭环。本图为基于仓库文档与代码入口重绘的解释性示意图。

11.1 静态 Skill 的问题

多数 Agent 系统部署后，Skill 就被冻结了。工具报错只出现在日志里，用户纠正只留在聊天记录里，下一次调用 Skill 仍然会犯同样的错误。

JiuwenSwarm 的自进化机制试图解决这个问题: 把失败、纠正和改进信号记录成结构化 evolution，并在后续调用中合并给 Agent 使用。

11.2 核心组件

Skill 自进化文档列出四个关键组件:

组件	职责
`SkillCallOperator`	读取 `SKILL.md`，执行技能逻辑，加载 evolution 笔记
`SkillOptimizer`	接收信号，判断是否值得改进，调用 LLM 生成修改建议
`SkillEvolutionManager`	扫描、生成、写入 `evolutions.json`，可选固化到 `SKILL.md`
`SignalDetector`	基于规则检测失败和用户纠正，不依赖 LLM

11.3 什么算信号

常见信号有两类:

1. 执行失败: timeout, error, exception, permission denied, command not found 等。
2. 用户纠正: wrong, should be, actually, not that, 或中文等价表达。

这些信号会被归因到当前激活 Skill, 再转换成 troubleshooting 或 examples。

11.4 Evolution 流程

```
用户聊天 / 工具运行
        ↓
SignalDetector 检测失败或纠正
        ↓
SkillEvolutionManager.scan()
        ↓
SkillEvolutionManager.generate()
        ↓
evolutions.json
        ↓
可选 solidify 合并到 SKILL.md
```

11.5 `evolutions.json` 的意义

每个 Skill 可以有自己的 `evolutions.json`。它记录改进条目、来源、时间、上下文、修改内容和是否已应用。 `applied: false` 表示待固化， `applied: true` 表示已合并。

这是一种很实用的设计: 不直接让 LLM 改写 Skill 主文件，而是先积累候选经验。用户或维护者可以审核后再固化。

11.6 手动触发

可以使用:

```
/evolve <skill_name>
/evolve list
```

手动触发适合维护者调优 Skill。例如某个 Skill 连续几次在同一类 API timeout 上失败，可以触发 evolution 生成排障提示。

11.7 自进化的风险

自进化不是越多越好。需要注意:

- 错误信号可能是偶发网络问题，不一定应该改变 Skill。

- 用户纠正可能只针对一次特殊场景，不一定是通用规则。
- 自动生成的改进需要审核，避免把错误经验固化。
- 对高危工具相关 Skill, 进化不应绕过权限系统。

推荐策略是: 自动记录，人工固化；低风险 Skill 可以更自动，高风险 Skill 必须审核。

章末延伸与实践：自进化的核心是“把错误变成结构化经验”

传统 Agent 的错误往往停留在三处：终端日志、聊天抱怨、用户脑子里。下次执行同一任务时，模型未必知道上次错在哪里。JiuwenSwarm 的 Skill 自进化把错误、超时、权限拒绝、用户纠正等信号转成 evolution 记录，让 Skill 在后续调用时能读到这些经验。

这并不等于让系统自动随意改 Skill。更稳的做法是先记录，再审核，再固化。`evolutions.json` 可以保存待应用的建议；`SKILL.md` 是最终执行规范。生产中建议把 auto scan 和 manual evolve 分阶段使用：先手动 `/evolve` 观察记录质量，再决定是否开启自动扫描。

哪些信号值得进化

- 工具超时、连接失败、命令不存在、权限拒绝等可复现错误。
- 用户明确纠正：“不是这个目录”“应该用上海不是北京”“输出格式不对”。
- 同一个 Skill 多次产生重复瑕疵，例如总是漏掉来源列表。
- 某个 Provider 或平台有特殊限制，需要写入 Troubleshooting。

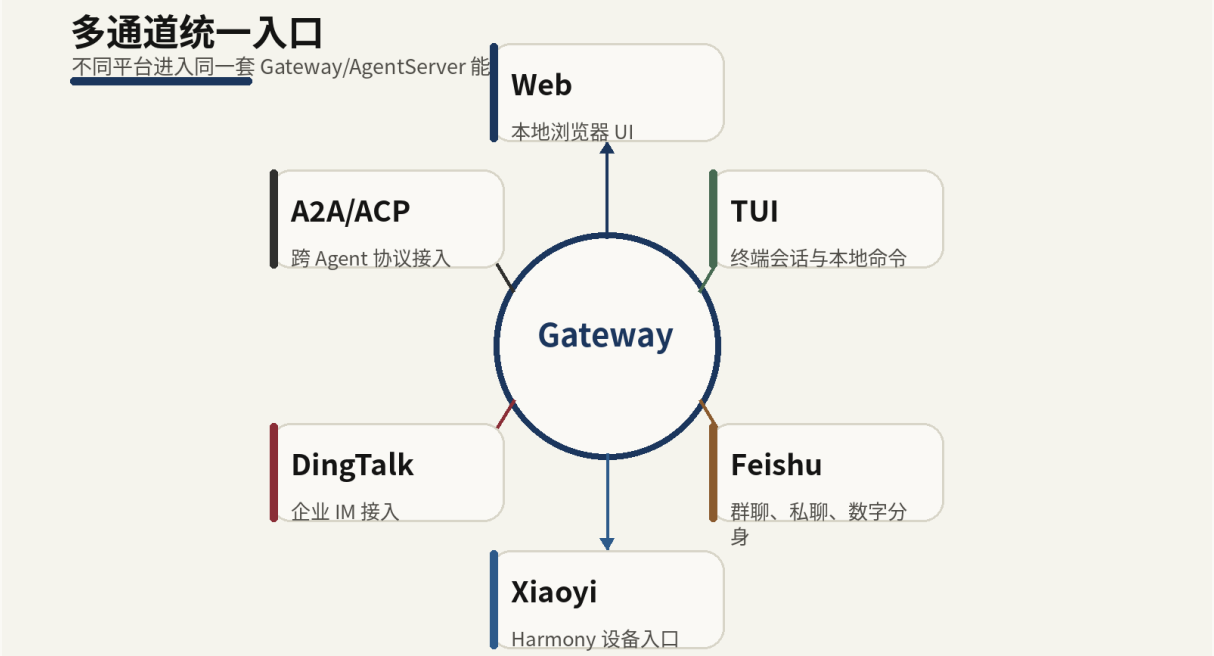
哪些不应该进化

一次性偏好不应直接固化为全局规则。用户临时要求“这次不要配图”，不代表以后所有电子书都不要配图。敏感信息也不应进入 evolution 内容。进化记录应该保存“规则”和“经验”，不是保存隐私和临时上下文。

审核建议

每隔一段时间检查 `evolutions.json`，合并重复项，删除低质量项，把稳定经验写入 `SKILL.md`。如果 Skill 已经因为进化记录变得臃肿，可以重写主流程，把零散经验整理成 Troubleshooting、Examples、Verification 三类。

第十二章 通道系统: 从 Web 到数字分身



多通道统一入口。本图为基于仓库文档与代码入口重绘的解释性示意图。

12.1 Channel 的本质

Channel 是 JiuwenSwarm 连接聊天平台的方式。它不是 UI 皮肤，而是把外部平台消息转换为内部 Agent 请求，再把 Agent 响应转换回平台格式的适配层。

文档明确提到 HarmonyOS 小艺、飞书等通道，并表示可支持更多平台。

12.2 Web 与 TUI

Web 是最直观的入口，适合配置、调试、查看状态、管理 Skill 和通道。TUI 则适合开发者、终端用户和需要快速切换工作区的场景。

建议的使用组合:

- Web: 配置模型、检查通道、管理 Skill、查看心跳和浏览器服务。
- TUI: 快速对话、代码任务、主动压缩、工作区路径切换。

12.3 小艺通道

小艺通道的路径大致是:

1. 在小艺开放平台创建 JiuwenSwarm 模式智能体。

2. 创建凭证，保存 AK/SK。
3. 配置白名单和调试用户。
4. 发布智能体。
5. 在 JiuwenSwarm 中绑定 AK、SK 和 agentId。
6. 通过 Web 或 Harmony 设备上的小艺 App 对话。

