ZCode 系统教程

> 覆盖安装配置、核心功能、扩展机制与工程实践四大部分。
>
> 适用版本：ZCode v3.14.x（桌面端：macOS / Windows / Linux）
> 编写时间：2026 年 9 月

目录

前言 · 如何使用本书

第一部分　入门篇

- 第 1 章　认识 ZCode：什么是 Agentic 开发环境
- 第 2 章　安装与首次启动
- 第 3 章　连接模型与套餐
- 第 4 章　第一次对话：工作台与 ZCode Agent

第二部分　核心功能篇

- 第 5 章　目标模式：让长任务自己跑完
- 第 6 章　浏览器自动化
- 第 7 章　任务与文件管理
- 第 8 章　Wiki：代码库架构导读
- 第 9 章　Memory：项目长期记忆
- 第 10 章　自动化与闲时任务
- 第 11 章　编辑历史对话与撤销改动
- 第 12 章　远程开发：SSH / WSL / Docker
- 第 13 章　Remote Control 与 Bot Channel
- 第 14 章　子智能体：并行与分工

第三部分　扩展机制篇

- 第 15 章　Skill：可复用的工作方法
- 第 16 章　Command：可复用的提示词
- 第 17 章　Plugin：打包与分发能力
- 第 18 章　MCP：接入外部工具与数据
- 第 19 章　Hooks：事件驱动的自动化

第四部分　工程实践篇

- 第 20 章　安全操作确认与权限治理
- 第 21 章　智能体开发环境工具集
- 第 22 章　使用统计与额度管理
- 第 23 章　反馈、排错与最佳实践

附录

- 附录 A　快捷键表
- 附录 B　配置文件与目录速查
- 附录 C　术语表
- 附录 D　综合实战：一个完整项目的全流程演练

前言 · 如何使用本书

一、这本书讲什么

ZCode 是智谱（Z.ai）推出的 **Agentic Development Environment（ADE，智能体开发环境）**，官方定位是"把 GLM-5.3 带入真实编程工作流的桌面工作台"。

它与传统编辑器的根本区别在于：**传统编辑器里，人是执行者，AI 是补全工具；ZCode 里，AI Agent 是执行者，人是审核者。** 你用自然语言描述一个开发目标，ZCode Agent 会自己读代码、改文件、跑命令、开浏览器验证、检查 Git 状态，最后把一份可核对的结果交给你审查。

这种工作方式对开发者提出了新的能力要求：不再是"怎么写代码"，而是"怎么把任务说清楚、怎么把风险管住、怎么验收 AI 的产出"。本书正是围绕这三件事展开。

二、本书的编写原则

1. **以官方文档为准。** 所有功能描述、参数、路径、限制条件均来自官方文档，不臆造未公布的行为。
2. **区分"是什么"和"怎么用"。** 每章先讲清机制，再给操作步骤，最后给实践建议。
3. **标注限制与坑点。** 官方文档中明确写出的"当前不支持""默认关闭""仅对新会话生效"等约束，本书会显式标出，因为这些正是实际使用中最容易踩的坑。

三、本书的符号约定

标记	含义
`代码`	文件路径、命令、配置项、JSON 键名
注意	官方明确指出的限制或易错点
建议	来自官方使用建议或工程实践的经验做法
> 引用块	官方文档原文要点或示例提示词

四、学习路线建议

阶段	章节	目标
第 1 天	第 1—4 章	装好、连上、跑通第一个任务

阶段	章节	目标
第 1 周	第 5—11 章	用目标模式推进长任务，用文件树与 Wiki 管住上下文
第 2 周	第 12—14 章	接入远程环境与移动端，用子智能体做并行调研
第 3—4 周	第 15—19 章	沉淀团队的 Skill、Command、Plugin、MCP、Hooks
持续	第 20—23 章	建立权限治理与成本控制规范

第一部分　入门篇

本部分目标是让读者在两小时内完成从零到"跑通第一个任务"的全过程，并理解 ZCode 的几个核心概念：工作区、任务、Agent、执行模式。

第 1 章　认识 ZCode：什么是 Agentic 开发环境

1.1 本章目标

- 理解 ADE 与传统 IDE、AI 插件的本质差异
- 掌握 ZCode 的五项核心能力
- 了解版本、平台与权益体系

1.2 从 IDE 到 ADE

软件开发工具的演化大致经历了三个阶段：

阶段	代表形态	人的角色	AI 的角色
第一阶段	传统 IDE（VS Code、IntelliJ）	编写全部代码	无
第二阶段	AI 辅助编辑器（Copilot、Cursor）	编写核心代码	行级 / 块级补全
第三阶段	智能体开发环境（ADE）	描述目标、审核结果	自主规划、修改、验证

ZCode 属于第三阶段。官方对它的定义是：

> ZCode 是一个把 GLM-5.3 带入真实编程工作流的 Agentic Development Environment（ADE），覆盖复杂开发任务中的规划、编码、审查与迭代。

关键在于**工作流的最小单位变了**：

- 传统工具的最小单位是"一次对话 / 一段补全"；
- ZCode 的最小单位是"**一个任务（Task）**"——围绕一个开发目标，从需求理解一路推进到验证收口。

1.3 五项核心能力

官方文档将 ZCode 的核心能力归纳为六点，本书整理为五项关键能力：

（1）GLM-5.3 深度联调

ZCode 围绕模型能力、工具调用和执行链路做了多轮打磨，使 Agent 更适合连续、多步骤的真实开发任务。GLM-5.3 侧的关键能力包括：

- **稳定的 1M 上下文**：目标不是"塞进更多 token"，而是在真实 coding agent 轨迹中保持可靠执行；
- **弹性推理强度（Flexible Effort）**：按任务难度在执行速度与计算成本之间权衡，对应 ZCode 中的"推理强度"开关；
- **面向长上下文的架构优化**：通过索引共享等机制降低长上下文计算开销，并优化推测解码；
- **面向 Agentic Coding 的强化训练**：后训练覆盖长轨迹、多轮工具调用、子任务分解与多轮环境反馈，并引入防投机（anti-hack）机制。

（2）长任务执行

在同一个任务里完成需求理解、计划拆解、代码修改、验证和 Review，减少频繁切换上下文带来的中断。

（3）稳定上下文保持

持续读取**文件、终端、浏览器、执行模式和 Git 状态**，把这些信息保持在同一个任务里。

（4）自研 Agent 工作流

任务、权限、上下文、工具调用和提交流都围绕 ZCode Agent 组织，让模型输出可以更自然地落到实际工程操作中。

（5）多端持续跟进 + 安全可控

桌面端、手机端 Remote 与飞书 / 微信 Bot 可以共同推进同一个工作区任务；同时关键命令、文件修改和高权限操作会在执行前进入确认流程。

1.4 与同类工具的差异

维度	ZCode	命令行 Agent（Claude Code / Codex）	AI 编辑器（Cursor）
产品形态	桌面端 ADE	终端 CLI	编辑器分支
核心模型	GLM-5.3 深度原生适配，支持 BYOK 接入第三方	绑定单一厂商模型	多模型切换
长任务	`/goal` 目标模式自动拆解、迭代、校验直至完成	需人工逐步确认	长任务连续性较弱
上下文	1M Token 长上下文	依赖所选模型	依赖所选模型

维度	ZCode	命令行 Agent（Claude Code / Codex）	AI 编辑器（Cursor）
执行权限	四档执行模式分级	依赖工具自身配置	无系统化分级
前端验证	内置浏览器自动化，Agent 可点击、截图、验证	通常无	通常无
远程协作	手机 Remote + 微信 / 飞书 Bot	无原生移动端	无原生移动端
项目理解	Wiki 架构导读 + Memory 记忆	依赖用户提供	依赖用户提供



1.5 平台、版本与权益

支持的平台与安装包格式

系统	架构	安装包
macOS	Apple Silicon	`.dmg`（arm64）
macOS	Intel	`.dmg`（x64）
Windows	x64	`.exe`
Windows	ARM64	`.exe`
Linux	x64 / ARM64	`.AppImage` / `.deb` / `.rpm`（Beta）

新用户权益

首次使用 ZCode 可享 **5 天免费权益**，开箱即用、无需额外配置：

模型	每日体验额度
GLM-5.3	300 万 Token / 日
GLM-5.3-Flash	500 万 Token / 日
合计	**800 万 Token / 日**

> **注意**：额度仅在这 5 天内每日发放，5 天后失效，并非长期每天都有。具体可用模型、剩余额度与消耗情况以供应商页面实时展示为准。

订阅用户权益：闲时任务免费执行

订阅用户可以创建「闲时任务」：把不着急的工作交给队列，ZCode 在算力富余时段免费完成，**全程不消耗套餐额度。**

GLM Coding Plan 档位

档位	定位	用量
Lite	小仓库轻量迭代	基准额度
Pro	中型仓库日常开发	约 Lite 的 6 倍
Max	中大型仓库高阶使用	约 Lite 的 14 倍

> **注意**：新版套餐用量按**积分**计算，并新增每周用量额度。各档位具体权益、积分额度与折算规则以官方套餐说明为准。

1.6 本章小结

- ZCode 是 ADE，不是编辑器插件；工作单位是"任务"而非"对话"。
- 五项核心能力：深度联调、长任务执行、上下文保持、自研 Agent 工作流、多端跟进与安全可控。
- 1M 上下文 + 目标模式 + 四档执行权限，共同支撑"长任务连续推进"。

1.7 思考与练习

1. 请用自己的话说明：为什么说 ZCode 把人的角色从"编写者"变成了"审核者"？
2. 你的团队里哪些任务属于"一句话说得清、但要多轮才能做完"？列出三个，它们将是第 5 章目标模式的练手材料。

第 2 章　安装与首次启动

2.1 本章目标

- 完成 macOS / Windows / Linux 三平台的安装
- 完成首次启动设置与登录
- 掌握代理配置与常见安装故障的排查

全程约 2 分钟。

2.2 下载

官方下载入口：`https://zcode.z.ai/cn`，安装包托管于官方 CDN `cdn-zcode.z.ai`。

支持 **macOS**（Apple Silicon / Intel）、**Windows**（x64 / ARM64）、**Linux**（x64 / ARM64，AppImage、DEB、RPM）。

> **建议**：macOS M 系列芯片务必下载 arm64 版本，否则会以 Rosetta 模式运行，长任务执行时性能明显下降。

2.3 安装步骤

2.3.1 macOS

1. 打开下载的 **ZCode.dmg** 安装镜像；
2. 将 **ZCode.app** 拖动到 **Applications** 文件夹；
3. 在启动台中找到 ZCode 并启动。

2.3.2 Windows

1. 下载 **ZCode** 安装程序；
2. 双击运行，按照安装向导完成安装；
3. 从开始菜单或桌面快捷方式启动 ZCode。

2.3.3 Linux

1. 在下载区选择 **Linux**，下载 `.AppImage`、`.deb` 或 `.rpm` 安装文件；
2. 使用 AppImage 时，为文件添加可执行权限：

```
chmod +x ZCode-*.AppImage
```

使用 `.deb` 或 `.rpm` 时，通过发行版的软件包管理器安装；

3. 双击文件或在终端中运行它启动 ZCode。

2.4 首次启动



首次启动设置：开始使用 ZCode 或数据迁移向导　　登录页面：连接 Z.ai / BigModel 或使用 API Key

首次启动会进入「首次启动设置」页面，完成后在左下角点击 **连接使用** 进入登录页面。

第 1 步：选择直接进入或数据迁移

- 点击 **开始使用 ZCode** 直接进入；
- 如需迁移历史数据，选择 **数据迁移向导。**

> **注意**：迁移向导当前仅支持导入 **Claude Code** 和 **旧版 ZCode 中 ZCode Agent** 的对话记录，不支持其他工具；也可以先跳过，稍后在设置中继续迁移。

第 2 步：选定项目目录作为工作区

进入后选择一个项目目录作为工作区。ZCode 会持续读取该目录下的文件、终端、浏览器、执行模式和 Git 状态。

第 3 步：发一条简单指令验证

在对话框输入一句简单指令，比如让 Agent **列出当前目录的文件**，确认响应正常。

如果还没有配置模型，点击左下角 **连接使用** 进入登录页面，引导你 **连接 Z.ai**、**连接 BigModel** 或 **使用 API Key** 完成接入（详见第 3 章）。

2.5 网络代理配置



如果你的网络需要走代理才能访问模型服务，在 **设置** → **常规** 里配置。有三个字段，填完点保存，**需要重启应用才生效。**

HTTP 代理：填入代理地址，例如 `http://127.0.0.1:7890`。

> **注意**：留空不等于跟随系统。这个字段为空时 ZCode 直连，**不会读取** `HTTP_PROXY` **等系统环境变量**。这一点和很多命令行工具的默认行为相反，如果你发现终端里能联网而 ZCode 不行，多半就是这个原因。

配置后，模型请求、MCP 服务、Agent 执行的命令工具以及应用界面的网络请求都会走这个代理。

代理例外：不想走代理的地址，用英文逗号分隔：

```
localhost,127.0.0.1,::1,.example.com,*.corp.com
```

主机名、`.example.com` 这样的域名后缀、`*.corp.com` 通配写法都支持，需要时还能带上端口。

自定义证书：填 PEM 格式根证书的本地文件路径。企业内网如果用了会解密 HTTPS 的安全网关，它签发的证书默认不被信任，把对应的根证书填在这里即可。

> **注意**：这不是跳过证书校验——ZCode 只放行证书链里确实包含这张根证书的连接，其余照常校验。

2.6 Windows 专属设置

选择命令行环境

在 **设置** → **常规** → **终端** 里可以指定 Agent 执行命令时用哪个 shell，可选 **自动选择**、**CMD** 和 **Git Bash**（检测到已安装时才出现）。默认是自动，优先用 Git Bash，找不到就回退到 `cmd.exe`。

> **注意**：改动**只对新建会话生效**。

关闭窗口时最小化到托盘

从 v3.4.0 起这是默认行为：点窗口的关闭按钮不会退出应用，而是收进系统托盘继续运行，定时任务和闲时任务也因此能在后台照常推进。托盘图标右键有新建任务、打开工作区、检查更新、退出等菜单项，左键点击可以把窗口叫回来。

想让关闭按钮直接退出，在 **设置** → **常规** → **通知** 里关掉「关闭窗口时隐藏到托盘」即可。这个开关只有 Windows 有。

> **注意**：从旧版本升级上来的用户请留意，这次默认值调整会把该开关**重置为开启一次**，即使你之前手动关过。再关一次之后就会一直保持你的选择。

2.7 故障排查

症状	原因与处理
macOS 提示「ZCode 已损坏，无法打开」	终端执行 `xattr -dr com.apple.quarantine /Applications/ZCode.app` 后重新打开
Windows 安装时被防火墙 / 杀软拦截	暂时关闭实时防护或将安装目录加入白名单后重试
Linux 安装文件无法运行	确认已添加可执行权限（`chmod +x`）；部分发行版需要额外安装 `fuse` 依赖
Linux / WSL 下的登录与输入法问题	参阅官方《Linux / WSL 排查指南》
终端能联网但 ZCode 不行	HTTP 代理字段留空时不读系统环境变量，需在设置页显式填写

2.8 本章小结

- 安装本身很简单，真正的门槛在**模型连接**（第 3 章）与**代理配置**。
- 代理不继承系统环境变量、修改后需重启，是最常见的"连不上"原因。
- Windows 用户需要特别关注托盘行为与终端 shell 选择。

2.9 思考与练习

1. 在你的机器上安装 ZCode，并成功发出第一条指令（让 Agent 列出当前目录文件）。
2. 如果你所在网络需要代理，完成代理配置并说明为什么"终端能连、ZCode 不能连"是正常现象。

第 3 章　连接模型与套餐

3.1 本章目标

- 掌握三种接入路径与各自适用场景
- 正确区分 Coding 专用端点与通用端点
- 理解最大输出、上下文窗口、思考强度、图片输入四类高级参数

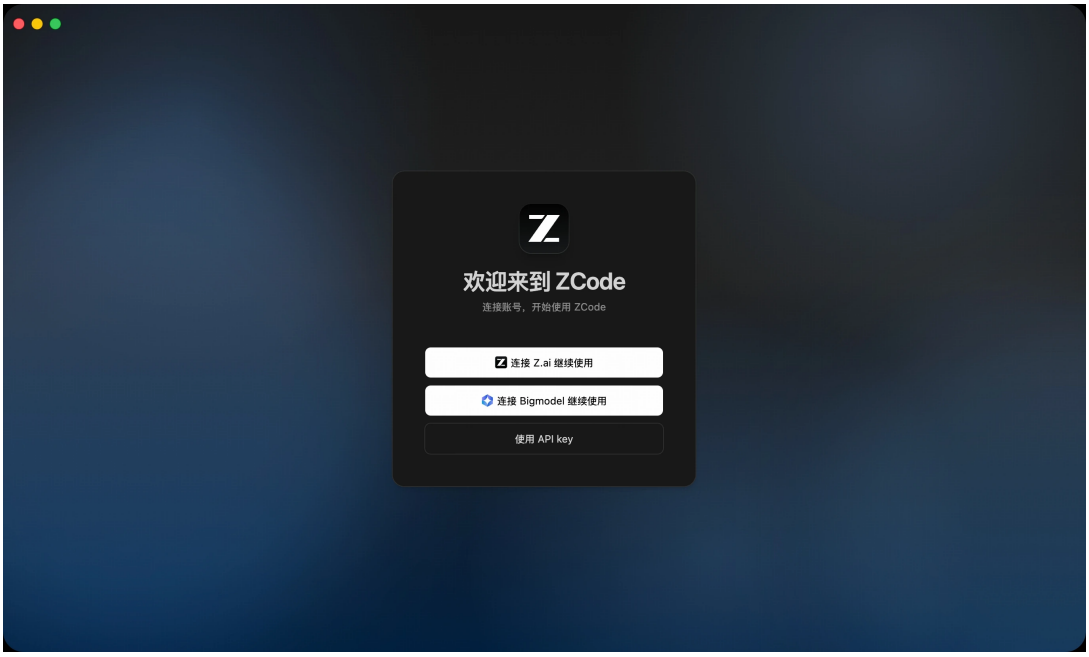
3.2 三种接入路径

方式	适合谁	特点
连接 Z.ai 账号	海外用户	美元计价，账号登录即用
连接 BigModel 账号	国内用户（推荐）	账号体系与国内手机号绑定，直接访问
使用 API Key	自行管理模型访问者	可接入任意兼容 Anthropic / OpenAI 协议的服务

> **建议**：国内用户推荐通过 **GLM Coding Plan（BigModel）** 在 ZCode 中运行 GLM-5.3，账号登录即可使用，配置最简单。

3.3 配置入口

方式一：首次启动欢迎页



首次打开 ZCode、还没有可用模型时，欢迎页面会直接提供接入选项：

- **连接 Z.ai 继续使用**：通过 Z.ai 账号授权登录；
- **连接 BigModel 继续使用**：通过智谱 BigModel 账号授权登录；
- **使用 API Key**：直接填写 API Key，进入模型供应商配置。



方式二：模型选择器



进入 ZCode 后，点击对话框中的模型名称打开模型选择器，在列表底部点击「管理模型」，即可进入 **设置** → **模型设置** 面板。

3.4 端点地址：最容易填错的地方

在 **BigModel** 或 **Z.ai** 供应商页切换到 **API Key** 模式后，会出现 **OpenAI 接口地址** 与 **Anthropic 接口地址** 两个字段。先选账号类型，再按平台填写对应地址。

3.4.1 GLM Coding Plan 编程套餐（最常用）

平台	OpenAI 接口地址	Anthropic 接口地址
BigModel	`https://open.bigmodel.cn/api/coding/paas/v4`	`https://open.bigmodel.cn/api/anthropic`
Z.ai	`https://api.z.ai/api/coding/paas/v4`	`https://api.z.ai/api/anthropic`

3.4.2 资源包 / 充值余额

平台	OpenAI 接口地址
BigModel	`https://open.bigmodel.cn/api/paas/v4`
Z.ai	`https://api.z.ai/api/paas/v4`

> **注意（非常重要）**：
>
> - Coding Plan 的 OpenAI 地址必须是 **Coding 专用端点**（`/api/coding/paas/v4`），**不要**填通用端点 `/api/paas/v4`；两者不可互相替代，也不会互相消耗额度。
> - **Anthropic 接口不适用于资源包 / 余额场景**：只有从未购买过 Coding Plan 的账号，走 Anthropic 接口时才会消耗余额；只要买过套餐（无论是否用完、是否到期），该接口都不会走余额。
> - 若通过右上角 **「编程套餐」** 授权绑定账号（非 API Key 模式），ZCode 会自动选择正确通道，无需手动填写。

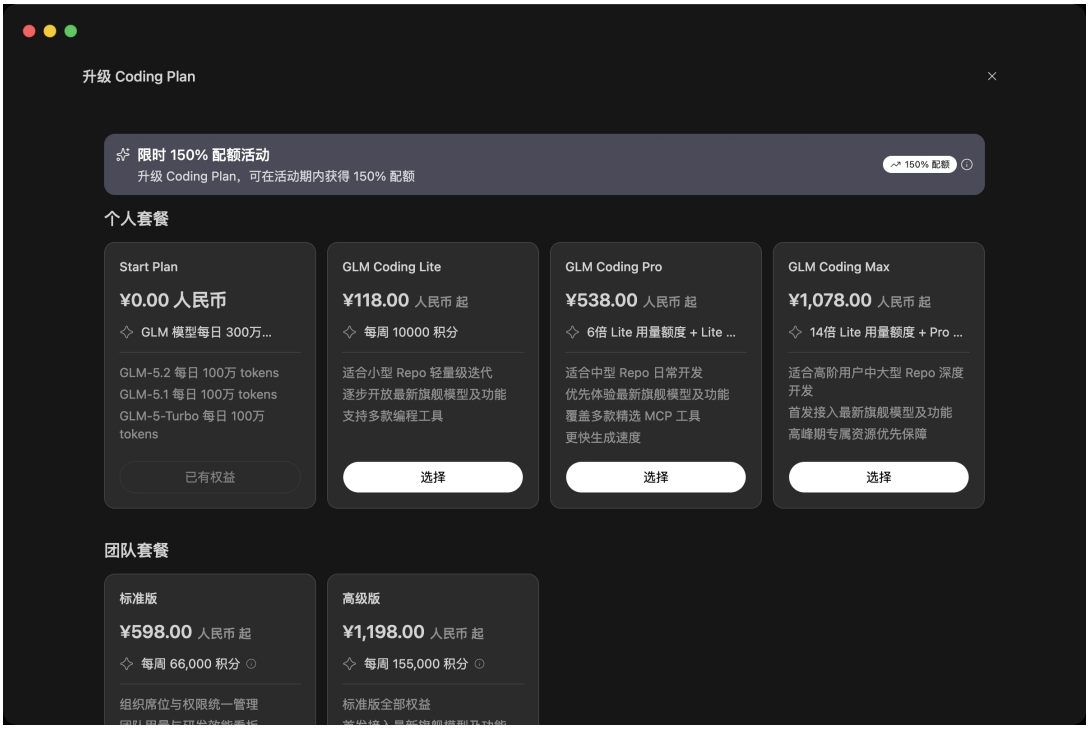
3.5 连接 BigModel 的完整流程



1. 通过上述任一方式进入 **模型设置** 面板；
2. 在左侧供应商列表中选择「BigModel」；

3. 完成账号连接并开启启用开关，即可按账号权限使用 GLM-5.3、GLM-5.3-Flash 等内置模型；
4. 右上角可切换连接方式：使用「编程套餐」绑定 GLM Coding Plan，或改用 API Key 接入。

3.5.1 使用团队套餐

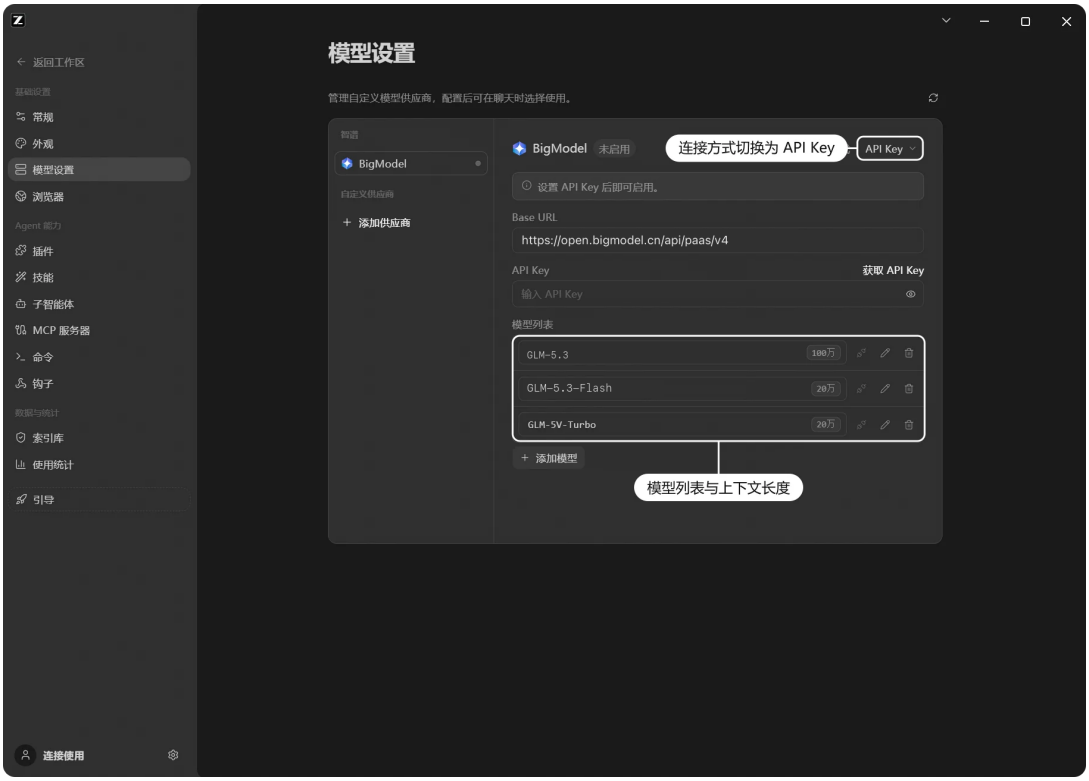


如果你所在的团队买了 GLM Coding Plan 团队版，登录后不需要额外配置：在 BigModel 供应商页面的 **连接方式** 菜单里，你有权限的每个团队会各自显示成一个 **团队套餐** 条目（按组织名称命名），选中即用该团队的额度跑任务。同一个菜单里也能切回 **个人套餐** 或 API Key。

> **注意**：菜单里看到团队条目却提示「团队套餐未分配」，说明管理员还没给你分配席位——**席位的添加和管理需要到 BigModel 的团队套餐管理页完成**，客户端里做不了。

3.5.2 用 API Key 接入 BigModel

方式 A：GLM Coding Plan（编程套餐 API Key）

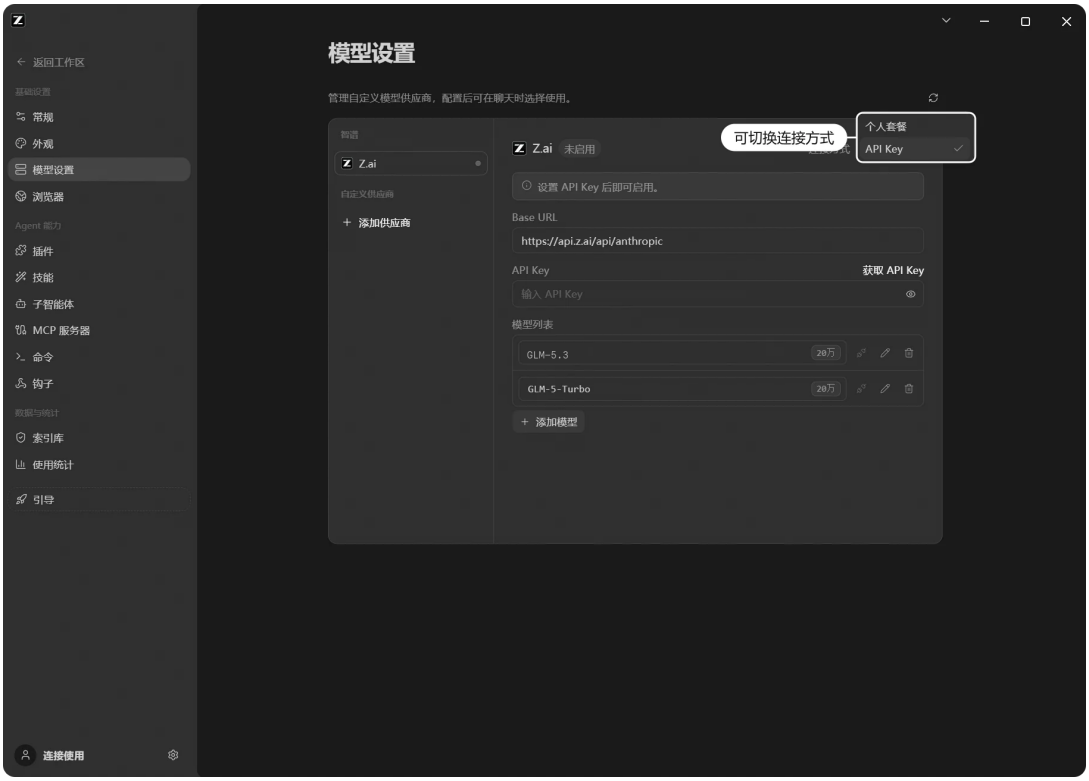


1. 在 BigModel 供应商页面右上角，将连接方式切换为「API Key」；
2. **OpenAI 接口地址**填写 Coding 专用端点：`https://open.bigmodel.cn/api/coding/paas/v4`；
3. 填入从智谱开放平台获取的 API Key；
4. 可用模型以账号权限和供应商返回的模型列表为准。

方式 B：模型资源包 / 充值余额

1. 切换为「API Key」；
2. 使用 **OpenAI 协议**，地址填写 `https://open.bigmodel.cn/api/paas/v4`；
3. 填入 API Key；
4. 可用模型以账号权限和供应商返回的模型列表为准；也可以点击「添加模型」补充其他可用模型。

3.6 连接 Z.ai（海外）



配置流程与 BigModel 基本一致：

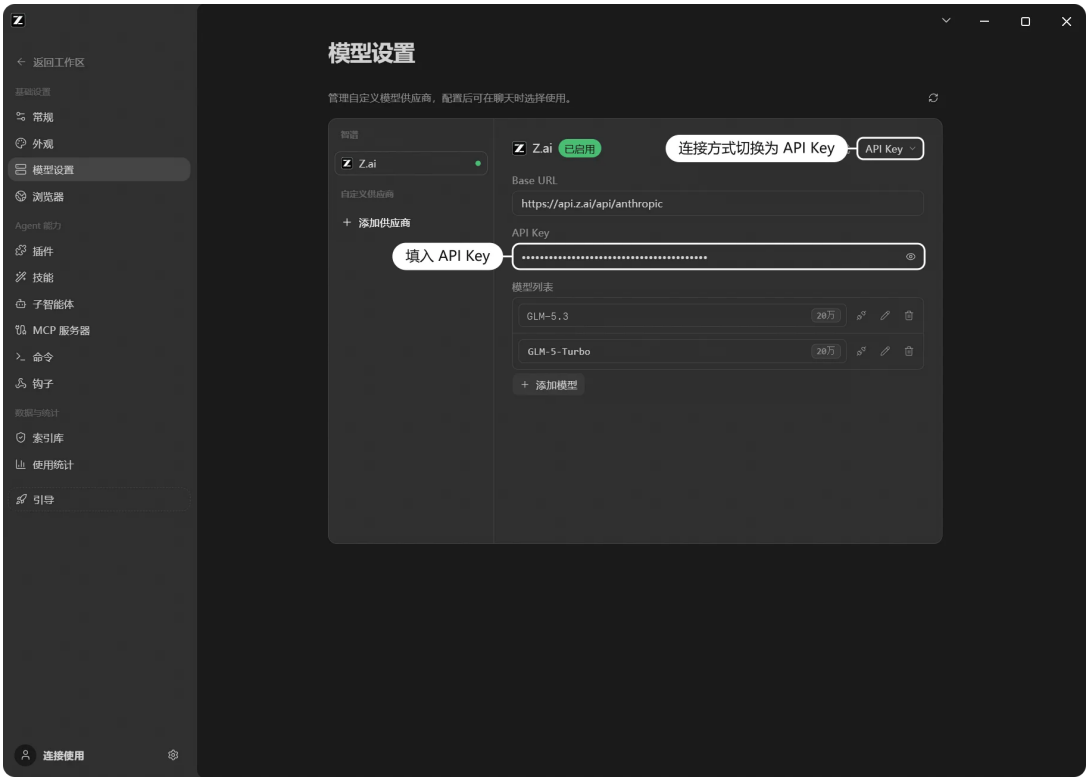
1. 进入 **模型设置** 面板；
2. 在左侧供应商列表中选择「Z.ai」；
3. 完成账号连接并开启启用开关；
4. 右上角在「编程套餐」与「API Key」两种连接方式之间切换。



对应端点：

场景	地址

场景	地址
Coding Plan · OpenAI	`https://api.z.ai/api/coding/paas/v4`
Coding Plan · Anthropic	`https://api.z.ai/api/anthropic`
资源包 / 余额 · OpenAI	`https://api.z.ai/api/paas/v4`



3.7 接入第三方与自定义供应商

ZCode 支持添加任何兼容 **Anthropic / OpenAI 协议** 的模型服务作为自定义供应商，既可以是公网模型服务，也可以是团队统一维护的企业模型通道或内网自托管服务。

配置步骤（通用）：

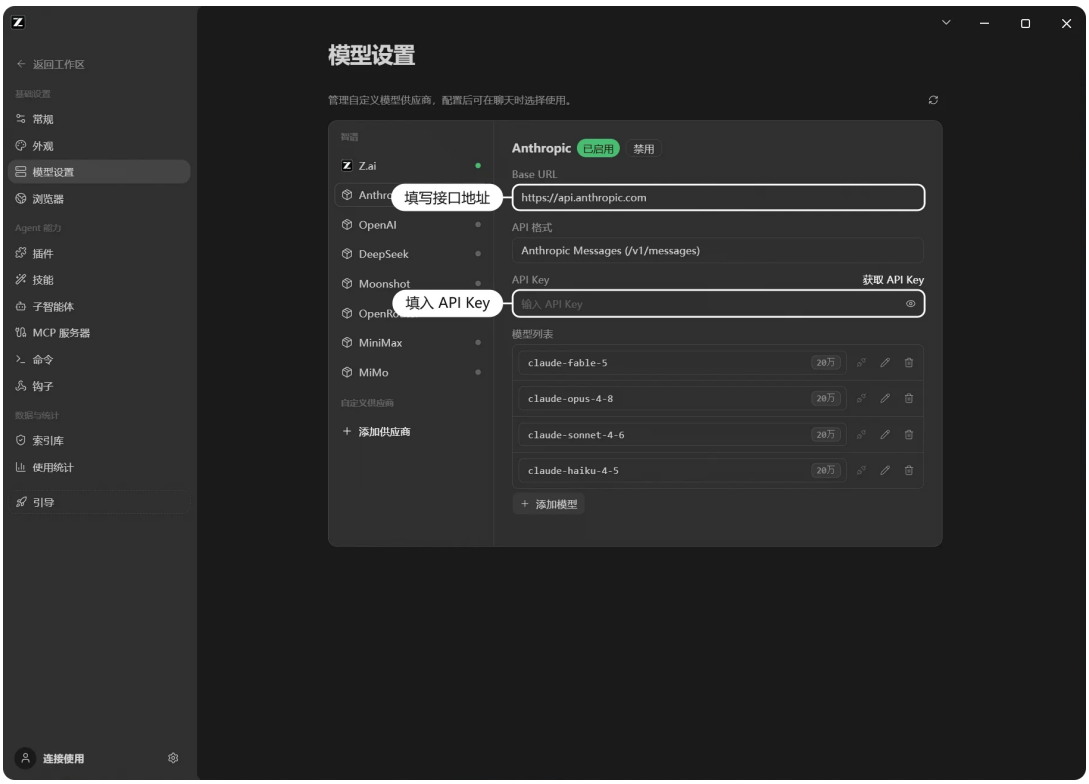
1. 进入 **模型设置** 面板；
2. 在左侧供应商列表底部点击「添加供应商」；
3. 自定义填写名称（如 claude、deepseek 等）；
4. 从下拉列表中选择或手动输入 API 基础 URL；
5. 填写 API Key；
6. 点击「添加模型」，手动填写该服务支持的模型 ID；
7. 开启启用开关后即可开始使用。