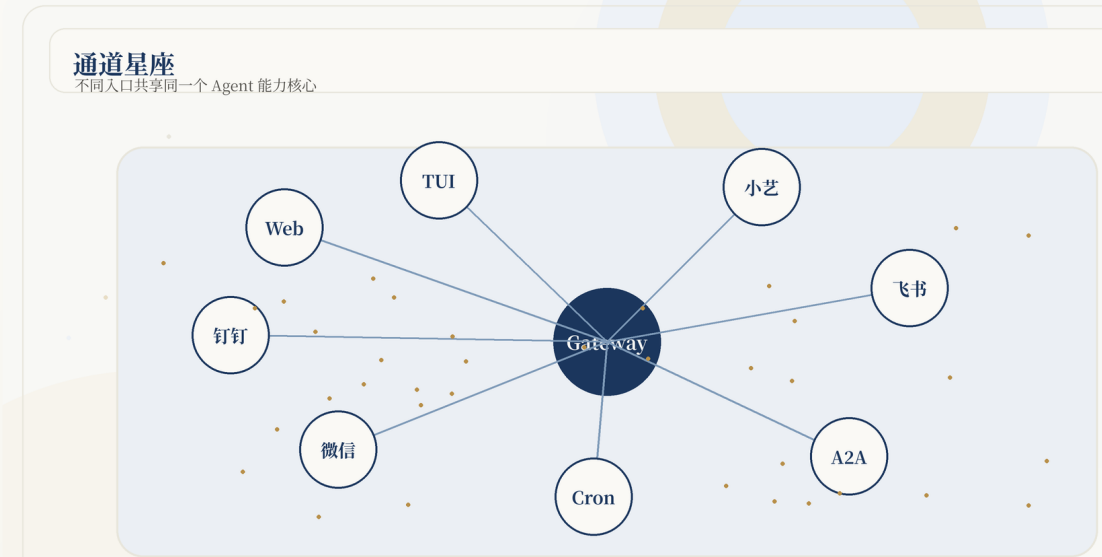
这个通道的战略意义在于移动端入口: 个人 AI 管家如果只能在浏览器里使用，主动性和可达性会明显受限。

12.4 飞书通道

飞书通道涉及更多企业集成步骤:

- 创建飞书自建应用。
- 添加机器人能力。
- 保存 App ID 和 App Secret。
- 开通消息与资源权限。
- 配置事件订阅。
- 发布应用。
- 可选加入群聊。
- 在 JiuwenSwarm 中启用飞书通道。

飞书通道适合把个人 Agent 放进团队沟通流。它也为后面的数字分身能力提供场景。



插图：不同通道像星座，Gateway 负责把它们映射到同一套会话和能力。

✦ 美术插图：多通道像一组星座，Gateway 是会话和能力的共同中心。

12.5 群数字分身

通道文档描述了 Group Digital Avatar: 在飞书和企业微信群聊中，机器人可以作为指定用户的数字分身，只处理与该用户相关的消息，并用第一人称回复。与个人待办和提醒相关的内容可以私信给用户，同时在群里给出简短确认。

这个能力很强，也很危险。因为群聊中的自动回复如果不加权限约束，很容易越权操作。因此文档要求为数字分身预配置工具和路径权限；在不能弹窗确认的群聊场景里，`ask` 会被降级为 `deny`。

12.6 多机器人与企业场景

飞书企业配置支持一个 JiuwenSwarm 实例服务多个飞书 App。每个 bot 是独立 channel，`channel_id` 类似 `feishu_enterprise:<app_id>`。这为多部门、多租户和多机器人管理提供了基础。

12.7 通道设计原则

1. 通道只做适配，不把业务逻辑写死在平台层。
2. 每个通道必须明确 session_id 和 user_id 的映射。
3. 群聊场景要默认更保守。
4. 不能确认的操作不应默认执行。
5. 对外部平台凭证要和模型 API Key 一样保护。

章末延伸与实践：通道接入的本质是“身份、会话和权限映射”

接一个通道不只是收发消息。每个平台都有自己的用户 ID、群聊 ID、消息 ID、文件上传方式、事件回调和权限模型。Gateway 的价值在于把这些平台差异映射成 JiuwenSwarm 内部可理解的 channel、session、message、params 和 files。

飞书、小艺、钉钉、Telegram、Discord、WhatsApp、微信、企业微信等通道的配置复杂度不同。个人使用可以从 Web/TUI 开始；企业协作通常先接飞书或企业微信；Harmony 生态用户可以接小艺；跨 Agent 场景再考虑 A2A/ACP。不要一开始把所有通道都打开，通道越多，凭证、白名单、日志和权限面越复杂。

数字分身为什么需要额外权限设计

群聊数字分身会自动判断哪些消息和代表用户相关，并可能以第一人称回复。这很强，也很危险。私聊里 Agent 可以在执行敏感操作前问用户；群聊分身往往不能逐步确认，因此 `ask` 可能需要降级为 `deny`。这就是 owner scopes 和工具白名单的重要性：你要提前定义群聊中允许它读什么、写什么、执行什么。

通道接入检查表

- 平台应用是否创建、发布并授权必要权限。
- 回调地址、加密 key、verification token 是否正确。
- allow_from 或白名单是否配置，避免开放给所有人。

- 文件发送和接收是否启用，并考虑敏感文件处理。
- 群聊入口是否配置数字分身权限，而不是沿用私聊权限。
- 日志中能否看到平台消息到达 Gateway。

排查顺序

通道问题先看平台侧事件是否触发，再看 Gateway 是否收到，再看 MessageHandler 是否路由，再看 AgentServer 是否执行，最后看回传是否成功。不要只盯模型回答。

第十三章 Heartbeat 与定时任务: 让 Agent 主动醒来

Heartbeat 与 Cron 主动工作模型。本图为基于仓库文档与代码入口重绘的解释性示意图。

13.1 Heartbeat 的两层意义

Heartbeat 首先是健康探测: Gateway 定期向 AgentServer 发起探测，确认连接和 Agent 状态。其次，它也可以触发工作区 `HEARTBEAT.md` 中配置的任务。

如果没有配置任务，响应就是 `HEARTBEAT_OK`。如果配置了活跃任务，Agent 会按心跳读取并执行。

13.2 Heartbeat 配置

典型 YAML:

```yaml
heartbeat:
  every: 3600
  target: web
  active_hours:
    start: 08:00
    end: 22:00
```

字段含义:

字段	作用
`every`	心跳间隔，单位秒
`target`	结果转发通道，如 web
`active_hours`	本地时间活跃窗口，可跨午夜

13.3 `HEARTBEAT.md` 怎么用

`HEARTBEAT.md` 可以放周期性任务，例如：

```
## 章末延伸与实践：主动任务让 Agent 从“被问才动”变成“到点醒来”

个人助理的价值不只是回答问题，还包括按时间提醒、总结、检查和推送。Cron 与 Heartbeat 是 JiuwenSwarm 实现主
          动性的两个入口。Cron 更像明确的定时任务：每天九点总结待办、每周一生成周报。Heartbeat 更像周期性
          探活和轻量任务入口：服务是否正常、是否有 HEARTBEAT.md 中列出的活跃事项需要处理。

Cron 任务的关键字段是 name、cron_expr、timezone、targets、enabled、description 和 wake_offset_seconds。
          description 不应只写“提醒我”，而应写清楚任务目标、输出格式、数据来源和回传方式。比如“每天
          18:30 总结今天 session 中未完成 todo，输出三条以内行动项，推送到 web”。

Heartbeat 的风险在于频率和任务边界。如果每分钟执行复杂搜索或浏览器任务，很快会造成成本和资源压力。建议
          Heartbeat 只做探活、轻量检查和少量固定任务；复杂、耗时、有副作用的工作放到 Cron 或人工触发。

### 主动任务设计模板

| 字段 | 推荐写法 |
|---|---|
| 触发时间 | 用明确 cron 表达式，不用模糊自然语言长期保存 |
| 任务说明 | 写目标、范围、输出格式、失败处理 |
| 回传通道 | Web 用于个人，飞书用于团队通知 |
| 权限 | 提前确认工具是否能无人工确认执行 |
| 失败处理 | 失败时输出原因，而不是静默跳过 |

### 常见误区

不要把 Heartbeat 当万能后台任务队列。它是周期探活和轻量主动执行机制，不适合堆积大量长任务。也不要让 Cron
          任务依赖当前聊天上下文；定时任务触发时应能靠 description、工作区记忆和文件状态独立执行。

# HEARTBEAT

- 每天第一次心跳时，检查今天是否有未完成计划。
- 工作时间内，每两小时总结一次待办进度。
- 如果发现高优先级提醒，推送到 web。
```

建议写得短而明确，不要把 Heartbeat 变成无限制执行脚本。心跳任务应该是轻量、可中断、可观察的。

13.4 Cron 定时任务

Scheduled Tasks 文档给出 Cron Job 的能力:

- 按计划运行一句自然语言指令。
- 让 Agent 定时执行搜索、日报等任务。
- 将结果推送到 Web 或飞书。

常见 Cron 表达式:

```
0 9 * * *       每天 09:00
30 18 * * *     每天 18:30
0 9 * * 1       每周一 09:00
0 * * * *       每小时整点
```

Job 存储在:

```
~/.jiuwenswarm/workspace/cron_jobs.json
```

13.5 Heartbeat 与 Cron 的区别

能力	Heartbeat	Cron
主要目的	健康探测 + 轻任务	按时间计划执行任务
配置位置	`config.yaml` + `HEARTBEAT.md`	Web UI 或 cron_jobs.json
任务粒度	周期性检查	明确日程
结果	可转发通道	可推送到目标通道

13.6 主动任务设计建议

- 低频开始，不要一上来每分钟运行。
- 每个任务要有可验证输出。
- 高风险任务只提醒，不自动执行。
- 生产环境要配权限规则。
- 定时任务结果要落地到会话或日志，便于追踪。

第十四章 浏览器自动化与 MCP: 让 Agent 操作真实网页

浏览器自动化与 MCP。本图为基于仓库文档与代码入口重绘的解释性示意图。

14.1 浏览器工具的价值

很多任务无法只靠 API 完成: 登录后台、填写表单、上传文件、读取需要登录的页面、操作企业内网页面。浏览器工具让 Agent 能连接到真实 Chrome 实例，基于已有登录态完成操作。

14.2 浏览器架构

Browser 文档给出的架构包含:

- Web UI: 浏览器服务面板，配置 Chrome 路径并启动服务。
- Backend: `browser_start_client.py` 启动带 remote debugging 的 Chrome。
- Runtime: Playwright MCP wrapper。
- Agent: `browser_run_task` 把自然语言转为浏览器动作。
- Sessions: 按 session_id 复用浏览器状态。

流程是:

```
UI 配置 Chrome -> 启动 Chrome -> Runtime attach -> Agent 执行网页任务
```

14.3 使用步骤

1. 安装 Chrome。
2. 在 `chrome://version` 查看可执行路径。
3. 在 Web UI 的 Browser service 面板中填写 `CHROME_PATH`。
4. 启动浏览器服务。
5. 在打开的 Chrome 窗口中手动登录相关网站。
6. 从聊天中让 Agent 操作已授权页面。

14.4 配置示例

```yaml
browser:
  chrome_path: "C:\\Users\\YOUR_USER\\AppData\\Local\\Google\\Chrome\\Application\\chrome.exe"
  remote_debugging_address: "127.0.0.1"
  remote_debugging_port: 9222
  user_data_dir: ""
  profile_directory: "Default"
```

对应 `.env` 中的 Playwright MCP 配置需要和端口一致:

```
PLAYWRIGHT_CDP_URL=http://127.0.0.1:9222
PLAYWRIGHT_TOOL_TIMEOUT_S=300
BROWSER_TIMEOUT_S=300
BROWSER_ALLOW_SHORT_TIMEOUT_OVERRIDE=0
```

`BROWSER_ALLOW_SHORT_TIMEOUT_OVERRIDE=0` 是一个好习惯，避免模型把等待时间缩得太短导致任务不稳定。

14.5 MCP Server 配置

MCP 让 Agent 可以连接外部工具服务。JiuwenSwarm 支持 stdio、sse 和 streamable-http 三类传输。

示例:

```yaml
mcp:
  servers:
    - name: my-local-tool
      enabled: true
      transport: stdio
      command: python
      args: ["path/to/server.py", "--transport", "stdio"]
      cwd: .
      env:
        LOG_LEVEL: INFO

    - name: remote-streamable
      enabled: true
      transport: streamable-http
      url: http://127.0.0.1:8000/mcp
      timeout_s: 60
```

管理命令:

```
/mcp list
/mcp reload
/mcp enable <name>
/mcp disable <name>
```

14.6 安全边界

浏览器自动化和 MCP 都是高权限能力。建议:

- 浏览器只连接专用 profile, 不要复用私人主账号环境。
- 重要网站操作先要求确认。
- MCP 服务器来源必须可信。
- 对本地命令型 MCP 使用权限策略。
- 对上传、删除、付款等操作设置 deny 或 ask。

章末延伸与实践：浏览器自动化不是爬虫，而是“受控的本地人机协作”

JiuwenSwarm 的浏览器工具强调控制真实 Chrome 实例。这与无头爬虫不同：真实 Chrome 可以保留登录状态、企业 SSO、Cookie、扩展和站点权限。用户先在被启动的 Chrome 中登录，Agent 再通过 Playwright MCP 执行点击、输入、读取页面、上传附件等动作。

这类能力的关键边界是授权。Agent 可以操作已登录页面，意味着它可能访问邮箱、工单、后台系统和内网应用。因此浏览器自动化必须和权限系统配合：哪些站点允许访问，哪些表单允许提交，是否允许上传文件，是否允许点击删除或支付类按钮，都应该有明确约束。

MCP 的价值在于把外部工具服务接入 Agent。浏览器 MCP 是一种具体场景；一般 MCP 还可以接本地脚本、远程 API、数据库工具、设计工具或内部平台。配置时要区分 stdio、sse、streamable-http 三类传输。stdio 适合本地工具，HTTP 适合已有服务。

浏览器任务最佳实践

1. 先让用户手动完成登录和验证码，不让 Agent 处理敏感认证。
2. 让 Agent 先读取页面并复述计划，再执行写入或提交。
3. 对删除、支付、发布、发送邮件等高风险动作保持 `ask` 或 `deny`。
4. 长流程保持同一个 session，避免标签页和登录状态丢失。
5. 任务结束后记录关键结果，而不是保留大量页面噪音。

MCP 接入检查点

MCP server 的 name 要稳定且全局唯一；远程 URL 要有超时；headers 中的 token 不要写入公开文档；启用后用 `/mcp list` 或等价方式确认实际已加载。浏览器场景还要确保 `PLAYWRIGHT_CDP_URL` 与 Chrome remote debugging 地址一致。