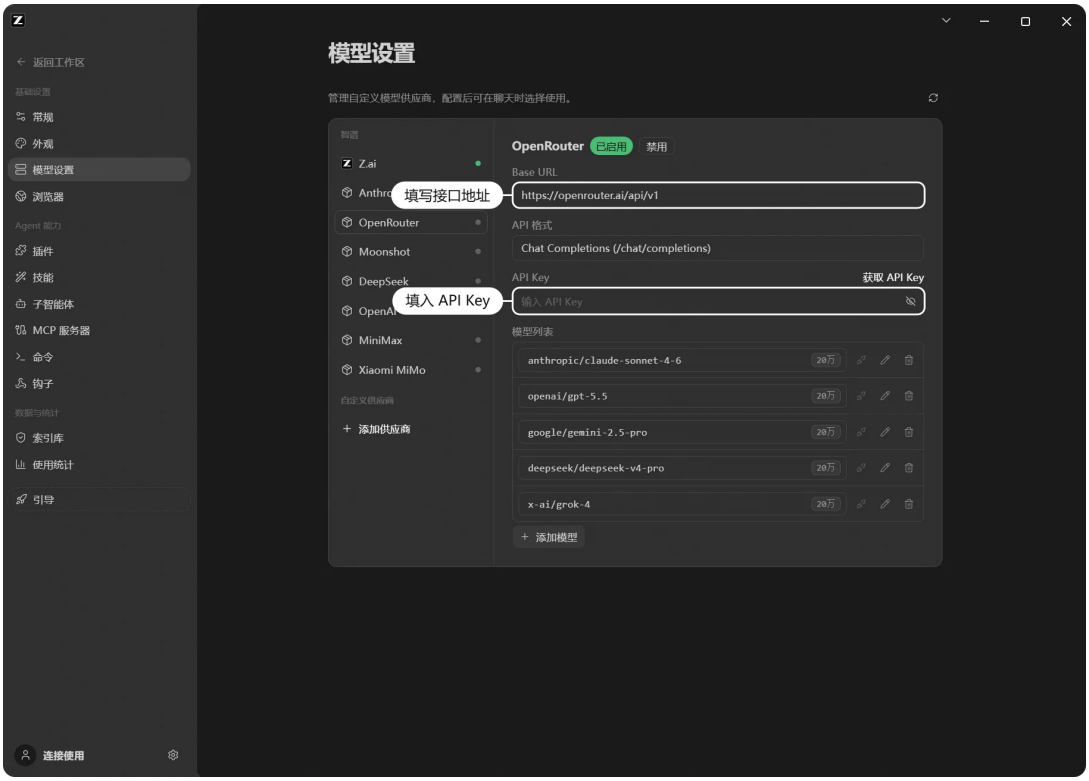
常用供应商参数速查表：

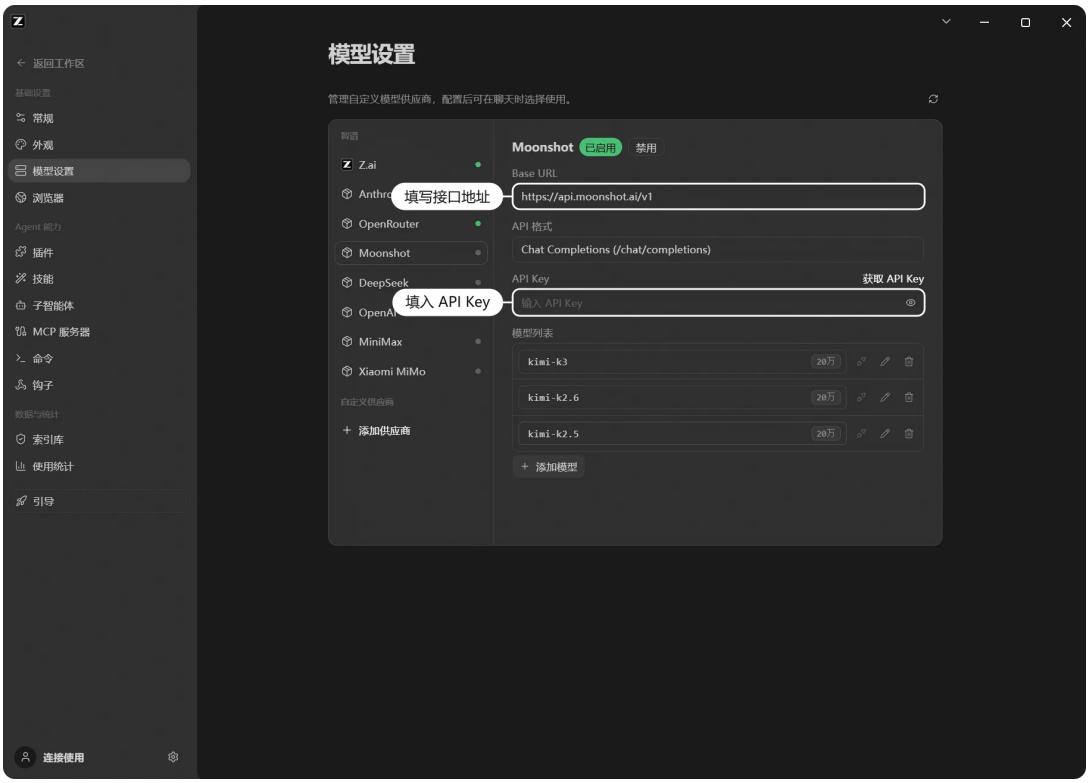
供应商	名称填写	接口地址

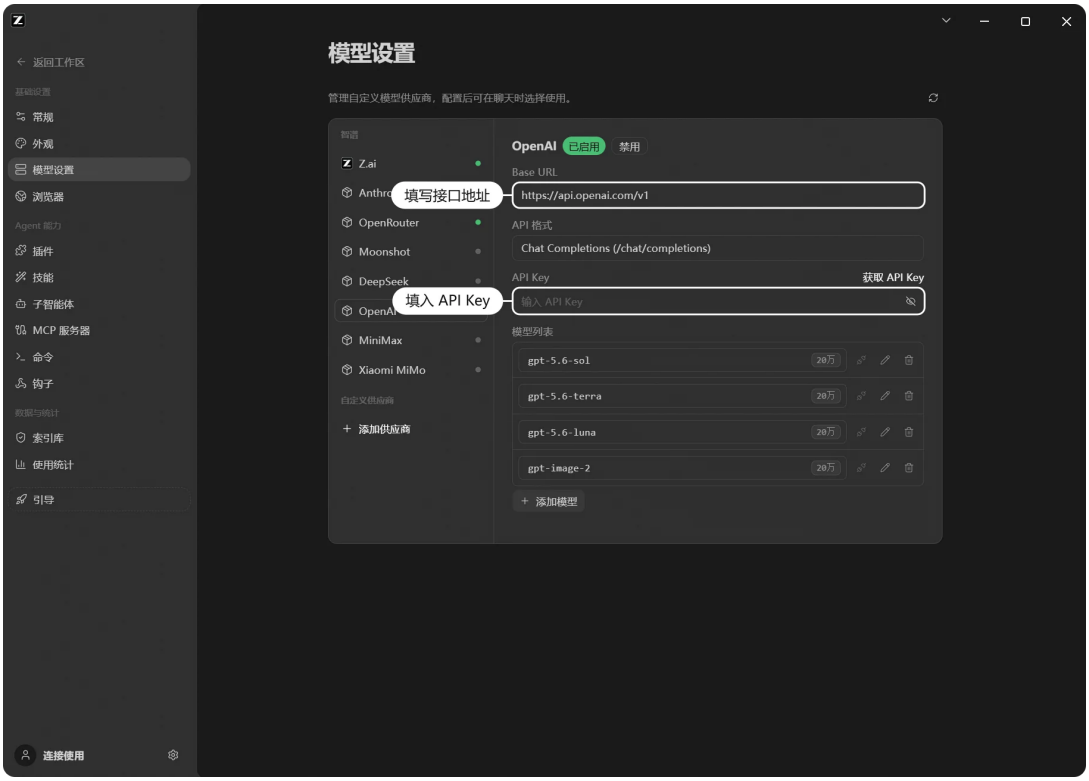
供应商	名称填写	接口地址
Anthropic（Claude API）	`Anthropic`	Anthropic 接口：`https://api.anthropic.com`
OpenRouter	`OpenRouter`	API 基础 URL：`https://openrouter.ai/api`
Moonshot	`Moonshot`	Anthropic 接口：`https://api.moonshot.cn/anthropic`
OpenAI	`OpenAI`	API 基础 URL：`https://api.openai.com`
MiniMax	`MiniMax`	Anthropic 接口：`https://api.minimaxi.com/anthropic`
小米 MiMo	`Xiaomi MiMo`	API 基础 URL：`https://api.xiaomimimo.com/v1`
DeepSeek（示例）	`DeepSeek`	Anthropic：`https://api.deepseek.com/anthropic`；OpenAI：`https://api.deepseek.com/v1`

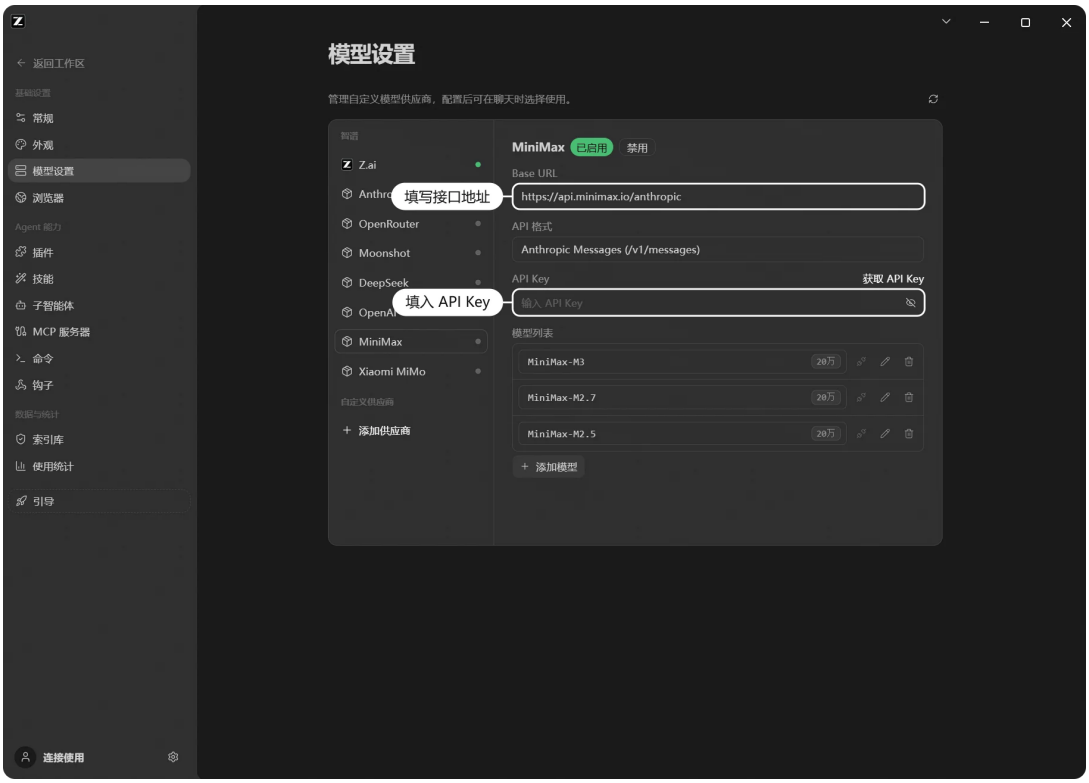
下面按上表顺序给出各供应商的配置界面示意。

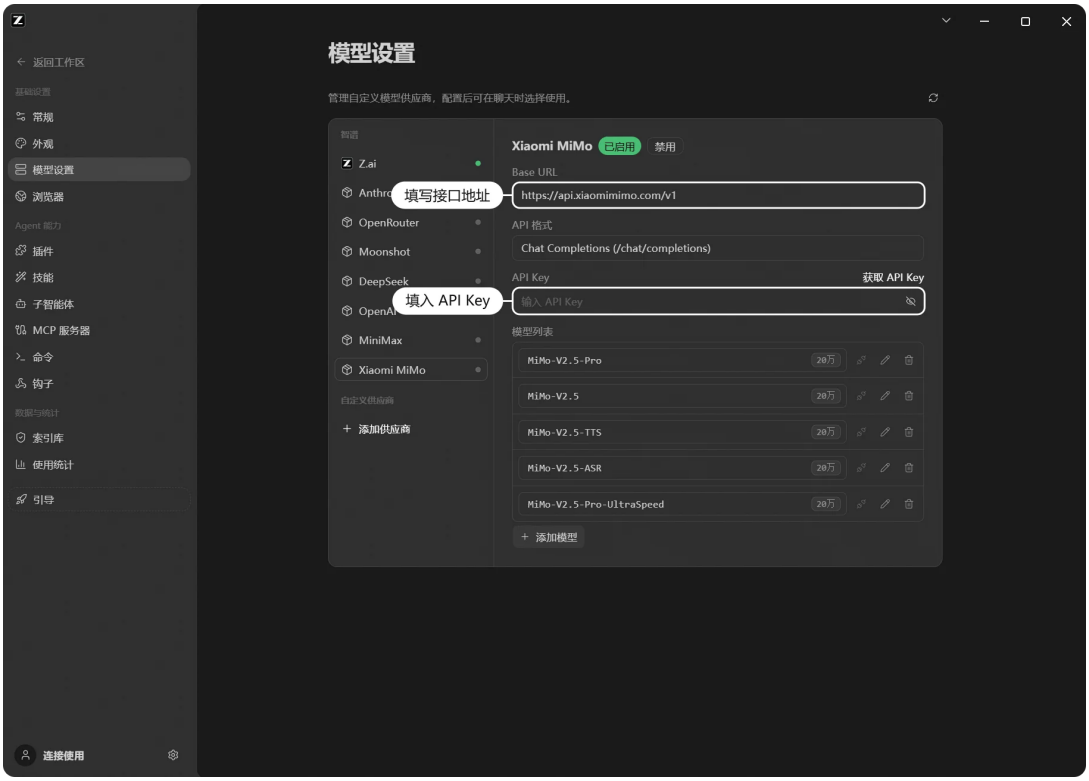


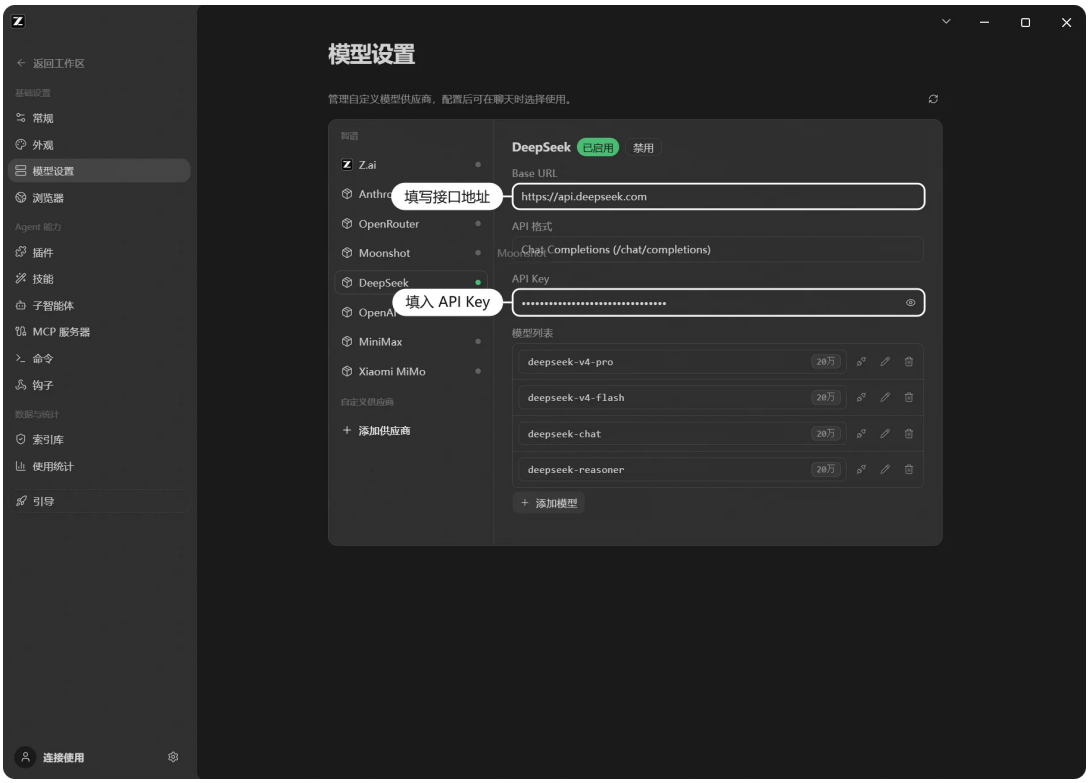

返回工作区
基础设置
常规
外观
模型设置
浏览器
Agent 能力
插件
技能
子智能体
MCP 服务器
命令
钩子
数据与统计
索引库
使用统计
引导
连接使用
模型设置
管理自定义模型供应商，配置后可在聊天时选择使用。
智谱
Z.ai
Anthro
OpenRouter
Moonshot
DeepSeek
OpenA
MiniMax
Xiaomi MiMo
自定义供应商
添加供应商
填写接口地址
填入 API Key
OpenRouter
已启用
禁用
Base URL
https://openrouter.ai/api/v1
API 格式
Chat Completions (/chat/completions)
API Key
获取 API Key
输入 API Key
模型列表
anthropic/claude-sonnet-4-6
20万
openai/gpt-5.5
20万
google/gemini-2.5-pro
20万
deepseek/deepseek-v4-pro
20万
x-ai/grok-4
20万
添加模型


返回工作区
基础设置
常规
外观
模型设置
浏览器
Agent 能力
插件
技能
子智能体
MCP 服务器
命令
钩子
数据与统计
索引库
使用统计
引导
连接使用
模型设置
管理自定义模型供应商，配置后可在聊天时选择使用。
智谱
Z.ai
Anthro
OpenRouter
Moonshot
DeepSeek
OpenA
MiniMax
Xiaomi MiMo
自定义供应商
添加供应商
填写接口地址
填入 API Key
Moonshot
已启用
禁用
Base URL
https://api.moonshot.ai/v1
API 格式
Chat Completions (/chat/completions)
API Key
获取 API Key
输入 API Key
模型列表
kimi-k3
20万
kimi-k2.6
20万
kimi-k2.5
20万
添加模型


返回工作区
基础设置
常规
外观
模型设置
浏览器
Agent 能力
插件
技能
子智能体
MCP 服务器
命令
钩子
数据与统计
索引库
使用统计
引导
连接使用
模型设置
管理自定义模型供应商，配置后可在聊天时选择使用。
智谱
Z.ai
Anthro
OpenRouter
Moonshot
DeepSeek
OpenAI
MiniMax
Xiaomi MiMo
自定义供应商
添加供应商
填写接口地址
填入 API Key
OpenAI
已启用
禁用
Base URL
https://api.openai.com/v1
API 格式
Chat Completions (/chat/completions)
API Key
获取 API Key
输入 API Key
模型列表
gpt-5.6-sol
20万
gpt-5.6-terra
20万
gpt-5.6-luna
20万
gpt-image-2
20万
添加模型


返回工作区
基础设置
常规
外观
模型设置
浏览器
Agent 能力
插件
技能
子智能体
MCP 服务器
命令
钩子
数据与统计
索引库
使用统计
引导
连接使用
模型设置
管理自定义模型供应商，配置后可在聊天时选择使用。
智谱
Z.ai
Anthro
OpenRouter
Moonshot
DeepSeek
OpenAI
MiniMax
Xiaomi MiMo
自定义供应商
添加供应商
填写接口地址
填入 API Key
MiniMax
已启用
禁用
Base URL
https://api.minimax.io/anthropic
API 格式
Anthropic Messages (/v1/messages)
API Key
获取 API Key
输入 API Key
模型列表
MiniMax-M3
20万
MiniMax-M2.7
20万
MiniMax-M2.5
20万
添加模型





> **建议**：企业模型通道建议由团队统一管理 Base URL、API Key、模型列表与访问权限，保证长任务执行过程中的连接稳定性和可追踪性。

3.8 单模型高级参数

在 **设置** → **模型供应商** 里点开某个模型，展开 **高级**，可以为这一个模型单独设置参数。

3.8.1 最大输出 Token

即模型单次回复的长度上限。**默认留空**，跟随模型自己支持的上限，通常保持留空就好。只有当你确定某个模型能输出更长内容、而 ZCode 没有识别到时，才需要手动调大。

> **注意**：输出空间和历史对话共用同一个上下文，输出上限调得越高，能带上的历史对话就越少，自动压缩也会来得更早。

3.8.2 上下文窗口：哪些能改、哪些不能

模型类型	是否可改
自定义供应商的模型	可以修改，保存后对新会话生效
编程套餐（Coding Plan / Start Plan）的内置模型	由服务端统一下发，本地修改会在下一次同步时恢复为官方值（有意设计）
名称以 `[1m]` 结尾的模型	固定按 1M 上下文处理，不可修改

自动上下文压缩会在接近窗口上限前提前触发：系统会为模型输出预留空间并保留安全缓冲（合计约 3.4 万 token），因此压缩点低于窗口标称值属于正常现象——

- 128K 窗口：约在 **9.4 万 token** 时触发；
- 1M 窗口：约在 **96.6 万 token** 时触发。

自动压缩目前**没有可配置的开关或阈值。**另外，输入框上方水位面板的分项占比是本地估算值，与顶部的真实用量口径不同，两者不能直接相加对齐。

3.8.3 思考强度（推理档位）

ZCode 会根据当前模型的实际能力显示可用的思考强度，不同模型的档位并不相同：

模型	可选档位	默认档位
GLM-5.3	`low` / `high` / `max`	`max`
GLM-5.2	`nothink` / `high` / `max`	`max`
GPT 系列	`low` / `medium` / `high` / `xhigh`	`medium`
Claude 系列	`low` / `medium` / `high` / `xhigh`（Opus 4.7 另有 `max`）	`medium`
Kimi K3（`kimi-k3` / `k3` / `k3-256k`）	`low` / `high` / `max`	`max`
DeepSeek V4 系列	`high` / `max`	`max`
其他自定义模型	「开启 / 关闭」两档，或不提供档位	由模型配置决定

底层参数转换：OpenAI 兼容接口使用顶层 `reasoning_effort`，Anthropic 兼容接口使用 `thinking` / `effort`。

> **注意**：**同名模型的第三方部署，接受的档位可能与官方端点不一致。**例如 DeepSeek V4 系列在 ZCode 中的最高档按官方端点约定发送 `max`，而部分第三方部署只接受 `xhigh` 及以下——此时选最高档会返回 400 参数错误，改选 `high` 档即可正常使用。

供应商私有思考参数的支持情况：

- 已适配：Qwen 系列的 `enable_thinking`、`thinking.type=enabled/disabled` 形式的开关；
- 不支持在 ZCode 内配置：GLM 的 `clear_thinking`、MiniMax 的 adaptive 思考模式；
- **不支持**为第三方模型追加自定义请求参数：`~/.zcode/v2/config.json` 中供应商的 `options` 只识别 `apiKey`、`baseURL`、`apiKeyRequired`、`headers` 等连接参数，手动加入的其他字段不会写入请求体，也不会有报错提示。

3.8.4 图片输入的支持判定

ZCode 会综合供应商与模型配置、模型目录中的能力信息、内置模型规则，以及当前接口协议，判断所选模型能否接收图片。判断结果分三种：

判定	行为
支持	保留图片，并随请求发送给模型服务
不支持	在请求发出前移除图片数据，并用文字提示替代
能力未知	不提前拦截，由模型服务最终决定是否接受（服务端不支持可能直接报错）

通过编程套餐连接 BigModel / Z.ai 时，内置的 **GLM-5.3-Flash** 是面向截图理解、看图分析和视觉问答的多模态模型，选中它后发送图片会直接按「支持」处理。

> **注意**：模型 ID 会参与能力匹配，但不是唯一依据。例如 GLM-5.2 通过官方 Anthropic 兼容接口接入时可保留图片交由服务端处理；通过 OpenAI Chat 兼容接口接入时则按不支持图片处理。遇到图片相关报错时，建议先使用供应商文档中的标准模型 ID，并确认当前接口本身支持图片输入。