第十五章 权限系统: allow, ask, deny

工具权限决策阶梯。本图为基于仓库文档与代码入口重绘的解释性示意图。

15.1 为什么权限是 Agent 的核心能力

Agent 一旦能读写文件、执行命令、访问网页、调用外部服务，就必须有权限系统。否则它不只是助手，也可能成为误删文件、泄漏信息或执行危险命令的自动化源。

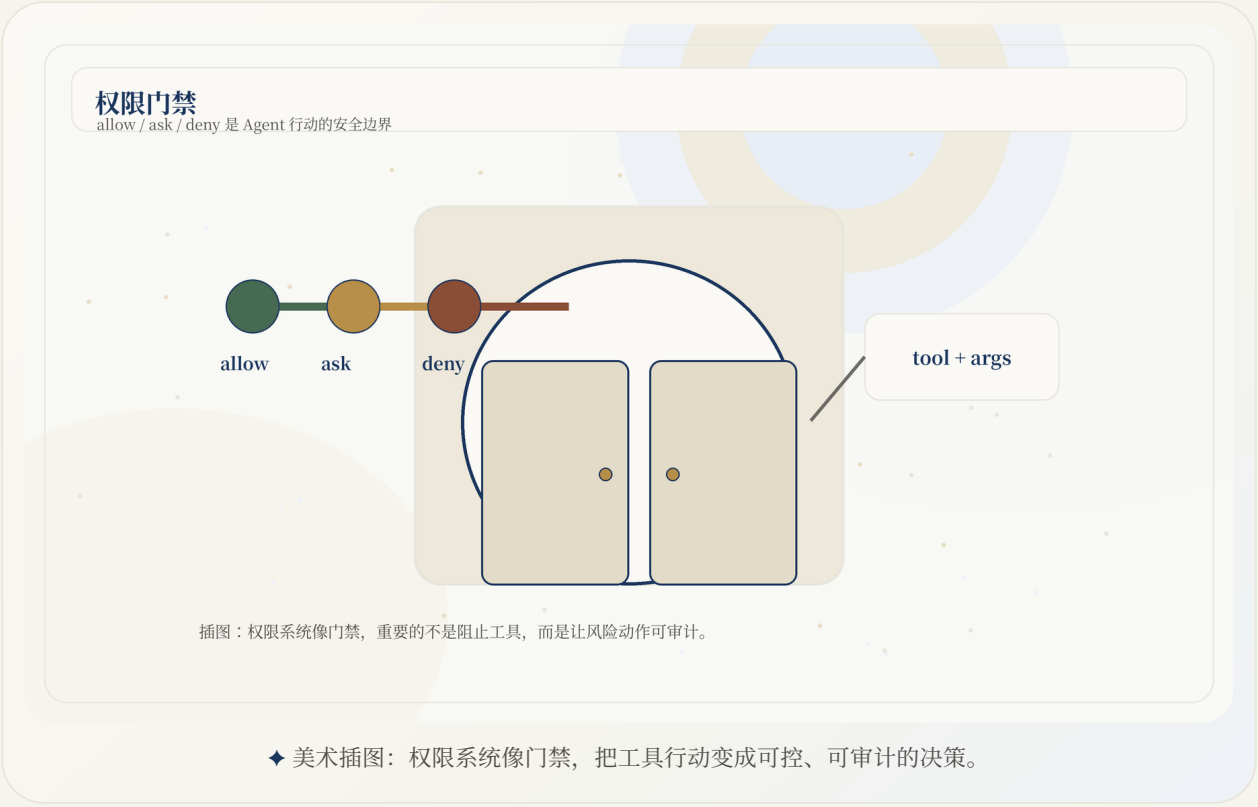
JiuwenSwarm 的权限文档围绕三层动作展开:

动作	含义
`allow`	直接执行
`ask`	执行前请求用户确认
`deny`	拒绝执行

15.2 主开关与模式

`permissions.enabled` 是主开关。关闭时引擎会对工具调用返回 allow, 但生产环境不建议长期关闭。

权限还有 `permission_mode`：`normal` 或 `strict`。在 strict 模式中，中高风险操作更容易从 allow 升级为 ask 或 deny。



✦ 美术插图：权限系统像门禁，把工具行动变成可控、可审计的决策。

15.3 分层策略

权限决策大致按以下顺序:

1. 读取工具级 baseline。
2. 检查内置参数规则。
3. 检查用户参数规则。
4. 检查 approval_overrides。
5. 合并最严格结果。
6. 检查外部目录。
7. 对 shell 操作符做风险升级。

关键原则是「最严格者胜出」。如果某条规则命中 deny, 后续 allow 通常不能绕过，尤其是内置 deny。

15.4 外部目录

当工具访问 workspace 外部路径时，ExternalDirectoryChecker 会追加判断。建议把外部目录默认设为 ask, 只对明确可信路径 allow。

示例思路:

```yaml
permissions:
  enabled: true
  external_directory:
    "*": ask
    "C:/Users/me/projects": allow
    "C:/Users/me/.ssh": deny
```

15.5 approval_overrides

当用户选择「记住此规则」时，系统会把 allow 记录写入 `permissions.approval_overrides`。这适合提高日常任务效率，但不应滥用。

建议只记住低风险、高频路径，例如项目工作目录。不要对整个 home 目录、系统目录或下载执行命令设置全局 allow。

15.6 数字分身下的权限降级

群数字分身场景里，Agent 无法像私聊或 Web 那样弹窗确认，因此 ask 会被降级为 deny。这是正确的设计: 在无人确认的群聊环境里，宁可拒绝，也不要替用户执行不确定操作。

15.7 权限配置原则

默认保守。 初始生产配置应是 ask 多于 allow。

按路径授权。 允许项目目录，不允许整个磁盘。

按工具分级。 读文件和删文件不是同一风险等级。

为群聊单独配置。 群数字分身必须比个人 Web 会话更严格。

保留日志。 权限日志是事后审计的重要材料。

章末延伸与实践：权限系统的目标不是阻止 Agent，而是让行动可控

Agent 最有价值的能力通常也是最危险的能力：读写文件、执行命令、访问网页、调用外部服务、发消息、记忆信息。JiuwenSwarm 的权限系统用 allow、ask、deny 三档描述工具调用是否可直接执行、是否需要确认、是否拒绝。生产中不建议长期全 allow，尤其是 shell、写文件、外部目录和群聊数字分身。

权限决策不是只看工具名。tiered policy 会先看工具级基线，再看内置参数规则、用户参数规则、持久授权和外部目录检查。一个 `bash` 工具本身可能是 ask，但 `ls` 可以低风险 allow，`rm` 应该 ask 或 deny，`rm -rf /` 这类模式应被内置规则拒绝。路径工具同理：读 workspace 内文件和读用户桌面私密文件，不应同等对待。

推荐的安全分层

场景	策略
个人本地试用	读操作较宽松，写操作 ask，shell ask
项目代码模式	只信任当前项目目录，外部目录 ask
群聊数字分身	默认 deny 高风险工具，只开放必要读写
企业部署	严格白名单、日志审计、密钥隔离
浏览器自动化	表单提交、发布、删除、付款类操作 ask/deny

approval_overrides 的边界

“记住这次允许”很方便，但不要把它当永久免死金牌。它适合固定项目目录、低风险命令和稳定工具链，不适合敏感目录、通配过宽命令或网络写操作。定期检查 overrides，删除已经过期的项目路径。

权限排查

工具不执行时先看返回是 deny、ask 未确认、还是工具本身失败。权限拒绝不是 bug，可能是配置正确地保护了你。真正需要改的是规则粒度，而不是简单关闭 master switch。

第十六章 多实例: 一台机器上运行多个独立 Agent

多实例隔离模型。本图为基于仓库文档与代码入口重绘的解释性示意图。

16.1 为什么需要多实例

多实例文档给出三个典型场景:

- 开发和生产隔离。
- 多租户或多用户环境。
- 并行运行不同模型、权限策略或模式。

单实例适合个人试用。多实例适合正式使用和开发调试。

16.2 隔离内容

每个实例独立拥有:

- Workspace。
- 配置文件。
- 端口。
- 进程组。
- 启动锁。

· PID 文件。

这意味着一个 dev 实例的记忆、技能和配置不会污染 prod 实例。

16.3 实例配置

`instances.yaml` 常见位置:

```
~/.jiuwenswarm/instances.yaml
```

示例:

```yaml
instances:
  dev:
    workspace: ~/.jiuwenswarm_dev
    ports:
      agent_server: 19092
      web: 20000
      gateway: 20001
      frontend: 6173
  prod:
    workspace: ~/.jiuwenswarm_prod
    ports:
      agent_server: 20092
      web: 21000
      gateway: 21001
      frontend: 7173
```

默认端口按 base port + instance index x 1000 分配。命名实例可以避免与默认实例冲突。

16.4 常用命令

```
jiuwenswarm-init --name dev
jiuwenswarm-start --list
jiuwenswarm-start --status dev
jiuwenswarm-start --name dev
jiuwenswarm-start --name dev app
jiuwenswarm-start --name dev web
jiuwenswarm-start --stop dev
jiuwenswarm-start --restart dev
```

16.5 启动锁与 PID

每个实例有 `.instance.lock` 和 `.instance.pid`。锁防止同一实例并发启动，PID 用于状态查询和停止进程。

这使 JiuwenSwarm 更像可运维服务，而不仅是脚本。

16.6 多实例最佳实践

- dev 和 prod 使用不同模型 Key 或不同预算账户。
- prod 打开更严格权限。
- 每个实例单独备份工作区。
- 端口写入文档，避免团队成员误连。
- 不要让多个实例共享同一 memory 目录。

章末延伸与实践：多实例是把“个人玩具”推向“多环境运行”的关键

多实例让同一台机器上同时运行独立的 dev、prod、客户 A、客户 B 或不同模型配置。每个实例都应该有自己的 workspace、`.env`、端口、PID、锁文件和日志。这样你可以在 dev 实例测试新 Skill，在 prod 实例保持稳定；也可以为不同用户提供隔离 Agent，而不把他们的记忆和配置混在一起。