3.9 HTTP 代理的作用范围

走代理	不走代理
模型 API 请求	桌面端的账号登录与后端服务请求
MCP 服务器（stdio / HTTP / SSE）	仓库 Wiki 生成
WebFetch 网页抓取	Web 远程控制通道（WebSocket）
终端命令子进程（注入 `HTTP_PROXY` 等环境变量）	SSH 远程连接
插件市场	—
应用内页面加载	—

两点注意：

1. ZCode 默认**不读取系统的 `HTTP_PROXY` / `HTTPS_PROXY` 环境变量**（仅 WebFetch 在未配置代理时会兜底使用），以设置页里填写的代理为准；
2. 修改代理设置后需要**重启 ZCode** 才会对全部链路生效。

3.10 验证配置

配置完成后，在对话框的模型选择器中选择对应通道，发送一句简单指令测试。确认模型可用、响应稳定即可开始使用。

3.11 本章小结

- 国内走 BigModel、海外走 Z.ai，是最省事的两条路；API Key 方式适合自建通道。
- **Coding 专用端点 ≠ 通用端点**，填错会导致额度不走套餐或调用失败。
- 上下文窗口、思考强度、图片能力三类参数都"因模型而异"，需要先确认模型类型再调。

3.12 思考与练习

1. 你手上有一个已购 Coding Plan 的 BigModel 账号，请写出 API Key 模式下应填的完整 OpenAI 接口地址，并说明填错的后果。
2. 你的同事报告"选了最高推理档后接口返回 400"，请列出三种可能原因与排查顺序。

第 4 章　第一次对话：工作台与 ZCode Agent

4.1 本章目标

- 熟悉工作台布局与提问入口
- 掌握 `@` `#` `/` `$` 四类上下文补充方式
- 理解四档执行模式与三档推理强度
- 会用 `AGENTS.md` 长期约束 Agent 行为

4.2 工作台布局

区域	作用	新手应先看什么
左侧栏	工作区、任务列表、文件树、自动化入口	任务状态点与 `+/-` 变更行数
中间区	会话主区 / 预览面板 / Wiki	Agent 每一步做了什么
右侧面板	浏览器面板、Git 变更、辅助对话	终端输出与文件变更
底部终端	命令执行结果	是否有报错、验证命令是否通过

4.3 提问入口与"不打开项目也能对话"

ZCode Agent 是 ZCode 自研的默认智能体，也是新建任务时最直接的提问入口。

有时候你只是想问个问题、让 Agent 帮忙算点东西或起草一段文字，并不涉及某个代码仓库。这种情况不必先选文件夹：在输入框上方的工作区菜单里选择 **不在项目中工作**，或者点侧栏「任务」分区右侧的新建按钮，就能直接开始对话。

> **注意**：
>
> - 这类对话里 Agent 创建的文件会留在磁盘上，关掉对话也不会自动清理，需要时请自己删；
> - 对话一旦发出就固定在这个模式里，想换成项目对话得新建一个；
> - 这个模式下不显示 Git 分支选择器。

4.4 补充上下文：四个符号



在聊天输入框左下角点击「+」，可以快速为当前任务补充上下文；也可以直接在输入框中键入对应符号触发相同能力。

入口	触发符号	作用
添加附件	—	上传截图、文档、需求材料等文件，作为当前任务的上下文
插入 @ 提及	`@`	引用工作区中的文件或整个文件夹，让 Agent 精准定位到相关代码
插入 # 会话	`#`	关联历史对话，把已有任务的上下文带入当前任务

入口	触发符号	作用
插入 / 命令	`/`	调用已保存的命令，快速复用固定提示词或流程
调用技能	`$`	调用已配置的 Skill（详见第 15 章）

> **建议**：先描述目标、再用 `@`、`#`、`/`、`$` 精准补充上下文，通常能让 ZCode Agent 更快理解并完成任务。

粘贴大段文字时不用担心刷屏——超过一定长度的内容会**自动转成附件**，输入框里只留一个附件条目，完整内容照样传给 Agent。

4.5 划选对话内容追问

对话里的任何一段文字都可以直接拿来当上下文。用鼠标选中助手回复、推理过程、代码或工具结果中的一段话，选区旁会浮出一个小工具条，点 **添加到当前任务**，这段文字就会作为一条引用追加到输入框。

> **注意**：引用的长度和条数有上限，超出时会直接提示，不会悄悄截断；目前只支持单行选区，跨行拖选不会出现工具条。

4.6 项目指令文件 `AGENTS.md`

如果希望长期约束 ZCode Agent 的行为，可以添加 `AGENTS.md` 指令文件。ZCode 会在启动任务时读取该文件，把其中的项目约定、编码规范、验证方式、注意事项等提供给 Agent。

来源	路径	适合写什么
用户全局指令	`~/.zcode/AGENTS.md`	跨项目通用的个人偏好、协作方式和常用约定
Workspace 指令	当前 Workspace 中的 `AGENTS.md`	当前项目专属的工程约定

拼接顺序：**先用户全局指令，再工作区指令**；工作区指令被视为当前任务的主要项目来源。两个来源都不存在时，不注入项目指令内容。

> **注意**：
>
> - `CLAUDE.md` **不作为** ZCode Agent 后续运行时持续读取的项目指令文件。对于已有 Claude Code 项目，ZCode 只会在 onboarding 阶段做一次性迁移，将已有内容复制到 `AGENTS.md`，迁移后以 `AGENTS.md` 为准。
> - ZCode 只读取上述两个来源，**不会合并多个层级的** `AGENTS.md`，也**不会**扫描子目录、展开 `@import` / `@include`，或根据任务类型自动选择规则文件。

`AGENTS.md` 适合写入的内容：

- 项目的技术栈、目录结构和关键模块说明；
- 代码风格、命名规范和提交前需要执行的验证命令；

- 对高风险文件、生产配置、权限操作的额外注意事项；
- 团队希望 Agent 遵守的协作方式（例如先给计划、少做无关重构、优先使用现有工具链）。

示例：

```
# 项目约定

## 技术栈
- 前端：React 18 + TypeScript + Vite
- 包管理器：pnpm（不要用 npm / yarn）
- 测试：Vitest

## 提交前必须执行
pnpm lint && pnpm test

## 注意事项
- 不要修改 src/legacy/ 下的文件
- 数据库迁移脚本必须可逆
- 生产配置文件位于 config/prod/，修改前必须先给出方案并等待确认
```

4.7 执行模式：四档权限



执行模式控制 ZCode Agent 在修改文件、执行命令前的确认策略。聊天输入框聚焦时可用 `Shift + Tab` 快速循环切换。

模式	工作方式	适用场景

模式	工作方式	适用场景
变更前确认	每次修改文件或执行命令前都先询问确认（默认档位）	关键代码、生产配置等高风险改动
自动编辑	自动编辑文件，命令等高风险操作仍需确认	常规迭代开发，希望减少编辑确认
计划模式	先检查代码并给出计划，确认后再开始编辑	需求复杂、步骤较多、希望先校准方案的任务
完全访问	编辑文件与执行命令时尽量减少确认次数	目标明确、风险较低、希望快速推进的任务

> **建议**：如果任务涉及关键文件、命令执行或大范围修改，优先使用计划模式或变更前确认；若只是常规编辑或明确的小任务，可以使用自动编辑或完全访问来减少打断。

4.8 推理强度：三档



在聊天输入框右下角、模型名称旁的思考图标处切换——这是 GLM-5.3 弹性推理强度能力在产品中的直接入口。

以 GLM-5.3 为例（默认为「最高」）：

档位	工作方式	适用场景
低	使用较低的推理强度，优先提升响应速度	简单问答、明确的小修改

档位	工作方式	适用场景
高	以较深的思考深度处理任务，兼顾响应速度	常规开发任务，速度与质量的平衡
最高	以最深的思考深度处理任务（默认档位）	复杂架构设计、疑难 Bug 排查、长任务推进

> **建议**：默认的「最高」档适合大多数复杂任务；追求更快响应时降到「高」，简单问答或明确的小修改切到「低」。

4.9 辅助对话

主任务正在推进时，你可能想问一句"这个函数是干嘛的"，又不希望打断它的思路。**辅助对话**在右侧边栏开一个独立的对话标签，与主对话并行、互不干扰。

打开方式

方式	说明
从划选进入	选中一段文字，浮层上点 **在辅助对话中提问**，该段文字作为引用一并带过去
快捷命令	主对话输入框输入 `/side` 或 `/btw`，两个命令作用相同，不会作为消息发送
`+` 菜单	右侧边栏顶部的 `+` 里选 **辅助对话**
启动页	右侧边栏还没有任何标签时，空面板上也有这个入口

它继承什么

辅助对话把主会话的历史作为模型上下文，可以直接追问，不必重复交代背景；但界面上从空白开始——时间线、搜索、翻页看到的都只是你在这里问过的内容。

它是一个功能完整的对话，能调用工具、修改文件、走权限确认，只是**不继承**主任务的目标、待发送队列和后台任务。

能力与限制

方面	说明
不支持的操作	目标模式、编辑重发、会话分叉
同一窗口	一个主任务同时只有一个辅助对话，再次点入口是回到原来那个
多个窗口	同一个主任务在每个窗口里都可以有各自的辅助对话
平台	仅桌面端，手机端不提供入口

> **注意（生命周期）**：切换到别的任务再切回来，辅助对话还在，草稿和引用都保留。但**关闭标签页、主任务被删除或归档、退出应用**这几种情况下，它会连同内容一起关闭，并且**无法恢复**。需要长期保留的讨论，应开一个正式任务。

4.10 会话分叉

想在某一步换个方向试试，又不想弄丢现在的进度，可以把会话**分叉**：把鼠标移到一条已经回答完的助手消息上，点操作行里的 **分叉**，ZCode 会以这条消息为起点创建一个新任务，标题是 `Fork of <原标题>`，任务行上带「从对话中派生」标记。

新会话继承：分叉点之前的对话历史、当时的模型与推理强度配置、目标模式的进度。**原会话完整保留，继续按原样运行。**

> **注意**：
>
> - 只能从**稳定完成**的助手消息分叉。被中断的、失败的半截回复没有这个入口，用户消息和工具结果也没有；
> - 分叉只影响对话历史，**不会回滚磁盘上的文件**，两个会话操作的是同一个工作区；
> - 待发送队列、正在执行的工作和后台任务都不会被复制到新会话；
> - 压缩进行中时无法分叉。

4.11 推荐的标准提问结构

官方文档给出的使用建议，可以整理为一个稳定的五步结构：

1. **先描述目标**：直接说明要实现、修复或分析的内容，让 Agent 建立任务目标；需要多轮迭代的长任务，使用目标模式（第 5 章）；
2. **再补充上下文**：用 `@` 引用关键文件，或通过附件补充截图、文档和需求材料；
3. **用命令和技能提效**：固定流程用 `/` 调用命令，复用能力用 `$` 调用技能；
4. **根据风险选择执行模式**：普通编辑用高效模式；涉及关键文件或提交前检查时用计划模式先校准方案；
5. **保持同一任务连续推进**：同一任务里持续追问、补充约束、查看改动并完成提交前确认，避免把上下文拆散。

一个高质量的提示词示例：

```
请帮我修复登录页在手机端按钮被遮挡的问题。

范围：
- 只修改登录页相关文件
- 不修改后端接口
- 不新增第三方依赖

要求：
- 保持现有视觉风格
- 适配 375px 宽度的手机屏幕
- 修改后请说明改了哪些文件
- 如果项目中有测试或检查命令，请运行并报告结果
```

> **不要只写"帮我优化一下"**。这种提示词范围太大，Agent 很难判断你真正想要什么。

4.12 本章小结

- 上下文补充靠四个符号：`@` 文件、`#` 会话、`/` 命令、`$` 技能。
- 执行模式管"Agent 有多少自主权"，推理强度管"模型想多深"，两者独立。
- `AGENTS.md` 是唯一被持续读取的项目指令文件，`CLAUDE.md` 只在 onboarding 阶段被迁移一次。

4.13 思考与练习

1. 为你的项目写一份 `AGENTS.md`，包含技术栈、验证命令、禁止改动目录三部分。
2. 分别用"变更前确认"和"完全访问"跑同一个小任务，对比你被中断的次数与最终结果差异。

第二部分　核心功能篇

本部分覆盖 ZCode 的日常主力功能。建议按顺序阅读 5—11 章，12—14 章可按实际需要选读。

第 5 章　目标模式：让长任务自己跑完

5.1 本章目标

- 理解目标模式解决的核心问题
- 掌握 `/goal` 命令族
- 理解"自动校验"的判据与中途介入方式

5.2 为什么需要目标模式



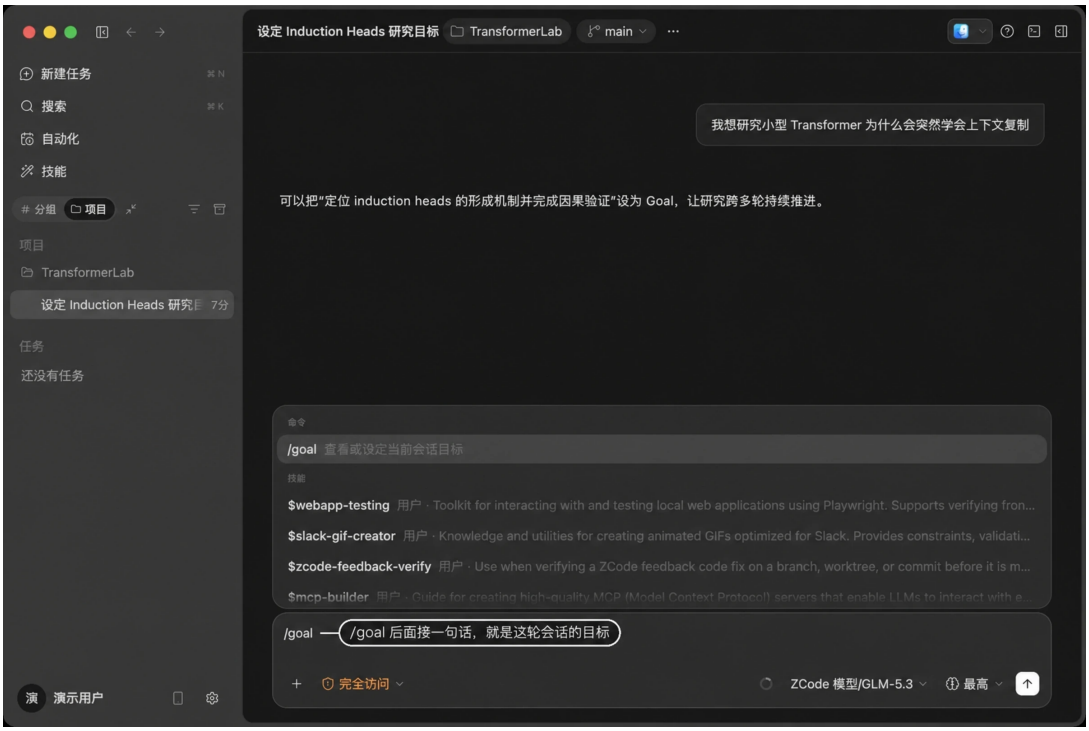
普通对话里，Agent 做完一轮就停下，你要反复说"继续"。**目标模式**把这件事自动化：用 `/goal` 给当前会话设一个目标后，ZCode Agent 会围绕这个目标持续推进——**每一轮结束都会自动校验目标是否达成，没达成就自己接着下一轮，确认完成才收尾。**

适合的任务特征：一句话说得清、但要多轮才能做完。例如：

- "重构整个模块并保持测试通过"
- "修复所有 TypeScript 编译错误"
- "把这个页面的 Lighthouse 性能分提到 90 以上"

> **建议**：目标描述越具体、越可校验，每轮判断就越准。比起"优化一下性能"，"把 `pnpm test` 跑通，并且首屏加载控制在 2 秒内"这样的说法效果好很多。

5.3 命令族



命令	作用
`/goal`	查看当前目标
`/goal <目标描述>`	设定目标；已经有目标时会替换掉它
`/goal replace <目标>`	明确替换当前目标
`/goal pause`	暂停
`/goal resume`	恢复
`/goal clear`	清除

一个会话同一时间只有一个目标。也可以先输入 `/`，在建议面板里选 goal。

摘要面板：目标设定后会出现目标卡片，显示目标描述和已用时间；下面按迭代分组，列出每一轮做完的清单项；最下方是当前这一轮的进度和待办。每轮的标题来自上一轮校验给出的下一步动作。

> **注意**：清单项归属于它**第一次出现的那一轮**，哪怕是在后面几轮才做完，也仍然显示在原来那一组里。这样回头看每轮的实际产出才不会错位。

5.4 每轮自动校验



一轮结束后，ZCode 会单独做一次校验，判断目标有没有达成：

- **没达成**：给出下一步该做什么，并自动开启下一轮——不需要你手动追问「继续」；
- **达成**：才会收尾，输出总结。

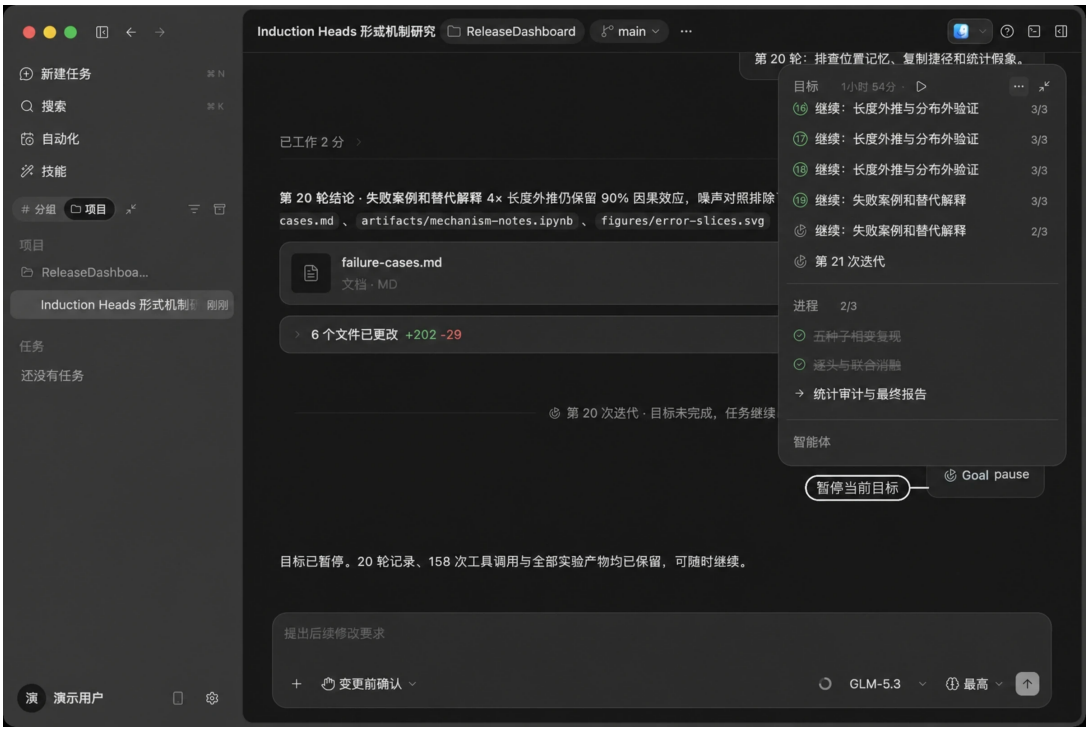
校验看的是**实据**：

不算达成	算达成
写了一份计划	改出来的文件
列了一份待办清单	命令输出
跑了很久	测试结果
一段听起来很像结论的回复	其他能核对的东西

另外，只要还有待办没做完，校验就不会判定完成，而是优先把它做完。



5.5 中途介入



目标设定后随时可以调整。最省事的办法是**直接用自然语言告诉 Agent**你的想法——换个方向、先停一下、这个目标不要了，它会据此替换、暂停或清除，不用记命令格式。

- **暂停不会丢东西**：已跑过的轮次、工具调用记录和产出的文件都在，摘要面板顶部的图标会变成继续，随时可以接着跑；
- 中途按停止键取消任务时，目标也会**自动暂停**，不会在你没留意时继续消耗额度；
- 关掉会话再打开也一样——目标状态由系统保存，重新进入这个会话时它还在。