端口规划很重要。一个 JiuwenSwarm 实例通常会涉及 AgentServer、Gateway、Web、Frontend 等多个端口。命名实例使用端口偏移可以降低冲突，但实际部署仍要检查端口是否被占用。启动失败时不要只看一个端口，要按服务类型逐个确认。

多实例设计建议

- 实例名使用业务含义，如 `dev`、`prod`、`team-a`，不要用临时随机名。
- dev 与 prod 使用不同 API Key 或至少不同 `.env`，避免测试消耗生产额度。
- 自定义 Skill 先在 dev 实例安装和进化，再同步到 prod。
- 重要实例用独立备份计划，尤其是 memory 和 skills。
- 停止和重启实例使用管理命令，不要只手动 kill 其中一个子进程。

什么时候不需要多实例

如果你只是个人本地试用，一个默认实例足够。过早使用多实例会增加路径、端口和配置复杂度。只有当你需要隔离环境、隔离用户、并行测试或不同通道策略时，多实例才明显有价值。

常见故障

实例启动失败但默认实例正常，通常是端口、workspace 或 bootstrap `.env` 问题。实例行为与预期不一致，先确认当前命令是否带了 `--name`，以及进程加载的是否是目标实例的 `.env`。

第十七章 Team 与分布式协作

分布式 Team 控制面与数据面。本图为基于仓库文档与代码入口重绘的解释性示意图。

17.1 Team 模式的目标

Team 模式把一个任务拆给多个 Agent 成员协作。Leader 负责任务拆解和协调，成员负责执行。持久团队还可以积累团队记忆。

本地 Team 已经有用；分布式 Team 进一步允许 leader 和 teammate 运行在不同进程或机器上。

17.2 分布式 Team 的关键配置

分布式文档列出几个必须理解的配置:

配置	作用
`team.runtime.mode`	`local` 或 `distributed`
`team.runtime.role`	`leader` 或 `teammate`
`team.transport.type`	常见为 `pyzmq`
`react.a2x_registry`	teammate 注册和 leader 查找的注册中心
`team.storage`	共享业务状态，例如任务和消息

✦ 美术插图：Team 模式更像蜂群协作，而不是简单复制多个聊天机器人。

17.3 控制平面和数据平面

文档特别区分控制平面和数据平面:

- 控制平面: teammate 注册空闲节点，leader 预留 teammate, 发送 bootstrap, teardown 时释放或重置。
- 数据平面: 业务消息、任务声明、完成状态和团队消息通过 team runtime 与共享存储流转。

这是一种成熟的分布式设计视角。控制平面解决「谁加入团队」, 数据平面解决「加入后如何协作」。

17.4 共享工作区注意事项

分布式文档提醒: 默认情况下 leader 和 teammate 会在各自 HOME 下创建形状相似但物理不同的 team-workspace。如果 leader 需要直接读取 teammate 写的文件，必须配置共享 `team.workspace.root_path`, 或通过消息、数据库状态和文件传输工具返回结果。

不要共享 `.agent_teams` 整个目录，因为其中包含 team.db、成员工作区、symlink 和本地运行态，跨节点共享可能破坏初始化和 kickoff。

17.5 什么时候需要分布式 Team

场景	是否需要
个人任务拆解	不一定，本地 Team 即可
大型代码库多角色开发	可能需要
不同机器上有不同工具环境	适合分布式
多用户共享一个 Agent 集群	适合分布式
只是想让回复更像多人讨论	不需要，提示词即可

章末延伸与实践：Team 模式不是“多开几个聊天机器人”

Team 模式的核心是协作结构：Leader 负责拆解、调度和汇总，成员负责具体任务。分布式 Team 进一步把成员放到不同进程或机器上，通过注册、预留、bootstrap、共享存储和消息协作完成任务。它解决的不是普通问答，而是长周期、多角色、需要状态沉淀的任务。

分布式 Team 文档强调控制面和数据面的分离。控制面负责让空闲 teammate 注册到 A2X Registry，leader 在需要时 reserve 可用节点并发送 bootstrap；数据面则通过 Team runtime 和共享存储传递任务、成员状态和业务消息。文件面又是第三个问题：如果 leader 必须直接读 teammate 写出的文件，就需要显式共享 workspace 或通过消息/文件传输返回结果。

适合 Team 的任务

- 大型代码改造，需要分析、实现、测试、文档多个角色。
- 长文档项目，需要资料整理、结构设计、撰写、校对、排版分工。
- 多服务系统排障，需要不同成员并行检查日志、配置和代码。
- 企业流程自动化，需要一个 Leader 协调多个工具型 Agent。

不适合 Team 的任务

简单问答、一次性小改动、无须并行的短任务不适合 Team。Team 有调度成本、状态成本和同步成本。一个人能十分钟完成的任务，用 Team 反而可能更慢。

部署提醒

分布式配置里不要在多机部署中使用 `127.0.0.1` 作为对外地址。共享数据库必须两端可见且连接字符串一致。共享工作区要谨慎，只共享 team workspace，不要把 `.agent_teams` 运行态目录直接放到 NFS 上。

第十八章 E2A 与 A2A: 多协议时代的 Agent 网关

E2A/A2A 协议关系。本图为基于仓库文档与代码入口重绘的解释性示意图。

18.1 E2A 是什么

E2A 即 Everything-to-Agent, 是 Gateway 归一化后发送给 AgentServer 的内部协议描述。它不规定传输层，而是规定请求包络、身份、会话、参数、附件和来源信息等字段。

文档强调一个坑: `method` 在不同来源中含义不同。Gateway 到 AgentServer 的 `chat.send`、ACP 的 `session/prompt`、内部 heartbeat 的 `null` 都可能出现在同一字段里，因此适配器必须明确转换。

18.2 E2A 的关键字段

字段	作用
`protocol_version`	协议版本
`request_id`	Gateway 与 AgentServer 的请求 ID
`session_id`	会话 ID
`message_id`	平台或 A2A 消息 ID
`channel`	来源通道
`method`	业务或桥接方法名
`params`	业务参数、文本、附件、模式等
`provenance`	来源协议和转换信息

最重要的是 `params`: 新协议不应使用顶层 payload, 而应把用户文本、文件和选项放进统一的参数字典。

18.3 A2A 入站

A2A 文档说明了 Gateway 侧 A2A Server 的职责: 外部 A2A client 请求进入 `A2AChannel`, 转为内部 Message/E2A, 经 ChannelManager 和 MessageHandler 送入 AgentServer, 再把响应映射回 A2A 事件或结果。

启用前需要安装可选依赖:

```
pip install "jiuwenswarm[a2a]"
```

并设置:

```
A2A_SERVER_ENABLED=true
A2A_SERVER_HOST=127.0.0.1
A2A_SERVER_PORT=19100
A2A_SERVER_PATH=/a2a
```

18.4 为什么协议层重要

没有协议层，项目很容易被每个通道的特殊字段拖垮。E2A 的意义是让各种入口最终进入统一语义:

```
外部平台消息 -> Gateway adapter -> E2A envelope -> AgentServer -> E2A response -> 平台响应
```

这为未来接入更多平台、更多 Agent 协议和企业网关打下基础。

章末延伸与实践：E2A 的价值在于把外部协议变成内部共同语言

Web、ACP、A2A 和 IM 通道的消息结构不同。如果 AgentServer 直接理解每个平台，就会不断膨胀。E2A 的思路是：在 Gateway 或 Adapter 边缘把外部请求转成统一信封，内部只处理 `session_id`、`channel`、`method`、`params`、`files`、`provenance` 等字段。

这种统一信封能减少重复逻辑，也让调试更清楚。你可以检查某个外部请求是否被正确映射：用户文本是否在 `params.query`，附件是否在 `params.files`，A2A 的 task/context 是否进入对应字段，时间戳和身份是否规范。若映射正确但执行失败，问题在 AgentServer；若映射错误，问题在 Adapter 或通道。

A2A 侧的 Agent Card 和 JSON-RPC endpoint 则让 JiuwenSwarm 可以作为外部 Agent 网络中的一个服务。它不是简单提供聊天 API，而是把外部 A2A Message 转成内部 Message，再把结果映射回 A2A stream 或响应。这为未来多 Agent 互操作打基础。