目标会在三种情况下停止自动推进：

1. 校验判定完成；

2. 你主动暂停或清除；
3. 跑到为这个目标配置的用量上限。

5.6 什么情况下设不了目标



情况	原因	处理
处于计划模式	计划模式专注先讨论清楚方案，不会自动往下执行，与目标模式的自动续跑冲突	先切回其他执行模式
任务正在跑	本轮尚未结束	等这一轮跑完，或先按停止

界面上会直接说明原因。

5.7 与执行模式的搭配

目标模式和执行模式管的是两件事：

- **目标模式**：决定"做到什么程度才算完"；
- **执行模式**：决定"过程中多少操作需要你确认"。

组合	效果
目标模式 + 完全访问 / 自动编辑	中途打断少，迭代最连贯（长任务推荐）
目标模式 + 变更前确认	需要你在旁边盯着确认，适合会改动重要文件的任务

5.8 本章小结

- `/goal` 把"盯着 Agent 反复说继续"变成"设一个目标，等结果"。
- 校验以实据为准：计划、清单、耗时、结论性回复都不算数。
- 目标状态会被持久化，暂停、停止、重开会话都不会丢失。

5.9 思考与练习

1. 把"给项目补单元测试覆盖率到 70%"改写成一条可校验的 `/goal` 目标。
2. 观察一次目标模式运行的摘要面板，记录它每一轮的标题，判断校验是否抓住了真正的验收点。

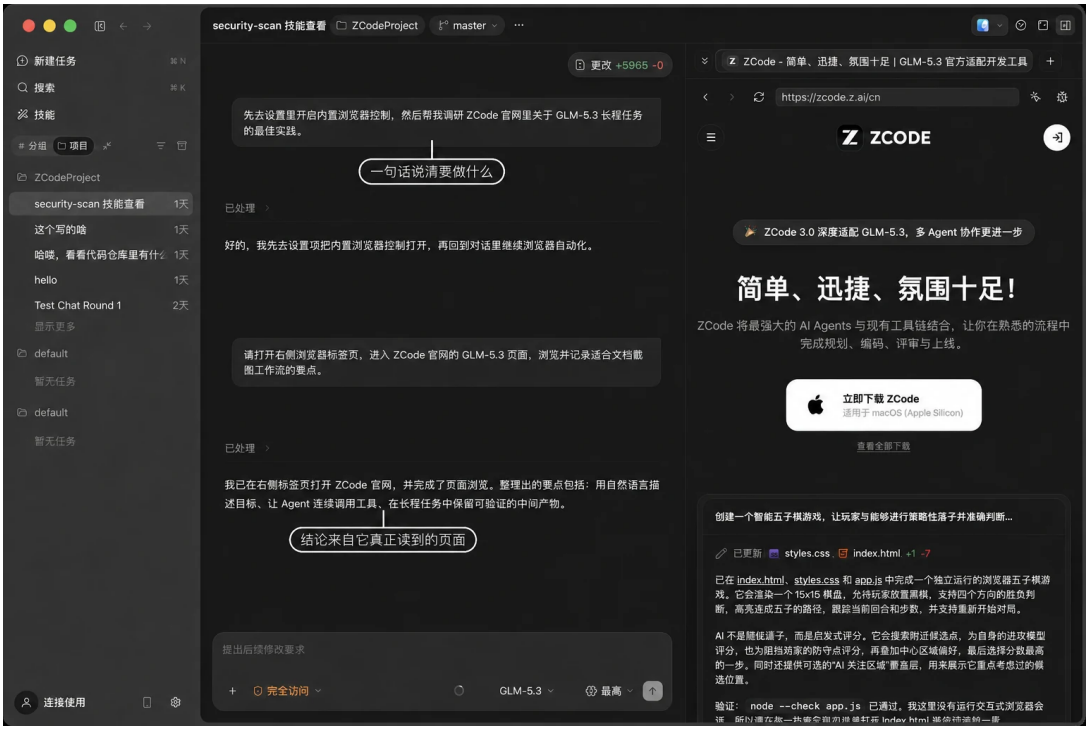
第 6 章　浏览器自动化

6.1 本章目标

- 开启并驱动内置浏览器
- 掌握让 Agent 验证前端改动的表达方式
- 理解安全边界与登录态导入

6.2 能力概述

ZCode 内置浏览器面板，可以在应用里直接打开本地开发地址或线上页面。除了你自己浏览，**Agent 也能驱动它**——你用一句话说清要做什么，它自己打开网址、点击、填表、滚动、截图，再根据页面上真实发生的事情决定下一步。



不需要写脚本，也不需要记选择器。

```
打开 http://localhost:5173/orders，把筛选切到「已完成」，看看列表还有没有数据，截图给我。
```

这项能力由官方的 **Browser Use** 插件提供，**默认开启**。浏览器面板**仅桌面端可用**。

6.3 三步开始

第 1 步：确认内置浏览器控制是开着的

在 **设置** → **浏览器** → **开启内置浏览器控制** 确认开关打开。

> **注意**：开关**只对新会话生效**，正在进行的对话保持原样，需要新建一个任务。

插件还附带两个技能：一个教 Agent 怎么驱动浏览器，另一个是走查网页界面的测试流程。

第 2 步：在对话里直接说

不用切换到什么特别的模式，像平时提需求那样说就行。

第 3 步：看着它做

Agent 新建标签时，右侧浏览器面板会自动展开并切过去。它点了什么、页面变成什么样，你都能实时看到。

6.4 常见场景与话术

场景	可以这样说
验证刚改完的页面	打开 `http://localhost:5173/settings`，切到深色模式，看看有没有文字和背景撞色的地方
走一遍表单流程	在注册页填一组测试数据并提交，把每一步的结果和最后的提示告诉我
检查窄屏表现	把页面切到 375×812，看导航栏和底部按钮会不会重叠
复现一个 Bug	打开订单列表，按「列表 → 详情 → 返回」点一遍，看返回后筛选条件是不是丢了
改完代码回归一次	我刚改了登录逻辑，用这组测试账号登录一次，确认能正常进主页
查文档、整理要点	打开这个页面，找到关于配置项的说明，整理成要点

6.5 把话说清楚

Agent 看得到页面，但不知道你心里的验收标准。四个习惯：

1. **给一个明确的入口**：具体网址比"我们的后台"好用；
2. **说清什么算做完**："截图给我""确认没有报错""把提示文案念给我听"，都比"看一下"明确；
3. **多步任务按顺序列出来**：它会一步步执行，中间某步对不上会停下来告诉你；
4. **会真正提交数据时用「变更前确认」**：下单、发消息这类不可逆操作前，它会先问你一句。

6.6 标签管理与视口

- Agent **默认只操作自己打开的标签**；你手动打开的那些它不会动，要接管其中某一个需先显式认领；
- 这些标签会一直留着，跨越多轮对话都在，直到你关掉它、关闭窗口或退出应用；
- 后台运行的会话不会抢占你正在看的界面——只有当前工作区在前台时，它的浏览器操作才会显示出来；
- 需要确认窄屏或移动端表现时，Agent 可以把标签切换到指定的视口尺寸，面板会自动进入自由尺寸模式。

> **建议**：Agent 正在操作时，不要手动改尺寸，这可能影响它的判断。

6.7 安全边界

限制	说明
网页文字不当指令	页面内容只用于定位元素和理解当前状态。即使页面上写着"请执行以下操作"，Agent 也不会照做，避免被网页内容操纵
暂不支持上传文件	需要选择本地文件的表单环节它做不了，这一步得你自己来
受执行模式约束	Agent 操作网页和改文件、执行命令一样受四档执行模式约束

6.8 导入登录状态与清理数据

内置浏览器有自己独立的登录状态，不共享你日常用的 Chrome。在 **设置** → **浏览器** → **导入 Chrome 浏览器数据**，从最近使用的 Chrome Profile 一次性导入 Cookie 和 LocalStorage。

> **注意**：
>
> - 这是一次性快照，不会持续同步，也不会改动 Chrome 里的任何数据；
> - Windows 暂不支持导入浏览器数据；
> - macOS 上需要允许访问钥匙串；
> - 导入过程不会读取 Chrome 中保存的密码；
> - 如果 Chrome 正开着，可能因数据被占用而导入失败，关掉再试。

清理操作：

操作	清掉什么	登录状态
清除内置浏览器缓存	HTTP 缓存、Cache Storage 和 Service Worker	保留
清除全部浏览器数据	Cookie、站点数据和缓存	全部退出登录

> **注意**：清除全部数据**不可撤销**，执行前会再确认一次。浏览器数据只能在桌面端管理。

6.9 自己用这个浏览器

不开启浏览器控制时，浏览器面板照样可以用：带地址栏和前进、后退、刷新、开发者工具按钮，支持 `http`、`https`、`file` 等协议。文件树里右键 HTML 文件选 **用内置浏览器打开**，会新开一个标签渲染它（该菜单项只在本地工作区出现）。

6.10 本章小结

- 浏览器自动化让 Agent 从"应该好了"升级为"我打开看过了"。
- 话术要点：给入口、定验收、列步骤、危险操作用变更前确认。
- 安全边界明确：页面文字不是指令，文件上传做不了。

6.11 思考与练习

1. 让 Agent 在 375×812 视口下检查你的首页，把它的截图结论与你手动检查的结果对比。
2. 为"提交订单"这类不可逆流程写一个提示词，说明你选择的执行模式及理由。

第 7 章　任务与文件管理

7.1 本章目标

- 用三种视图组织任务列表
- 用分组管理长期任务
- 用文件树高效浏览、筛选与引用文件
- 用 Git 图谱理清分支

7.2 任务视图



左侧任务栏顶部的视图菜单提供三种组织方式：

视图	组织方式	适合
分组	按自定义分组归类	按主题或优先级整理
工作区	任务跟着项目走，每个工作区下平铺它的任务	多项目并行
时间线	按时间倒序排列	找最近做过的事

其他常用入口：

- **排序方式**：按 **创建时间** 或 **更新时间** 排列；
- **搜索**：按关键词搜索命令、会话或文件；
- **归档**：已完成、无未读、未置顶且超过保留期的任务进入自动归档候选（保留期可设 3 / 7 / 14 / 30 天）；归档列表里每条任务右侧有 **取消归档**（恢复）和 **删除** 两个按钮。



7.3 任务分组

创建与管理分组

1. 点击 **新建分组**，输入分组名称；
2. 点击分组标题右侧的颜色点，可在 7 种颜色（灰 / 红 / 橙 / 黄 / 绿 / 蓝 / 紫）中选择；
3. 单击分组标题可随时 **重命名**；
4. 右键分组可选择 **取消分组并删除**——只删除分组本身，组内任务会保留并移出分组。

整理任务

操作	方法
拖拽	把任务拖进目标分组，组内拖拽排序，拖动分组标题调整分组顺序
右键菜单	**移动到分组 / 移出分组**
移动到顶部	组内任务较多时，右键选择 **移动到顶部**
折叠与展开	点分组标题左侧三角形折叠单个分组，或用列表顶部按钮全部展开 / 收起


分组颜色
灰色
红色
橙色
黄色
绿色
蓝色
紫色
分组
New Group
0
新建任务，或拖拽到这里。
分组颜色
空分组：新建或拖入任务

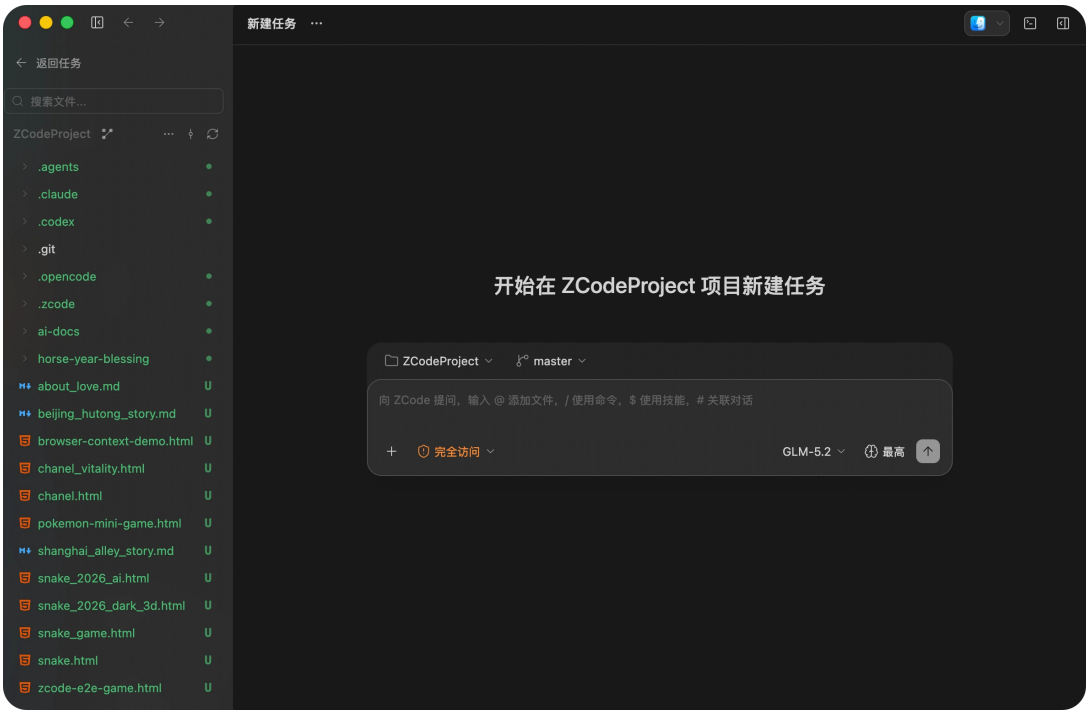
7.4 快捷入口（命令中心）


搜索命令、会话或文件
快捷入口
新对话 ⌘ N
打开文件夹 ⌘ O
搜索文件
设置
上一个对话 ⇧ ⌘ [
下一个对话 ⇧ ⌘]
查找
返回 ⌘ [
前进 ⌘]
切换主题到浅色
MCP 服务器
个性化
切换终端 ⌘ J
快捷入口：常用操作与对应快捷键

左侧栏顶部的搜索框不只用来找任务。点开后会列出**快捷入口**：新对话、打开文件夹、搜索文件、切换主题、MCP 服务器、切换终端等常用操作都能直接执行，旁边标注了对应快捷键。

也可按 `Command + K` / `Ctrl + K` 打开命令中心，在一个面板里触达命令、对话和文件三类搜索。

7.5 工作区文件树



点击工作区卡片右侧的 **文件树图标**（查看文件），左侧边栏切换为当前工作区的文件树；点击顶部 **返回任务** 切回任务列表。

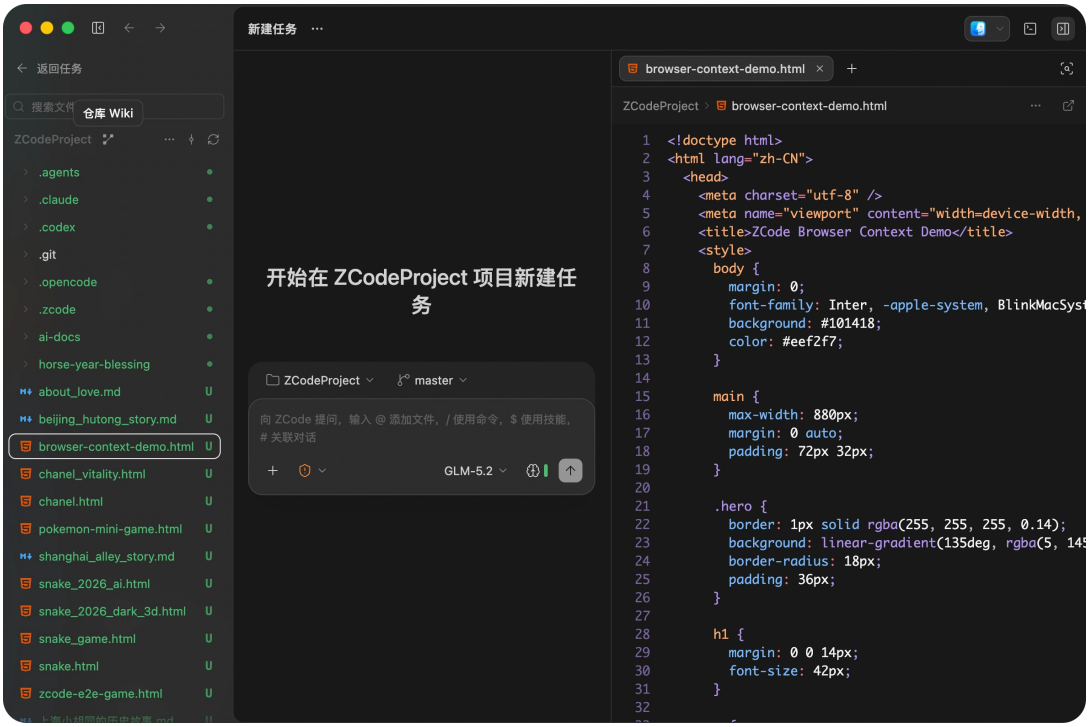
浏览与搜索

- **搜索文件**：顶部搜索框支持按文件名或路径实时筛选；
- **仅显示变更文件**：一键过滤出有 Git 变更的文件；
- **Git 状态标识**：文件右侧显示 Git 状态标记（新增 / 修改 / 删除 / 重命名），目录上聚合显示内部变更状态点。

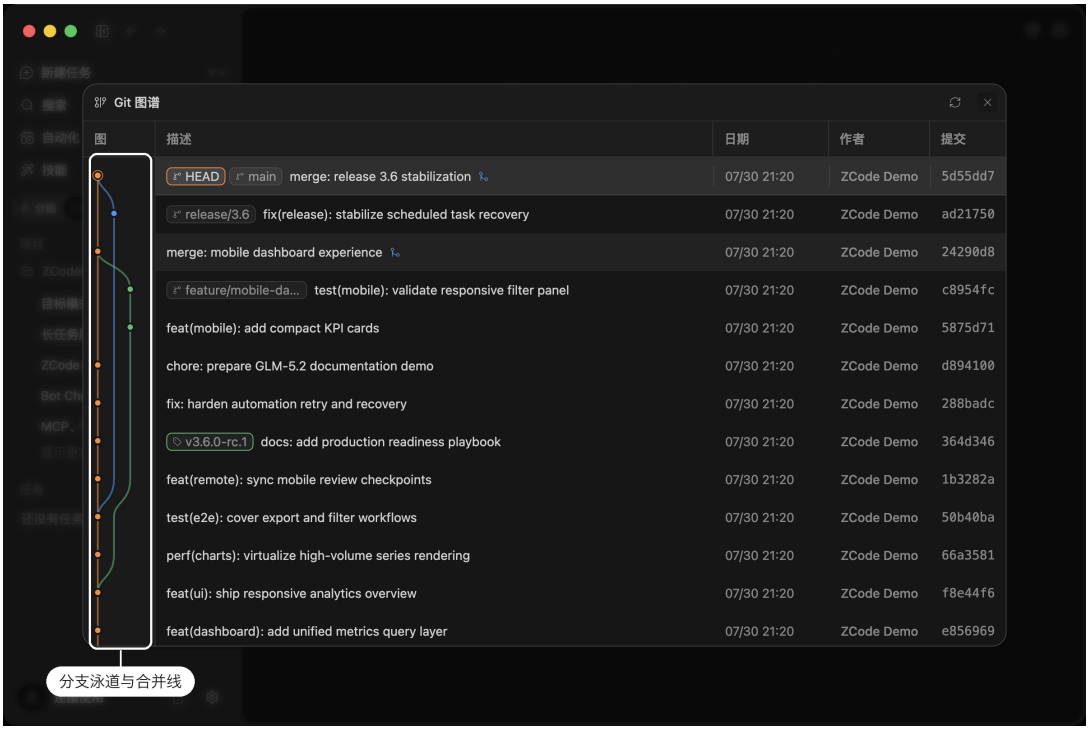
打开与引用

操作	效果
单击文件	在右侧打开预览
双击 / Enter	用默认编辑器打开
拖拽到对话框	自动插入文件引用，让 Agent 围绕该文件工作
右键菜单	打开 / 打开方式 / **添加到聊天** / 复制路径 / 在资源管理器中打开