协议开发建议

- 不要把平台原始 JSON 全部塞进核心执行层；只保留必要 normalized fields。
- `method` 在不同协议中含义不同，适配时要显式映射。
- 附件字段要规范成统一 file ref，避免每个通道各读各的格式。
- 保留 provenance，方便调试某条消息从哪个协议转换而来。
- 新增协议先写最小 SendMessage 闭环，再加 streaming、files 和 auth。

排查方式

协议问题最怕“看起来收到了消息，但 Agent 不理解”。此时打印或记录 E2AEnvelope，比盯着平台原始回调更有用。只要信封正确，后续执行就能复用同一套 AgentServer 能力。

第十九章 二次开发路线图

二次开发路线图。本图为基于仓库文档与代码入口重绘的解释性示意图。

19.1 从 pyproject 看项目边界

`pyproject.toml` 是理解项目边界的好入口。它声明了多个 console scripts:

命令	入口
`jiuwenswarm-app`	编排 AgentServer + Gateway
`jiuwenswarm-agentserver`	单独启动 AgentServer
`jiuwenswarm-gateway`	单独启动 Gateway
`jiuwenswarm-web`	Web 通道
`jiuwenswarm-start`	启动与实例管理
`jiuwenswarm-init`	工作区初始化
`jiuwenswarm-desktop`	桌面应用
`jiuwenswarm-tui`	TUI
`jiuwenswarm-acp`	ACP 连接
`jiuwenbox`	另一个打包在项目中的工具入口

二次开发时，先找到对应入口，比盲目搜索全仓库更高效。

19.2 你可以改什么

按风险从低到高排序:

1. 写 Skill: 最安全，扩展能力不必改核心代码。
2. 改配置模板: 适合定制默认模型、通道和权限。
3. 接入新 MCP 工具: 保持工具外置，系统通过协议调用。
4. 写新 Channel: 需要理解 Gateway 和 MessageHandler。
5. 改记忆系统: 需要理解索引、文件监听和检索。
6. 改 Team runtime: 需要理解分布式协议和共享存储。
7. 改权限引擎: 高风险，必须配测试。

19.3 新 Channel 开发思路

一个新通道至少要解决:

- 如何接收外部消息。
- 如何识别用户和会话。
- 如何处理文件和附件。
- 如何把消息转换为内部 Message/E2A。
- 如何把 Agent 响应发送回外部平台。
- 如何处理控制命令。
- 如何配置凭证和权限。

不要一开始就把平台所有功能做完。先支持纯文本单聊，再支持文件，再支持群聊和事件。

19.4 新 Skill 开发思路

开发一个 Skill 的推荐流程:

1. 写清楚触发场景。
2. 用最小 `SKILL.md` 跑通。
3. 加入输入约束和输出格式。
4. 加入验证步骤。
5. 加入失败处理。
6. 用真实任务测试。
7. 打开 evolution, 收集失败经验。
8. 审核后固化进 `SKILL.md` 。

19.5 新 MCP 工具开发思路

如果你已有一个本地脚本或服务，不一定要把它直接改进 JiuwenSwarm。更好的做法是包装成 MCP server, 让 Agent 通过 MCP 调用。

适合 MCP 的工具:

- 企业内部搜索。
- 数据库查询。
- 工单系统。
- 本地文件处理。
- 报表生成。
- 专用爬虫或采集器。

19.6 测试建议

- 对 Skill 写端到端任务样例。
- 对权限策略写危险命令用例。
- 对 Channel 写 session_id 映射测试。
- 对记忆写检索召回测试。
- 对定时任务写重复触发和幂等测试。
- 对多实例写端口冲突测试。

章末延伸与实践：二次开发要按风险层级推进

二次开发 JiuwenSwarm 时，不建议一开始就改 Agent 核心。更稳的路径是从外围到核心：先读文档和配置，跑通最小闭环；再写一个本地 Skill；然后接一个 MCP 工具或轻量通道；再改权限、记忆或上下文策略；最后才进入 E2A、A2A、Team 和分布式协作。

写 Skill 是最适合入门的扩展点。它不要求理解全部进程架构，却能立刻提升实际能力。比如写一个“仓库电子书生成 Skill”，把资料收集、章节规划、配图生成、PDF 渲染检查固化下来。等 Skill 稳定后，再考虑把其中某些脚本变成 MCP server，或把结果推送到飞书。

推荐开发环境

- 使用源码安装和独立 dev 实例，不要直接在生产工作区改。
- 所有自定义 Skill 用 Git 管理，避免演进记录覆盖原始设计。
- 修改通道前先写模拟输入，验证 MessageHandler 归一化结果。
- 修改权限前准备一组允许、询问、拒绝的单元测试样例。
- 修改协议前保存真实请求和转换后的 E2AEnvelope 对照。

代码阅读顺序

先看 `pyproject.toml` 的入口脚本，理解有哪些进程；再看 `init_workspace` 和工作区工具，理解数据放哪里；再看 Gateway 与 AgentServer 入口，理解消息流；之后按目标阅读 Channels、Modes、Skills、Memory、Permissions、Team。不要从最深的 Runner 开始，否则会缺少上下文。

开发者验收标准

一个扩展是否合格，不只看“能跑”。还要看是否可配置、可禁用、可记录日志、可恢复失败、可被权限系统约束、是否污染工作区、是否有最小文档和示例。Agent 扩展越靠近工具执行，越需要这些工程化边界。

第二十章 实战落地路径

落地实施蓝图。本图为基于仓库文档与代码入口重绘的解释性示意图。

20.1 第一天: 跑通最小闭环

目标: 能从 Web UI 对话。

步骤:

1. 安装 Python 3.11 或 3.12。
2. 创建虚拟环境。
3. `pip install jiuwenswarm`。
4. `jiuwenswarm-init`。
5. `jiuwenswarm-start`。
6. 打开 Web UI。
7. 配置默认模型。
8. Test 成功。
9. 发起第一轮对话。

验收标准: 模型能回复，配置能保存，重启后仍可用。

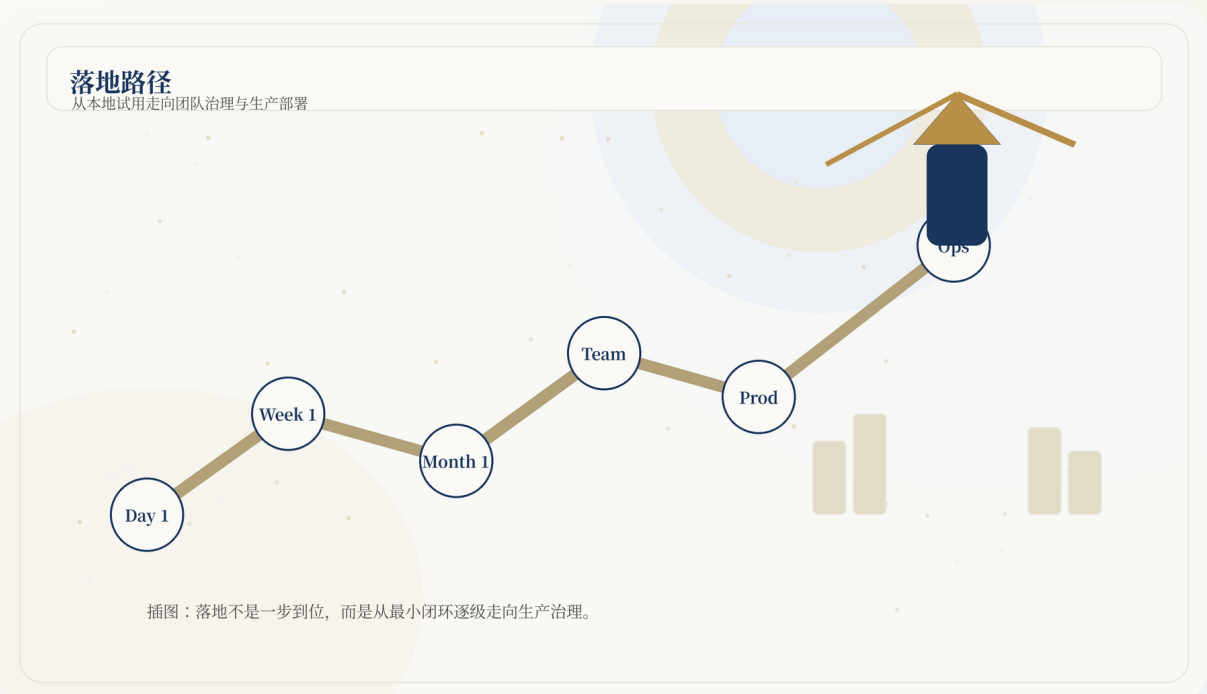
20.2 第一周: 形成个人工作流

目标: 让 JiuwenSwarm 记住你、能执行任务、能复用技能。

建议任务:

- 配置 Embedding。
- 让它记录你的基本偏好和常用项目路径。
- 安装或导入 1 到 2 个 Skill。
- 开启任务规划，完成一个多步骤任务。
- 配置一个每天一次的 Cron 总结任务。
- 设置权限策略，至少保护外部目录。

验收标准: 它不再只是聊天，而能在一个具体工作流中节省时间。



✦ 美术插图：从本地试用到生产治理，需要分阶段推进。

20.3 第一个月: 做成个人 AI 管家

目标: 接入常用入口，形成长期记忆和主动任务。

建议任务:

- 接入 TUI 或飞书。
- 设计 `HEARTBEAT.md`。
- 配置浏览器自动化。
- 为常用重复任务写自定义 Skill。
- 打开 Task Memory。
- 定期审核 `MEMORY.md`，`USER.md`，`evolutions.json`。

验收标准: 它能主动提醒、能跨会话记住偏好、能通过 Skill 处理重复流程。

20.4 团队或生产前检查

上线前至少确认：

项目	检查点
模型	Key 权限、预算、限流策略
工作区	备份、路径权限、实例隔离
通道	凭证保护、回调地址、用户映射
权限	高危工具 deny, 外部目录 ask
日志	错误日志和权限日志可查
定时任务	幂等、频率、失败重试
Skill	来源可信、允许工具明确
记忆	不写入敏感凭证

章末延伸与实践：从试用到落地的六阶段路线

落地 JiuwenSwarm 不应该从“接入所有通道”开始，而应从一个稳定闭环开始。第一阶段是本地可用：Web 或 TUI 能正常对话，默认模型稳定，日志可查。第二阶段是数据边界：明确工作区、备份、记忆规则和权限策略。第三阶段才接通道：根据实际工作入口选择飞书、小艺或其他平台。

第四阶段是主动任务：把每天、每周固定工作交给 Cron 或 Heartbeat。第五阶段是能力沉淀：把反复出现的流程写成 Skill，把重复错误写进自进化记录，把项目经验写进记忆。第六阶段是运维化：多实例、升级回滚、权限审计、日志归档和故障演练。

个人用户落地样例

个人开发者可以先把 JiuwenSwarm 当成“项目助理”。Web 负责日常对话，TUI 负责代码项目，Cron 每天傍晚总结未完成任务，Memory 记住项目约定，Skill 负责生成周报、整理 issue、撰写文档。权限上，默认允许读项目目录，写文件需要确认，shell 操作保持 ask。

团队用户落地样例

团队可以先从飞书机器人进入，但不要马上启用全自动数字分身。先让机器人在测试群中处理只读查询和文档总结，再逐步开放创建任务、推送提醒和读取项目知识库。生产群启用前，应配置 allow_from、owner scope、工具白名单和敏感信息禁记规则。

成功指标

- 重复性任务是否减少人工步骤。
- Agent 是否能跨会话记住稳定事实。
- Skill 是否让输出质量更稳定。
- 权限是否减少误操作风险。

- 出错后是否能通过日志和工作区追踪原因。
- 升级前后是否能保留记忆和自定义能力。

第二十一章 项目维护观察

项目维护检查点。本图为基于仓库文档与代码入口重绘的解释性示意图。

21.1 文档质量对 Agent 项目尤其重要

JiuwenSwarm 的文档覆盖面很广，包括安装、快速开始、配置、工作区、模式、技能、自进化、通道、CLI、心跳、记忆、MCP、权限、A2A、E2A、分布式 Team 和多实例。这是优点。

但文档越多，越需要同步维护。快速开始文档中存在一处明显的 Git merge conflict 残留片段，出现在「何时清空会话」附近。正式发布前应修复此类冲突标记，避免新用户复制到错误内容。

21.2 命名历史需要统一

部分代码注释仍出现 JiuWenClaw 等历史名称，而 README 和包名使用 JiuwenSwarm。这不一定影响运行，但会影响新开发者理解。建议在贡献指南中说明命名演进，或者逐步统一注释与文档。

21.3 安全默认值需要谨慎

配置文档中权限开关默认关闭。对个人本地试用可以理解，但对生产或企业通道，建议提供一份安全默认模板，让用户更容易从「保守可用」开始，而不是从「完全放开」开始。

21.4 Skill 自进化需要治理

Skill 自进化很有价值，但也需要治理机制。建议维护者为 evolution 设计审核工作流，例如：

- 自动记录 evolution。
- 每周批量审查。
- 标记误报。
- 固化高质量经验。
- 将通用改进回贡献到上游 Skill。

21.5 未来可以强化的方向

- 更清晰的架构图和启动时序图。
- 更完整的权限策略示例库。
- Skill 开发模板和测试框架。
- 通道开发脚手架。
- Memory 可视化和审计工具。
- 多实例 Web 管理界面。
- A2A/ACP 的端到端集成示例。

章末延伸与实践：快速演进项目必须学会“交叉验证”

开源 Agent 项目变化快，文档、代码和截图很容易不同步。维护观察不是挑错，而是帮助读者建立判断习惯：README 给方向，安装文档给路径，配置模板给当前默认值，pyproject 给真实命令入口，源码给最终行为。它们发生冲突时，应以代码和当前版本配置为准，同时记录文档差异。

第一版中已经注意到 Quickstart 文档存在冲突标记残留。这类现象说明文档可能处于快速合并中。读者在复制命令前应确认上下文，不要把冲突块当成正式步骤。类似地，路径名从旧版迁移到新版时，文档可能同时出现旧路径和新路径，要结合实际工作区检查。

维护者检查表

检查点	为什么重要
README 与 pyproject 命令是否一致	避免用户按不存在的脚本启动
中英文文档是否同步	避免不同语言用户看到不同流程
截图是否对应当前 UI	配置入口变化会直接影响上手
默认配置是否和说明一致	尤其是 permissions、memory、evolution
路径是否统一	workspace 迁移最容易造成误解
升级说明是否覆盖备份	记忆与 Skill 是用户最重要资产

读者如何保护自己

正式部署前，把本书当作结构化地图，而不是替代官方最新文档。遇到关键命令、路径、配置默认值和安全策略，回到仓库当前版本核对。升级前备份，修改前建分支，接通道前开测试群，开权限前先跑只读任务。这些习惯比任何单份文档都可靠。

附录 A 常用命令速查

安装与启动

```
python -m venv jiuwenswarm-env
source jiuwenswarm-env/bin/activate
pip install jiuwenswarm
jiuwenswarm-init
jiuwenswarm-start
```

TUI

```
pip install jiuwenswarm-tui
jiuwenswarm-tui
```

多实例

```
jiuwenswarm-init --name dev
jiuwenswarm-start --list
jiuwenswarm-start --status dev
jiuwenswarm-start --name dev
jiuwenswarm-start --stop dev
jiuwenswarm-start --restart dev
```

模式切换

```
/mode agent
/mode code
/mode team
/mode agent.plan
/mode agent.fast
/mode code.plan
/mode code.normal
/switch plan
/switch fast
/switch normal
```

会话控制

```
/new_session
/branch
/branch <name>
/rewind 3
/rewind confirm 3
/rewind cancel
```

Skill 与 MCP

```
/skills list
/evolve <skill_name>
/evolve list
/mcp list
/mcp reload
/mcp enable <name>
/mcp disable <name>
```

上下文压缩

```
/compact
```

附录 B 配置速查

默认模型

```yaml
api_base: https://api.openai.com/v1
api_key: sk-your-key
model: gpt-4o
model_provider: OpenAI
```

Embedding

```yaml
embed_api_base: https://api.siliconflow.cn/v1
embed_api_key: sk-your-key
embed_model: BAAI/bge-large-zh-v1.5
```

Heartbeat

```yaml
heartbeat:
  every: 3600
  target: web
  active_hours:
    start: 08:00
    end: 22:00
```

Browser

```yaml
browser:
  chrome_path: "/Applications/Google Chrome.app"
  remote_debugging_address: "127.0.0.1"
  remote_debugging_port: 9222
  user_data_dir: ""
  profile_directory: "Default"
```