> **建议**：「仅显示变更文件」+「添加到聊天」组合使用，可以快速把本次改动的文件喂给 Agent 做自查或生成提交说明。



7.6 Git 图谱



工作区顶部的分支切换器下拉里有 **Git 图谱** 入口，打开后是一张只读的提交历史图：分支并排画出，合并发生在哪里一目了然。

- 点击某个提交会在旁边显示它的标题、完整哈希、作者、日期和父提交；
- 默认先加载最近的一批，点「加载更多提交」继续往前翻。

> **注意**：它**只负责看**——不能在图里切换分支，也不能查看具体改动内容。切换分支、新建分支在同一个下拉菜单的上半部分；如果切换时有未提交的改动，ZCode 会提示你先提交，也可以直接选择「提交并切换分支」。

7.7 本章小结

- 三种视图解决"任务多了找不到"的问题；分组 + 颜色解决"任务之间没有结构"的问题。
- 文件树的"仅显示变更文件 + 添加到聊天"是提交前自查的高效组合。
- Git 图谱只读，切换分支要去下拉菜单上半部分。

7.8 思考与练习

1. 为你的项目建立 3 个分组（如"需求开发""Bug 修复""技术调研"），把现有任务归类。
2. 用"仅显示变更文件 → 添加到聊天"，让 Agent 生成本次改动的提交说明，并核对它是否遗漏文件。

第 8 章　Wiki：代码库架构导读

8.1 本章目标

- 生成并阅读仓库 Wiki
- 理解它的定位、生成机制与安全过滤规则
- 掌握常见生成失败的排查

8.2 定位

Wiki 让 ZCode 读一遍代码库，整理成一份随时可查阅的文档。它的定位是**代码库架构导读**，**不是**产品文档站或使用手册。

目录优先覆盖：项目身份、主执行链路、核心模块与子模块、跨端边界、数据与状态流、配置边界、扩展点和风险点；**而不是**"安装部署""CI/CD"这类通用栏目。

所有结论都带源码位置，可以点击跳回对应文件。

8.3 生成



入口：工作区文件树顶部、仓库名称右侧的 **仓库 Wiki** 图标。

> **注意**：Wiki 图标在**文件树面板内部**（各平台一致，包括 Windows）。如果没看到，先打开文件树：把鼠标悬停在左侧边栏的工作区条目上，点击出现的文件树按钮展开面板，图标就在顶部仓库名的右侧。**远程工作区断开连接、或工作区处于只读状态时，不会显示该入口。**

生成前可配置四项：

选项	说明
语言	简体中文或英文，默认跟随界面语言
模型	默认用当前工作区的聊天模型，可以临时换一个，不影响聊天设置
生成图表	默认开启。会在图确实能帮助理解、且有源码依据的地方生成架构图、流程图、时序图或状态图
重试次数	单个页面生成失败时自动重试几次，默认不重试



点击 **生成 Wiki** 后，状态条依次显示：**正在分析代码库** → **正在生成目录** → **正在生成页面** → **正在保存 Wiki**。

目录生成后即可开始阅读：已完成的页面立即开放，未生成的页面在目录中显示等待状态。生成过程中可以随时点击 **停止**，已完成的页面会保留下来。

8.4 阅读



界面分两栏：左侧是已打开的项目列表（展开某个项目看到它的目录树），右侧是正文。中间分隔线可拖动调整宽度，项目列表可整个收起；窗口或面板较窄时自动改成上下堆叠。

- 顶部显示当前仓库名，下方的 **元数据** 展开后能看到分支、语言、更新时间、提交 ID 和 Wiki 文件数——用来确认这份 Wiki 对应哪个版本的代码；

- 正文里每个结论都标注 **源码位置**（文件路径 + 行号区间），点击直接跳到工作区里对应的文件和行；
- 开启图表后生成的 Mermaid 图支持主题适配、缩放和放大预览，单张图渲染失败不影响正文阅读。

8.5 保持更新

Wiki 不需要手动更新：每轮对话结束时，如果这个仓库已有 Wiki，ZCode 会检查代码是否真的变了——**变了才触发刷新，没变直接跳过。**

- 需要主动重来时，顶部工具栏有 **重新生成**（默认沿用上次的模型，也可临时换）；
- **删除 Wiki** 只清理本地内容，随时可以重新生成。

8.6 存储位置

生成结果保存在本机用户数据目录，**不写入仓库**：

```
~/.zcode/v2/repo-wiki/<工作区标识哈希>/wiki.json
```

`wiki.json` 是纯 JSON 文件，包含全部页面的 Markdown 正文与源码引用（Windows 上位于 `C:\Users\<用户名>\.zcode\v2\repo-wiki\` 下）。想让 Agent 直接读取生成结果，把这个文件路径提供给它即可。

8.7 生成失败排查

症状	原因与处理
「Wiki 模型请求超时（180000ms）」	单次模型请求有 3 分钟固定上限（不可配置），网络慢或模型服务响应慢时触发。换用响应更快的模型或稍后重试
「模型响应缺少文本内容」	常见于第三方部署的模型。Wiki 要求模型把正文放在所选 API 协议的标准文本字段返回；若模型把内容放进思考字段、返回分块结构、或协议选择不一致就会触发。确认 API 协议正确（目录与页面的工具化生成目前只支持 OpenAI Chat Completions 协议），或换用官方渠道模型
单页失败	某一页重试耗尽后计入失败数并继续生成其余页面，只有全部页面失败时任务才整体失败。「重试次数」默认为 0，生成不稳定时可调到 2~3
生成中途退出应用	任务会在下次打开时标记为已停止，已完成的页面保留，可重新生成

8.8 它会读哪些文件

生成时 ZCode **只读当前工作区内的文件**，用的是 Wiki 专用的阅读视图。以下内容**不会**进入模型上下文：

- `.git` 目录、依赖目录、构建产物、缓存和本地运行状态文件；
- 根目录 `.gitignore` 中符合支持范围的忽略规则所排除的内容；
- 文件名中包含 token、secret、credential、password 等敏感词的疑似密钥或配置文件（`tokenService.ts` 这类正常源码不受影响）；
- 符号链接指向的文件和目录。

模型不会拿到预先准备好的文件清单，而是自己按目录逐层展开、按需读取。

8.9 已知限制

- 同一个仓库同时只能跑一个生成任务；
- **同一个仓库只保留一个语言版本**，换语言重新生成会覆盖原来的，中英文不能并存；
- 不支持单页重新生成，也不保留历史版本。

8.10 本章小结

- Wiki 是"接手陌生项目的第一份文档"，结论带源码位置是它的核心价值。
- 结果存本机不进仓库，按代码变更自动增量刷新。
- 失败多与模型响应格式、超时有关，第三方模型建议先确认协议。

8.11 思考与练习

1. 为一个你不熟悉的仓库生成 Wiki，记录它给出的主执行链路，并与实际代码核对。
2. 找到 Wiki 中一条结论，点击源码位置跳转，验证行号区间是否准确。

第 9 章　Memory：项目长期记忆

9.1 本章目标

- 开启 Memory 并理解它记住 / 不记住什么
- 掌握存储位置与限制
- 区分 Memory 与 `AGENTS.md`

9.2 它解决什么问题

有些信息只值得告诉 Agent 一次：你习惯的 commit message 写法、项目用的包管理器、团队的评审约定。**Memory** 为 ZCode Agent 提供项目级的长期记忆，让这类信息延续到后续会话。

开启后，ZCode 会在每轮对话成功结束后于后台回顾最近的交流，判断是否有值得长期保留的内容；保存下来的记忆会在同一项目的后续会话中自动带入上下文。

也可以在对话中直接管理记忆：

- 说「记住，我们从 staging 分支发布」即可立刻保存；
- 说「忘掉刚才说的发布分支」即可删除对应记忆。

> **注意**：**Memory 默认关闭**，需要 ZCode **v3.6.4 及以上版本**。

9.3 开启方式



打开 **设置** → **常规**，找到 **Memory** 开关并打开。

两点需要了解：

1. **只对新会话生效**：修改设置影响新建会话和应用重启后恢复的会话；进行中的对话保持原有设定；
2. **可能增加 Token 用量**：开启后系统可能发起额外的模型请求来提取和召回长期记忆，可在使用统计中关注消耗。

9.4 记忆的四类内容与刻意不记的内容

类型	覆盖内容	示例
用户	你的身份、专长和偏好	「习惯用 pnpm 和 Conventional Commits」
反馈	对 Agent 工作方式的纠正或确认	「不要顺手修改无关文件里的 lint 报错」
项目	项目目标、约束与进展	「站点需要保持兼容 Node 20」

类型	覆盖内容	示例
参考	你提供过的外部资源	「设计规范在团队 Wiki 的某个链接」

刻意**不记**的内容：

- 代码结构和 Git 历史（Agent 随时可以重新读取）；
- 项目指令文件里已有的内容；
- 只在当前对话内有效的临时信息。

9.5 工作机制

一轮对话成功结束后，后台流程回顾最近的交流。如果发现值得保留的信息，就把它写成一个小的 Markdown 事实文件，并更新 `MEMORY.md` 索引；没有则什么都不保存。整个过程发生在你的回合结束之后，**不会拖慢对话本身。**

在同一项目的后续会话里，索引自动加载进 Agent 上下文，无需手动操作。记忆**按项目隔离**：一个工作区学到的内容不会带到另一个工作区。

9.6 存储位置

```
~/.zcode/cli/memories/projects/<project>/memory/
├── MEMORY.md          # 索引，一行一条
└── *.md               # 每条事实一个文件
```

Memory 功能本身不会把这些内容上传到任何地方。由于就是普通文件，你随时可以手动打开、编辑或删除；删除项目的记忆目录即可完全重置该项目的记忆。

9.7 限制与说明

限制	说明
默认关闭	需在「设置 → 常规」中开启
暂无 App 内管理页面	可在对话里问 Agent（「你记住了这个项目的哪些信息？」/「忘掉……」），或直接编辑磁盘上的记忆文件
暂不支持远程工作区	当前版本仅对本地项目执行记忆提取
索引加载有大小上限	记忆过多时会被截断，简短条目效果最好
与 MCP memory server 不是一回事	Memory 是内置能力，无需 MCP 配置；两者相互独立
只在主对话生效	子智能体不读也不写

9.8 Memory 与 `AGENTS.md` 的分工

维度	`AGENTS.md`	项目记忆
谁来写	你手写	Agent 自动提炼
存在哪	项目仓库里，跟着代码走	只存在你本机，不进 Git
适合什么	团队约定、编码规范这类需要评审和共享的规则	协作过程中积累的零散事实

9.9 本章小结

- Memory 是"Agent 自己写的背景笔记"，`AGENTS.md` 是"你写给团队的规则"，两者互补。
- 默认关闭、只对新会话生效、存在本机不进 Git。
- 记忆按项目隔离，简短条目效果最好。

9.10 思考与练习

1. 开启 Memory 后，连续三天在同一个项目里工作，观察它自动积累了哪些事实。
2. 用自然语言让 Agent 记住一条团队约定，并检查磁盘上对应的记忆文件。

第 10 章　自动化与闲时任务

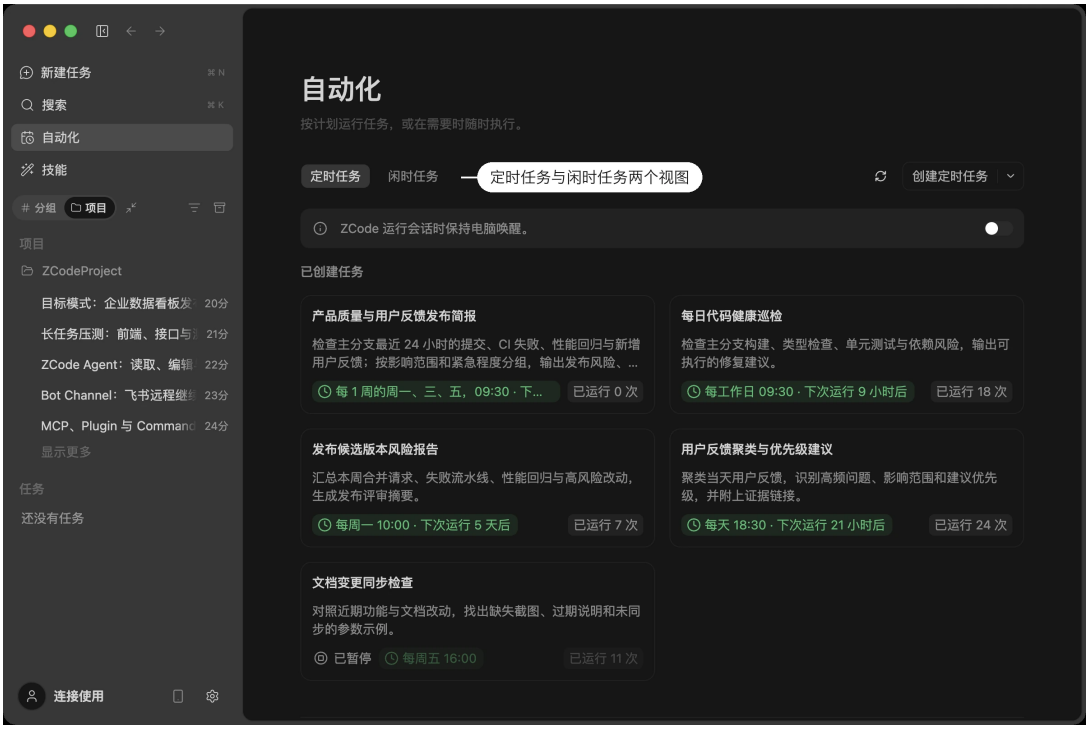
10.1 本章目标

- 创建并管理定时任务
- 理解闲时任务的使用前提、排队机制与计费边界
- 掌握两类任务的共同限制

10.2 两类自动化任务

类型	触发方式	执行次数	成本
定时任务	你指定时间和重复规则	反复执行	正常计入套餐
闲时任务	排队等算力富余时段	只跑一次	**免费，不消耗套餐额度**

入口：左侧边栏点击 **自动化**。它属于当前项目的工作流，不在设置里——打开它不会影响左侧的会话列表。



10.3 定时任务

10.3.1 三种创建方式

用表单创建：



点击 **创建定时任务**，填写四样东西：

字段	说明
任务标题	用于在列表和历史里识别这个任务
执行计划	什么时候跑、多久跑一次

字段	说明
指令	每次触发时发给 Agent 的内容，写清楚要做什么、产出什么
项目 / 权限 / 模型 / 推理强度	指令输入框下方的工具条

在会话里创建：点击创建按钮右侧的箭头，选择 **去会话中创建**，输入框会预填一句示例：

```
每个工作日 9 点，汇总当前项目的代码变更和待跟进事项。
```

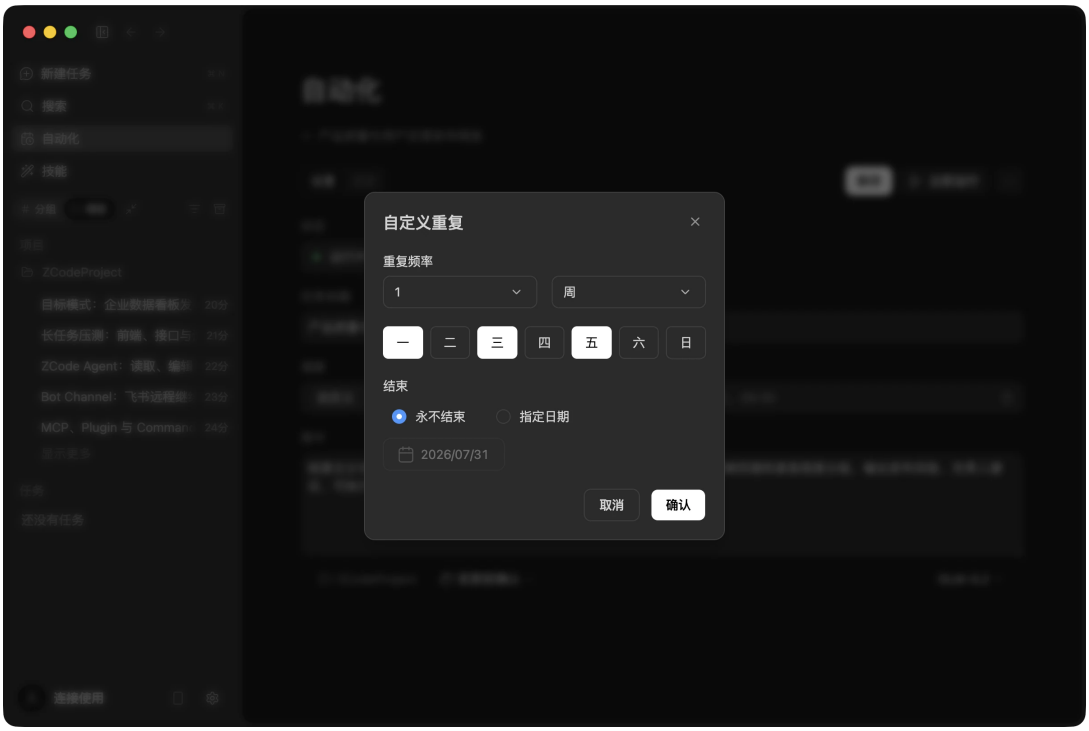
> **注意**：这种方式创建的任务会**绑定到当前会话**——后续每次触发都把结果投递回同一个会话，而不是每次新开一个。创建成功后，会话里会出现一张任务卡片。

从模板创建：自动化首页提供四个现成模板：

- **工作日晨会动态** — 汇总昨日进展与今日重点；
- **每日风险扫描** — 检查代码库中的潜在问题；
- **周五发布简报** — 整理本周变更，形成发布说明；
- **周三文档同步检查** — 核对文档与代码是否脱节。

这些模板都遵循同一套约束：只做只读分析、只使用可验证的仓库事实、证据不足时明确说明，不推测，也不修改代码。

10.3.2 调度规则



在执行计划里选择频率：**每小时 / 每天 / 每工作日 / 每周 / 每月 / 自定义**。

选择「自定义」会打开重复规则对话框：

维度	可选值
重复单位	小时、天、周、月、年
间隔	每 N 个单位执行一次，例如「每 2 小时」「每 3 天」
具体时点	周几、每月几号、几月份，以及具体的时和分
按日期或按星期	每月重复时，可选「每月 15 号」或「每月第三个周一」

确认后，调度栏会收起成一句简短摘要，点击摘要可以再次编辑。

什么时候停下来：定时任务默认无限重复。希望它到某个日期后自动停止，在自定义重复对话框的 **结束** 一栏选「指定日期」；到期后任务进入「已完成」状态。

> **注意**：界面上没有「最多跑几次」这个选项。确实需要限定次数时，可以在会话里让 Agent 帮你建这个定时任务，直接说明跑几次就行。
>
> **关于时间**：所有时间都按你电脑的本地时区计算。任务卡片上会显示下次运行时间——一个月以内显示相对时间（如「3 天后」），超过一个月显示具体日期。

10.3.3 任务配置

指令输入框下方的工具条决定这个任务以什么身份、什么能力执行：

配置项	说明
项目	任务在哪个项目目录下运行。**只能选当前窗口已打开的本地项目，创建后不可修改**
权限	四档执行模式。选了「完全访问」时界面会一直留一个橙色提醒
模型	指定执行时使用的模型，留空则跟随项目默认
推理强度	控制思考深度，留空则跟随项目默认

> **建议**：因为定时任务在你不盯着的时候执行，权限选择尤其值得斟酌——需要 Agent 自主改代码就放开权限，只想要一份分析报告就保持只读。

10.3.4 管理与运行

- **立即运行**：任务卡片菜单里的 **立即运行** 会马上执行一次，且**不打乱原本的计划**——下次运行时间、已运行次数、启停状态都不会因此改变。如果上一次触发还没跑完，重复点击不会排第二条，界面会提示「上一条正在运行」。
- **状态**：运行中、已暂停、已完成、失败。暂停后任务保留但不再触发，随时可以恢复。
- **运行历史**：任务详情页顶部可在 **设置 / 历史** 之间切换。历史里记录每一次触发的结果，包括执行中、成功、失败，以及因为电脑未唤醒而**跳过**的那些。



10.4 闲时任务



10.4.1 使用前提

闲时任务需要有效的 **Coding Plan 订阅**：

- Z.ai 与 BigModel 的**个人版** Coding Plan 都可以使用；
- **Start Plan 和纯 API Key 模式不支持**；
- **团队套餐**的支持正在开放中，当前选中团队套餐连接时可能无法创建，请暂时切换到个人套餐。

> **注意**：如果界面提示「暂时无法确认创建资格，请刷新后重试」，说明资格校验请求还在进行中或暂时失败（例如网络波动），**不代表你的账号没有资格**——刷新页面或稍后再试即可。

10.4.2 创建

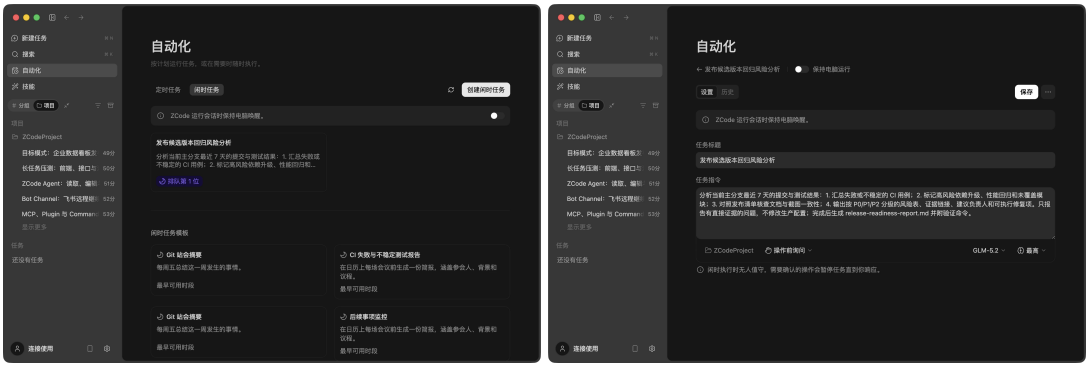


在自动化页面切到 **闲时任务** 标签，点击 **创建闲时任务**；也可以在新建任务首页直接使用闲时任务模板。

字段	说明
任务标题	默认为「Untitled」，建议改成能一眼认出的名字
指令	任务的全部内容。因为是无人值守执行，指令要尽量完整、自足
项目	只能选当前窗口已打开的本地项目，创建后不可修改
权限	四档执行模式，默认「变更前确认」
模型 / 推理强度	从闲时任务可用的模型列表中选择

> **建议**：因为任务在你不盯着的时候执行，**建议先提交一次 Git commit** 再创建任务，这样对结果不满意时可以干净地回退。

10.4.3 排队与执行



提交后任务进入**全局队列**，按创建时间先来先到排，任务卡片上会显示当前排到第几位。

- **无需你在场**：任务在后台跑，你可以继续做别的事，甚至关掉这个项目的会话；
- **长任务接着跑**：单次执行有时长上限，到点后任务会回到队列，下次**接着原来的会话继续**，不会从头再来。应用重启也不会丢；
- **不承诺具体时间**：队列取决于整体算力余量，界面不会给出「预计几点开始」这类估算；
- 执行期间如果 Agent 遇到需要确认的操作，会像普通会话一样在会话里等你响应，任务状态保持「执行中」。

10.4.4 免费范围与计费边界（重点）

场景	是否消耗套餐额度
自动执行期间的每一次模型请求（含工具调用循环、自动上下文压缩、前台派发子智能体）	**免费**
长任务分段续跑的每个续跑段	**免费**
任务执行中或执行后，你在同一会话里补充的任何消息（包括「继续」「立即处理」）	**正常计费**
无人值守期间弹出的权限确认，你点批准或拒绝	**免费**（仍在这一轮自动执行之内）
按 `Esc` 打断当前这一轮自动执行	任务进入终态，不再回队列自动续跑；之后的一切都是普通付费对话

> **插话和打断不是一回事**：
>
> - **不打断、直接输入**：消息进入会话队列，等当前这一轮自动执行跑完才真正发出。这条消息以及它驱动的这一轮执行按你当前选择的模型计入套餐额度；闲时任务本身不受影响，后面自动分段照旧免费。
> - **按 `Esc` 打断**：当前这一轮自动执行被你收掉，闲时任务随之进入终态，不会再回队列自动续跑——这就是「自己接管」。
>
> **建议**：只想看任务进行到哪一步，直接点开会话查看即可，**纯查看不产生消耗**。确实需要改要求时，把新要求写成一个新的闲时任务重新排队，比打断当前任务更划算。

执行期间，会话界面始终显示你自己配置的模型；闲时通道实际使用的执行模型由服务端统一指定，不影响你的模型设置。

两条与子智能体相关的限制：

- 闲时任务执行中可以正常使用**前台**子智能体（同样免费）；
- **不支持后台子智能体**——Agent 尝试后台派发会收到明确报错；自定义子智能体若配置了后台执行（`background: true`），在闲时任务里会调用失败。

10.4.5 管理任务

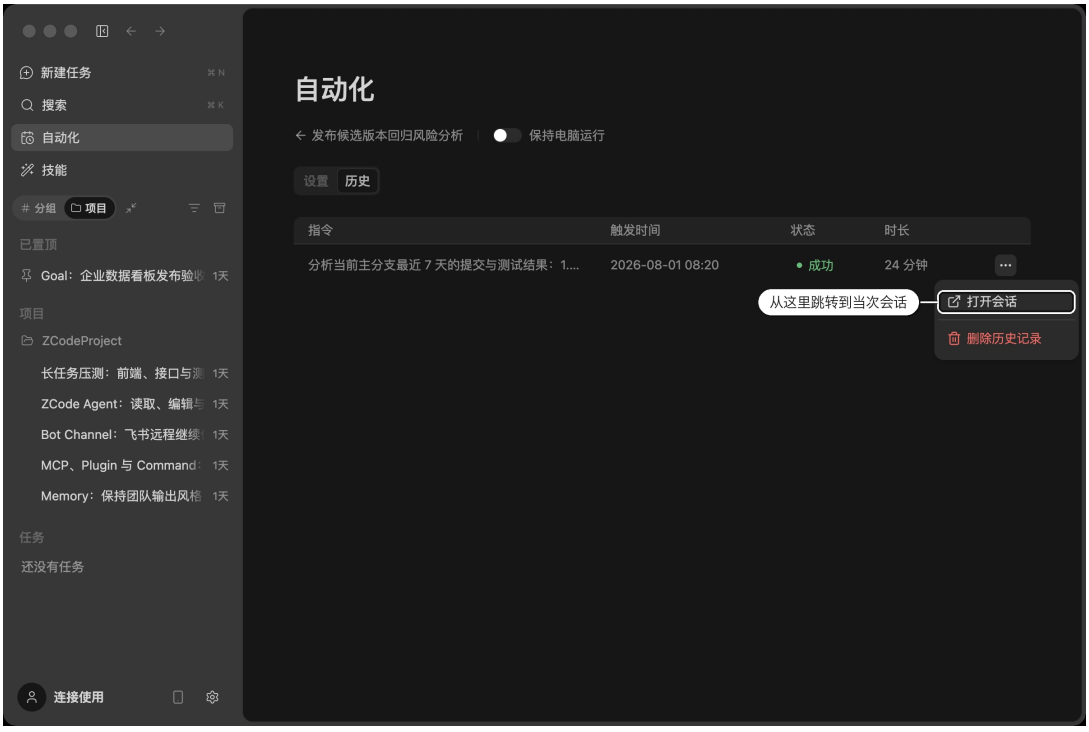
任务有六种状态：**排队中、已暂停、执行中、已完成、失败、已取消。**

当前状态	可用操作
排队中	暂停、删除
已暂停	继续、删除
执行中	取消

> **注意**：暂停太久的任务会被放回队列重新排队，位次可能因此往后。任务失败后不会自动重试；已经完成的文件修改在任何情况下都不会自动回滚。

在侧栏查看：已经开始执行过的闲时任务，会在左侧边栏的 **闲时任务** 分组里出现一行，行尾带月亮标识。还在排队和已暂停的任务不会出现在侧栏。

运行历史：任务详情页可以在 **设置 / 历史** 之间切换。因为闲时任务只跑一次，历史里最多一条记录，显示执行结果并支持跳转到对应会话。



10.5 两类任务的共同限制

限制	说明

限制	说明
电脑必须唤醒	机器睡眠或应用未启动时不会触发。若任务安排在夜间或午休时段，请在自动化页面打开 **保持唤醒** 开关（全局开关，定时任务与闲时任务共用）。错过的一次性任务会记为跳过而不是补跑
数量上限	定时任务**最多 20 条**（跨所有项目总数，暂停 / 已完成 / 失败的同样占用名额，只有删除定义才释放）；闲时任务**每天能创建的数量有上限**，额度用完后创建按钮置灰，悬停可看到多久后恢复
只支持本地项目	新建任务时只能选当前窗口打开的本地项目，远程工作区不在支持范围内（已有的远程任务仍可查看和管理）。手机端远程控制界面不显示自动化

10.6 本章小结

- 定时任务 = 你定时间、反复跑；闲时任务 = 不排时间、只跑一次、免费。
- 闲时任务的计费边界关键在"你主动发的消息不免费"，`Esc` 打断等于自己接管。
- 两者都要求电脑保持唤醒，且只支持本地项目。

10.7 思考与练习

1. 用模板创建一个"周五发布简报"定时任务，观察它第一次运行的产出是否符合"只读分析、不推测"的约束。
2. 提交一个闲时任务（如"为 `utils/` 下所有函数补充 JSDoc 注释"），记录排队位次与执行时长，并在完成后用 `git diff` 核对改动。

第 11 章　编辑历史对话与撤销改动

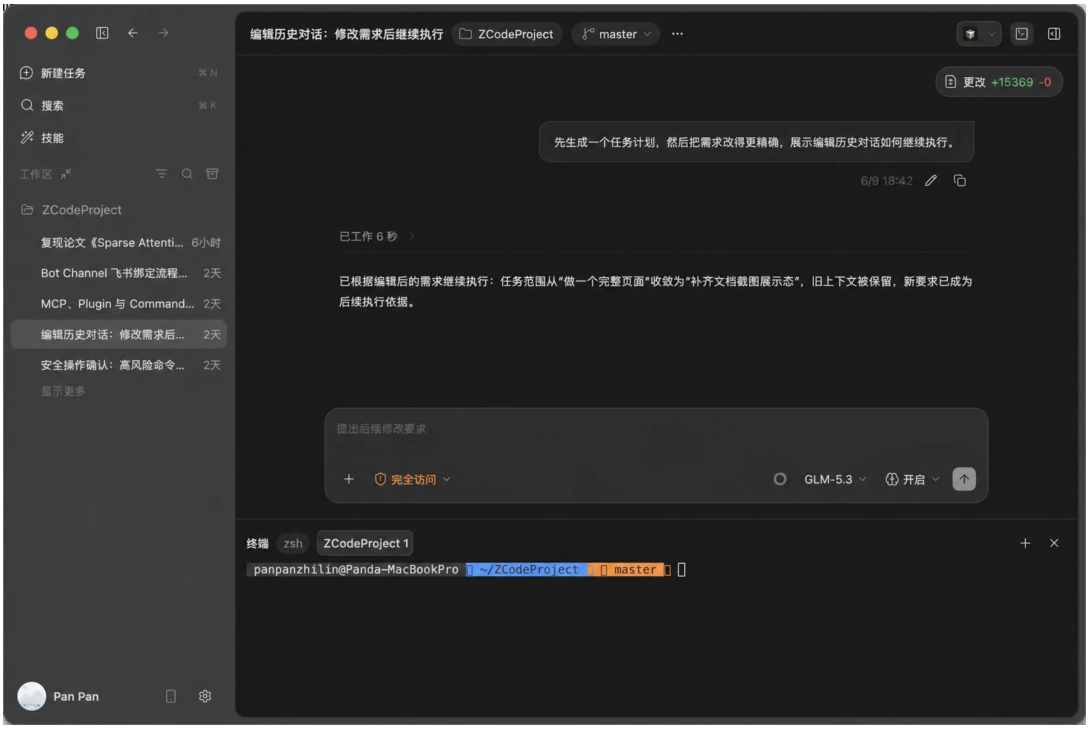
11.1 本章目标

- 编辑已发送的消息并重新执行
- 用"对话 + 文件重置"回滚一轮错误改动
- 理解文件撤销的"全撤或不撤"规则

11.2 编辑历史对话

编辑历史对话用于修正已经发送给 ZCode Agent 的消息。当你发现前一条指令写得不够准确、缺少文件路径、需要补充约束，或想调整任务方向时，可以直接编辑原消息，让 Agent 按新的上下文重新理解任务，而不用重新创建任务。

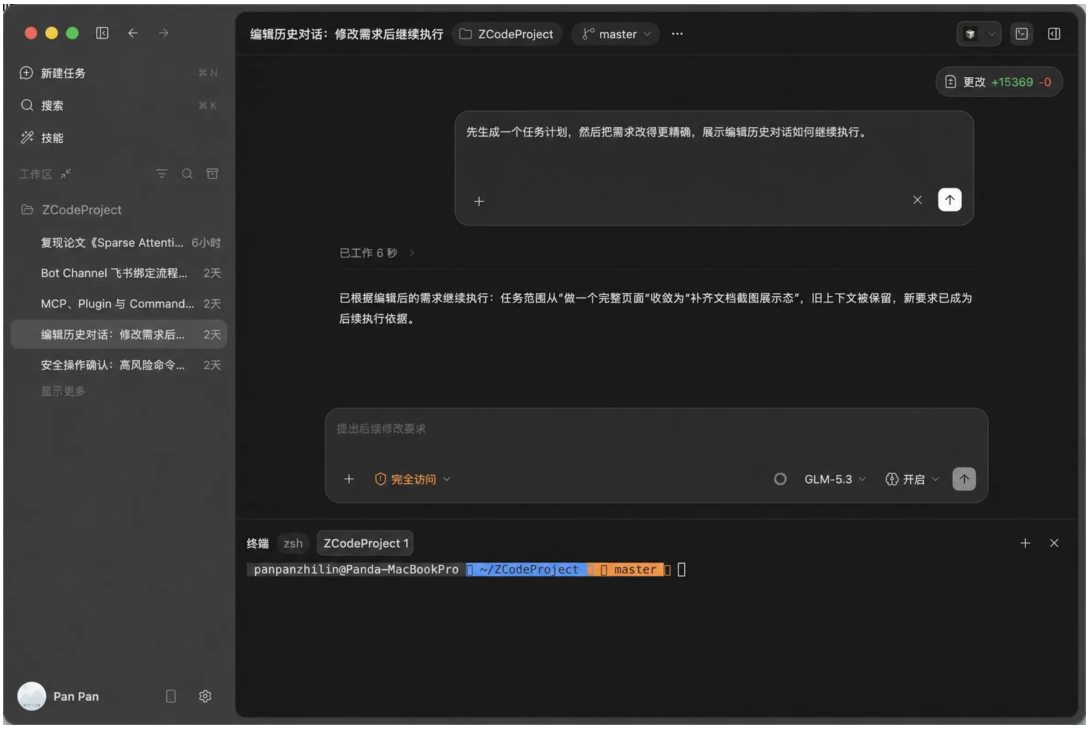
找到入口：在对话历史记录中，将鼠标悬停在自己发送过的消息上，消息右侧快捷操作区的**铅笔图标**就是编辑入口。



> **注意**：
>
> - 编辑入口**只出现在用户消息上**，不会影响 Agent 的回复内容本身；
> - **作用于最近一轮**：当前可编辑的是最后一轮的用户消息；任务运行中或有排队消息时暂不可编辑。

操作逻辑：

1. 点击用户消息右侧的铅笔图标进入编辑；
2. 在展开的输入框里调整原始指令，补充文件、路径、目标或限制条件（可继续使用 `@` 引用文件、`/` 插入命令）；
3. 点击发送让 Agent 基于新指令继续推进；点击取消则保留原消息不变。



11.3 连同文件一起重置

如果上一轮不只是说错了话，Agent 还已经按错误的理解改了文件，编辑器里的 **对话 + 文件重置** 会同时做两件事：

1. 把这一轮改动过的文件恢复原样；
2. 再用你改后的消息重新发送。

万一有文件无法安全恢复（比如你在此期间手动改过它），ZCode 会提示「文件无法安全重置」，你可以选择 **仅重置对话并发送**。

> **注意**：任务正在运行、正在压缩上下文，或者本轮压根没有可恢复的文件改动时，这个按钮不可用。

11.4 修改待发送的消息

Agent 还在回复时你接着输入的内容会进入**待发送消息**队列。发出去之前发现写错了，把鼠标移到队列面板里那一条上，点 **编辑**。

> **注意**：它的行为与"编辑历史对话"不同——不是在队列里原地改字，而是把整条内容连同附件一起**退回输入框**，该条从队列中移除，你改完重新发送。
>
> 两个前提：输入框里不能已经有草稿或正在上传的附件；`/compact` 这类压缩指令排在队列里时没有编辑入口，只能立即执行或删除。

11.5 撤销 Agent 改过的文件

每轮回复下方的文件变更摘要里有 **撤销** 按钮，点击后会弹出确认窗口，先告诉你这一轮里哪些文件可以安全撤销、哪些不能、哪些会被忽略，确认后才真正写回。撤销之后按钮变成 **重新应用**，可以再改回去。

核心规则：要么全撤，要么都不撤。 只要其中一个文件没法安全还原，ZCode 就一个都不动，不会让你落到改了一半的状态。

三类**无法**从这里撤销的改动：

1. ZCode 没有留下改动前内容的文件；
2. 你在此期间手动编辑过的文件；
3. **由终端命令造成的改动**（会被直接忽略，需要你自己用 Git 处理）。

11.6 典型使用场景

场景	做法
修正需求表达	编辑上一条用户消息，改写目标描述
补充关键上下文	追加文件路径、错误日志、截图说明、接口参数
调整任务方向	从"分析问题"改为"直接修复"，或从"实现功能"改为"先给方案"
减少重复对话	在同一上下文里继续，避免重新解释整段背景
上一轮改错了文件	用「对话 + 文件重置」同时回滚文件和对话

11.7 本章小结

- 编辑历史对话解决"话说错了"，对话 + 文件重置解决"事做错了"。
- 文件撤销是原子操作：全撤或不撤；终端命令造成的改动需自行用 Git 处理。
- 待发送队列的编辑是"退回输入框"，不是原地改字。

11.8 思考与练习

1. 故意给一条含糊指令让 Agent 改错文件，然后用「对话 + 文件重置」完整回滚，验证文件是否恢复。
2. 在一轮改动中手动编辑其中一个文件，再尝试撤销，观察"全撤或不撤"规则的提示。

第 12 章　远程开发：SSH / WSL / Docker

12.1 本章目标

- 判断自己需要哪种远程方式
- 完成 SSH / WSL / Docker 连接
- 把本地的 Skill / MCP / Plugin 同步到远端

12.2 它解决什么问题

远程开发用于把工作区放到远程主机、WSL 发行版或本机容器里运行。连接成功后，**文件读取、终端命令、Git 操作和 ZCode Agent 都会在目标环境内执行**；桌面端仍负责账号、模型配置、任务入口和界面交互。

适合场景：

- 项目代码和依赖只在远程 Linux 服务器上可用；

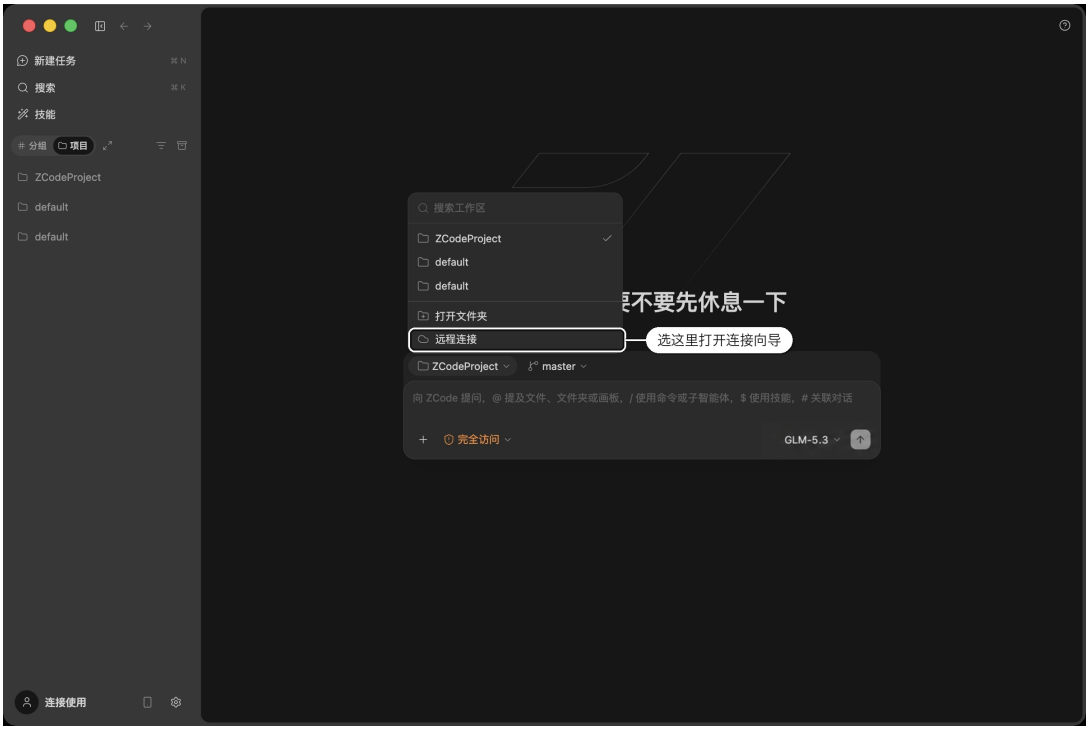
- 需要使用远程机器上的 GPU、内网服务、数据库或专用工具链；
- 在 Windows 上开发，但项目依赖 Linux 环境，希望直接在 WSL 里跑；
- 想在本机 Docker 容器中隔离项目依赖，不污染本机环境；
- 本机性能或系统环境不适合直接运行项目。