MCP

```yaml
mcp:
  servers:
    - name: remote-streamable
      enabled: true
      transport: streamable-http
      url: http://127.0.0.1:8000/mcp
      timeout_s: 60
```

权限骨架

```yaml
permissions:
  enabled: true
  permission_mode: normal
  defaults:
    "*": ask
  external_directory:
    "*": ask
  tools:
    bash: ask
    write_file: ask
```

附录 C 术语表

术语	解释
AgentServer	负责 Agent 执行、工具、技能、团队和扩展的服务进程
Gateway	负责通道接入、消息处理、路由、心跳和定时任务的服务进程
Workspace	保存配置、记忆、技能、会话和任务状态的运行目录
Skill	可安装、可管理、可复用的能力包
Evolution	Skill 根据失败和用户纠正积累的改进记录
Memory	持久化记忆系统，包括文件、索引和外部记忆
Dreaming	空闲时扫描历史会话并巩固长期记忆的机制
Task Memory	任务级经验检索和学习系统
Coding Memory	Code 模式下专用的代码经验记忆
Heartbeat	网关到 AgentServer 的周期性健康探测和轻任务触发
Cron	定时任务系统，用 cron 表达式触发 Agent 工作
MCP	Model Context Protocol, 用于连接外部工具服务
E2A	Everything-to-Agent, Gateway 到 AgentServer 的内部归一化协议
A2A	Agent-to-Agent 协议入口，用于外部 Agent 客户端接入
Team	多 Agent 协作模式，支持 leader 和 teammate

附录 D 本书资料来源

本书基于 AtomGit 仓库 openJiuwen/jiuwenswarm 的 README、配置文档、模式文档、Skill 文档、记忆文档、通道文档、权限文档、协议文档和关键入口代码整理。重点参考路径如下:

- `README.md`：项目定位、核心特性、安装命令、文档导航、许可证。
- `pyproject.toml`：包名、版本、Python 要求、依赖、console scripts。
- `docs/en/Quickstart.md`：快速开始、模型配置、会话和记忆清理。
- `docs/en/InstallGuide.md`：安装路径、源码安装、升级与备份。
- `docs/en/Configuration.md`：模型、Embedding、第三方服务、自进化、上下文压缩和权限配置。
- `docs/en/Agent.md`：工作区结构。
- `docs/en/Modes.md`：Agent、Code、Team 模式。
- `docs/en/Skills.md`：Skill 概念、来源、安装和管理。
- `docs/en/SkillSelfEvolution.md`：Skill 自进化组件、信号、流程和 `evolutions.json`。
- `docs/en/Memory.md`：内置记忆、外部记忆、Dreaming、Team Memory 和检索栈。
- `docs/en/TaskMemory.md`：任务记忆工具和算法。
- `docs/en/CodingMemory.md`：Code 模式记忆。
- `docs/en/TaskPlanning.md`：TodoToolkit 和长任务规划。
- `docs/en/ScheduledTasks.md`：Cron 定时任务。
- `docs/en/Heartbeat.md`：心跳机制和 `HEARTBEAT.md`。
- `docs/en/Channels.md`：小艺、飞书、数字分身和多通道。
- `docs/en/Browser.md`：浏览器自动化和 Playwright MCP。
- `docs/en/MCPConfiguration.md`：MCP server 配置。
- `docs/en/ToolPermissionsSecurity.md`：权限策略、外部目录和审批覆盖。
- `docs/en/A2A.md` 与 `docs/en/E2A-protocol.md`：A2A/E2A 协议与字段映射。
- `docs/en/DistributedTeam.md`：分布式 Team。
- `docs/en/MultiInstance.md`：多实例运行。
- `jiuwenswarm/app.py`，`jiuwenswarm/server/app_agentserver.py`，`jiuwenswarm/gateway/app_gateway.py`，`jiuwenswarm/start_services.py`，`jiuwenswarm/init_workspace.py`：运行入口与进程结构。

附录 E 第三版图示清单

第三版保留第二版的 22 张解释性图示，并新增 9 张原创美术插图。解释性图示负责说明结构、状态和流程；美术插图负责在长文中建立更直观的产品隐喻与阅读节奏。

E.1 解释性图示

图	主题	用途
图 0	阅读路线图	帮助不同角色选择阅读路径
图 1	能力地图	建立入口、执行、状态、安全的总览
图 2	首次启动闭环	避免把安装和可用混为一谈
图 3	工作区结构	明确哪些数据需要备份和迁移
图 4	进程拓扑	区分 Gateway 与 AgentServer 故障边界
图 5	配置地图	理解模型、记忆、通道、权限的关系
图 6	模式矩阵	选择 agent/code/team 的依据
图 7	Slash 解析边界	知道命令在哪里被处理
图 8	任务状态机	理解长任务如何保持进度
图 9	记忆管线	区分写入、索引和读回
图 10	Skill 包结构	设计可维护 Skill
图 11	自进化闭环	把错误变成可审计经验
图 12	多通道入口	理解通道归一化
图 13	主动工作模型	设计 Cron 与 Heartbeat
图 14	Browser/MCP	理解浏览器自动化授权边界
图 15	权限阶梯	排查 allow/ask/deny 决策
图 16	多实例隔离	规划 dev/prod 或多租户
图 17	分布式 Team	区分控制面、数据面和文件面
图 18	E2A/A2A	理解协议适配层
图 19	开发路线	按风险从 Skill 走向核心改造
图 20	落地蓝图	从试用到运维化部署
图 21	维护检查	快速演进项目的验证习惯

E.2 美术插图

插图	放置位置	叙事用途
插图 0	封面	将 JiuwenSwarm 视觉化为个人 AI 管家工作台
插图 1	第一章	强化“聊天只是入口，长期运行才是价值”的产品隐喻
插图 2	第五章	把配置系统表现为能力控制台
插图 3	第九章	把记忆系统表现为需要照料的花园
插图 4	第十章	把 Skill 表现为可反复锻造的能力工具
插图 5	第十二章	把多通道表现为围绕 Gateway 的星座
插图 6	第十五章	把权限策略表现为工具调用门禁
插图 7	第十七章	把 Team 模式表现为蜂群协作
插图 8	第二十章	把落地路径表现为逐级推进的路线

附录 F 出版预检报告

本报告记录第四版出版增强版的自动化检查结果。它不能替代人工校对、法律审查、出版社三审三校、EPUBCheck 官方校验和印厂预检，但可以作为送审前的基础质量记录。

F.1 自动化检查摘要

检查项	结果
PDF 页面规格	A4
PDF 页数	89 页
PDF 文件大小	5.0 MB
EPUB 文件大小	3.6 MB
PDF 标题元数据	`JiuwenSwarm: 打造可进化的个人 AI 管家`
PDF 作者元数据	`OpenAI GPT-5.5 Pro 整理 · 出版增强版`
PDF 加密	否
EPUB ZIP 完整性	通过
渲染抽检	已生成 PNG 渲染抽检图，未发现明显空白页或封面缺图
交付包	PDF、EPUB、Markdown、HTML、CSS、图片素材和预检报告已打包

F.2 已补齐的出版要素

- 补充版权页与出版声明，明确本书不是官方出版物。
- 补充来源、商标、插图与免责声明，降低误读风险。
- 保留资料来源附录，方便后续事实复核。
- 保留图示清单，区分解释性图示和美术插图。
- 修正 PDF/EPUB 中文标题与作者元数据。

F.3 仍需人工确认的出版要素

项目	建议动作
ISBN/CIP	若正式出版，交由出版社流程办理
人工校对	至少进行一轮技术校对和一轮文字校对
事实复核	对命令、路径、版本号、默认端口、配置字段逐项核对最新仓库
授权审核	核对源项目 LICENSE、第三方名称和引用边界
EPUBCheck	用官方 EPUBCheck 进行完整规范校验
无障碍 PDF	若平台要求 tagged PDF，应另行制作结构化 PDF
纸质印刷	另行制作封面、书脊、出血、CMYK/灰度和印厂专用 PDF

F.4 建议出版流程

1. 以本版 Markdown 作为可校对母稿。
2. 对所有命令和配置块做技术复核。
3. 补出版主体、版权归属、ISBN/CIP 信息。
4. 使用 EPUBCheck、PDF 预检工具和目标平台后台各跑一次。
5. 根据平台反馈生成最终上架版。