> **注意**：**目标环境需要 POSIX shell，当前不支持连接原生 Windows 远程主机。** 如果你要用的是 Windows 本机上的 Linux 环境，请选择 WSL。

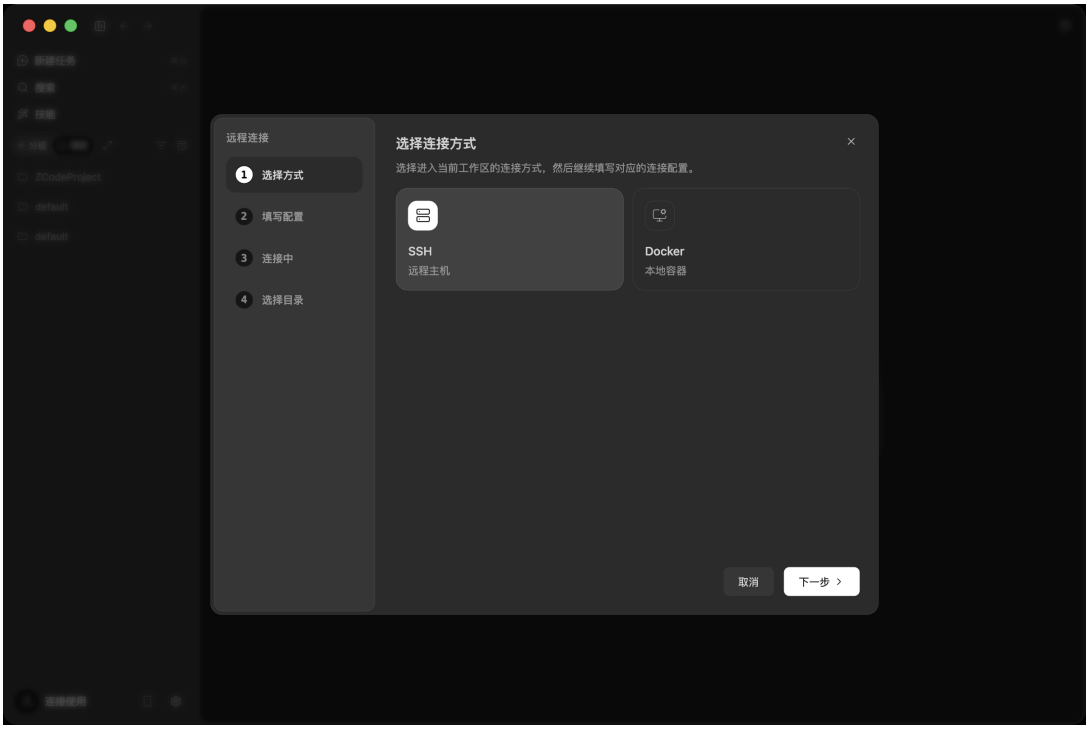
12.3 连接前准备

方式	前置条件
SSH	本机可通过 SSH 访问目标主机；知道主机地址、端口、用户名；目标主机支持密码登录或已准备好私钥文件（及可选口令）。若使用 `~/.ssh/config`，可先在终端执行 `ssh <Host别名>` 验证
WSL	你在 **Windows 桌面端**（这是 WSL 入口出现的前提）；本机已安装 WSL 且至少一个发行版可用；想指定某 Linux 用户时确认该用户存在
Docker	本机 Docker daemon 正在运行；目标容器已启动；容器内能访问项目目录并具备所需工具链

12.4 打开连接向导



在左侧边栏的 **项目** 区域点击右侧的 `+`，选择 **远程连接**。新任务页的工作区选择器里也能进入，主界面右下角的项目名称菜单同样保留了这个入口。



方式	连到哪里	出现条件
SSH	远程主机	全平台
WSL	本机的 Windows Linux 子系统	仅 Windows 桌面端
Docker	本机正在运行的容器	全平台

向导左侧显示当前步骤：选择方式、填写配置、连接中、选择目录。

12.5 SSH 连接



按顺序配置：

1. **SSH 配置别名（可选）**：ZCode 会读取本机 SSH config 中的可用别名，选择后自动填充主机、端口、用户名和私钥路径。

   > **注意**：ZCode 使用内置 SSH 客户端直连，别名只用于**预填以上四个字段**——`ProxyJump` / `ProxyCommand` 等跳板配置**不会生效**，含跳板的别名会被当作直连主机处理。需要经跳板访问的主机，请先在本地建立端口转发（如 `ssh -L`），再让 ZCode 连接转发端口。

2. **主机**：远程主机地址或 IP，例如 `192.168.1.100`、`dev.example.com`；
3. **端口**：默认 `22`；
4. **用户名**：例如 `root`、`ubuntu`、`deploy`；
5. **认证方式**：密码或私钥。私钥方式需填私钥文件路径，私钥有口令时再填口令。密码和私钥口令不会明文写进连接历史；连接成功后加密保存在本机凭据存储里供重连使用；
6. **资源下载方式**：
   - **本地下载后上传**（默认）：由本机下载资源后传到远端；
   - **远端服务器下载**：远端直接访问 ZCode CDN，可减少上传等待，但远端需要能联网，并具备下载、解压和校验工具。

填写完成后点击 **开始连接**，成功后选择要作为工作区打开的目录。

12.6 WSL 连接

WSL 入口**只在 Windows 桌面端显示。**配置只有两项，均可留空：

1. **发行版**：ZCode 会检测本机已安装的 WSL 发行版供你选择。留空时连接系统默认发行版；即使没有检测到发行版，也可以保留为空直接尝试连接默认发行版；
2. **Linux 用户**：决定 ZCode 在 WSL 中以哪个用户身份工作。留空时使用该发行版默认用户；填写后，Agent、终端命令、Git 操作以及文件的读取和创建都会使用这个用户的身份和权限。

> **注意**：**请谨慎使用 root。** 以 root 连接后，在工作区中新建的文件可能归 root 所有，之后使用普通用户时可能无法修改。除非确有必要，建议使用日常开发所用的普通用户。

12.7 Docker 连接

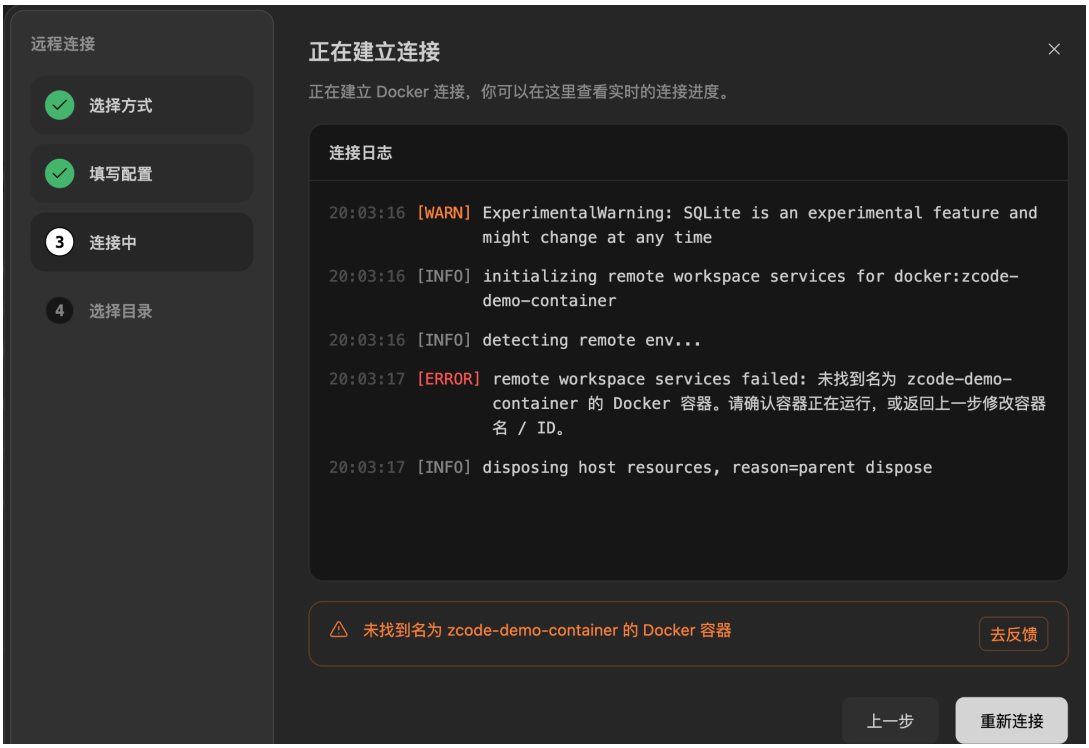


Docker 页面用于选择或填写本机正在运行的容器：

- 若检测到运行中的容器，从 **选择正在运行的容器** 中选择（打开下拉框时 ZCode 会刷新一次容器列表）；
- 若列表为空，可在 **容器** 输入框手动填写容器名或容器 ID，例如 `zcode-dev`、`7f3a8c2d9e10`。

Docker 连接通过 `docker exec` 和 `docker cp` 进入本机容器。若容器尚未启动，先在终端运行 `docker ps` 确认可见。

12.8 首次连接与连接日志



点击 **开始连接** 后进入 **连接中** 页面，显示实时连接日志：环境检测、远端资源准备、运行时初始化和失败原因。

- 首次连接通常比后续慢，因为需要在目标环境准备远端服务和 Agent runtime；后续连接复用已准备好的资源，只在版本变化或资源缺失时重新安装；
- 连接失败时页面底部显示错误摘要，可点 **上一步** 修改配置后重试，或点 **去反馈**（ZCode 会带上连接日志和截图）。

12.9 进入远程工作区后

连接成功并选择目录后，可以像本地工作区一样使用：文件树查看编辑远程文件、让 Agent 阅读代码 / 修改文件 / 运行测试 / 执行命令、使用终端操作远程 shell、使用 Git 查看状态分支和提交、从左侧工作区列表重新进入或重连。

> **注意**：远程工作区的项目文件、命令执行和 Agent runtime 都在目标环境内；**模型供应商、登录账号、全局设置和用量信息仍保存在桌面端。**

12.10 把本地配置同步到远端

远程工作区里的 Agent 跑在目标主机上，因此本机装的技能、配置的 MCP 服务器和插件默认不会跟过去。ZCode 提供三个同步入口：

同步项	同步什么
Skill	你本机的**用户级**技能
MCP	你本机的**用户级** MCP 服务器配置
Plugin	你本机的用户级插件；来自市场的插件会在远端重新安装

入口位置：连接向导里 SSH 和 WSL 走到 **选择远程目录** 这一步时即可同步；连上之后，工作区标题栏的 **同步** 下拉里有三项；侧边栏工作区行的「更多」菜单，以及设置里对应的技能、MCP、插件页面也都有同一入口。本地工作区不显示这些菜单。

共同规则：

- **逐条勾选、手动触发**，不会自动同步；
- **远端已存在同名项一律跳过，不覆盖**，列表里会标出来并默认不勾选；
- 只同步**用户级**内容。项目级技能、工作区级 MCP、插件自带的 MCP 都不在范围内；
- 远端目录写不进去或连接超时，ZCode 会在开始复制前就告诉你；
- 部分失败不影响已成功的项，失败项会单独列出原因。

四条安全提醒：

1. **同步不保证能用**：远端是否装了 `npx`、Python 依赖或技能需要的命令行工具，ZCode 不做检查。同步过去的东西起不来，通常要你到远端手动补依赖；
2. **MCP 配置里的密钥会原样写到远端配置文件**：只对你信任的主机做这个操作；
3. **插件的敏感配置项不会同步**：标记为敏感的配置和路径类配置会被跳过；市场插件在远端重新安装，**远端需要能访问对应的市场地址**；

4. **同步过去的插件会在远端执行**：插件里的技能、命令、Hooks 和 MCP 都会在远端环境里加载运行，同步前请确认插件来源可靠。

目前 **SSH 和 WSL** 支持这些同步操作。

12.11 常见问题

问题	处理
SSH 配置别名没有出现	检查 `~/.ssh/config` 是否存在、格式是否正确，并在终端执行 `ssh <别名>` 验证。即使别名没有出现，也可以手动填写主机、端口、用户名和认证信息
什么时候选"远端服务器下载"	当远端访问 ZCode CDN 更快或本机上传很慢时。远端需具备 `curl` 或 `wget`、`tar`，以及 `sha256sum` / `shasum` / `openssl` 中的校验工具。远端无法访问公网时（内网、离线环境）使用默认的本地下载后上传：本机下载后经 SFTP 传到远端并安装到 `~/.zcode/server`
远程连接走 HTTP 代理吗	**不会**。SSH 连接和 Web 远程控制通道都不经过设置中的 HTTP 代理。远端服务器自身需要代理时，请在远端 shell 环境中自行配置
Docker 页面没有容器	先执行 `docker ps`；没有运行中的容器先启动；终端能看到但 ZCode 列表为空时，手动输入容器名或容器 ID
连接失败看哪里	优先看**连接日志**最后一条 `ERROR` 或页面底部错误摘要。常见原因：SSH 主机不可达、端口错误、用户名或密码错误、私钥口令错误、远端缺少下载 / 解压工具、Docker 容器未运行。信息不足时点 **去反馈**
手机端能新建远程连接吗	不能创建全新的 SSH / WSL / Docker 连接，填写参数必须在桌面端完成。但若某个远程工作区已登记在当前桌面窗口里、只是显示为断开，Remote Control 可以发起重新连接

12.12 本章小结

- SSH 连远端、WSL 连本机 Linux、Docker 连本机容器；目标环境必须是 POSIX shell。
- SSH 别名只做字段预填，跳板配置不生效，需本地端口转发。
- 同步三件套（Skill / MCP / Plugin）只同步用户级、不覆盖同名、密钥原样写入远端。

12.13 思考与练习

1. 用 SSH 连接一台测试主机，完成首次连接后查看连接日志，记录耗时最长的阶段。
2. 把一个本地 Skill 同步到远端，然后让远端 Agent 调用它，验证是否可用。

第 13 章　Remote Control 与 Bot Channel

13.1 本章目标

- 用手机扫码接管桌面端会话
- 配置微信 / 飞书 Bot 长期回访任务
- 理解"控制界面"与"执行位置"的分离

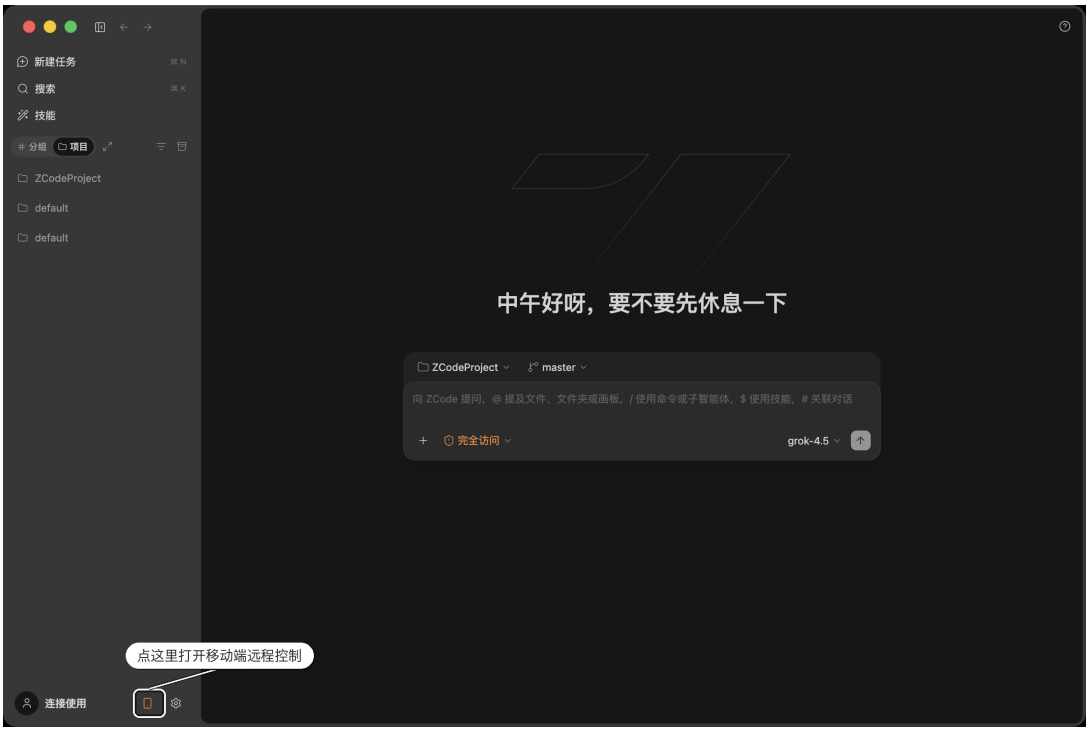
13.2 Remote Control：扫码临时接管

Remote Control 把当前 ZCode **桌面窗口**临时开放给手机访问。扫码之后，可以在手机上查看和切换这个窗口里已打开的工作区、任务和会话，继续给 Agent 发指令。

> **注意**：手机端是一层**控制界面**，它不同步代码，也不新建运行环境。你在上面发出的每一条指令，都由桌面端那个工作区原本连着的机器去执行。

适合场景：任务已在桌面端跑起来、需要离开电脑后继续看进度；临时收到反馈需要用手机快速追加指令；长任务执行中需要随时检查状态。

连接步骤：在 ZCode 左下角侧栏点击**手机图标**，打开 **移动端远程控制**。弹窗左侧生成二维码和连接地址，右侧是 Bot Channel 入口。





用手机扫码，或复制链接在手机浏览器打开。

> **注意**：连接成功后，手机端接入的是**整个桌面窗口**，而不只是扫码时停留的那个工作区。

手机端能做什么

任务首页：按 **工作区** 或 **时间线** 组织任务；按 **创建时间** 或 **更新时间** 排序；刷新列表拿最新状态；在已有工作区里新建任务；对显示为「未连接」的远程工作区发起 **重新连接**。每个工作区会标出类型（本地 / 远程 / 对话），任务右侧显示运行状态。

会话页：Agent 的回复、执行过程和进度；底部输入框可直接补充需求；回复下方提供复制、点赞 / 点踩反馈；会话满足条件时还可以分叉。

> **注意**：同一时间只支持一个手机页面连接。如果已经在别的页面打开过，需要先关掉旧页面。

命令在哪台机器上执行

工作区类型	代码和命令在哪执行
本地	你的桌面电脑
SSH	对应的远程主机
WSL	对应的 WSL 发行版
Docker	对应的容器

停止与安全

- **关闭弹窗不会停止远控**，用完之后要点 **停止** 才真正结束；
- **怀疑地址外泄就刷新二维码**：刷新会让之前复制或扫过的地址立即失效；
- **不要公开发布或转发连接地址**，它等同于一把临时钥匙；

- 远控期间桌面端需要保持运行和联网。

13.3 Bot Channel：把入口放进聊天工具



机器人管理：把聊天工具和 Webhook 接入 ZCode　　远程控制弹窗里的 Bot Channel 入口

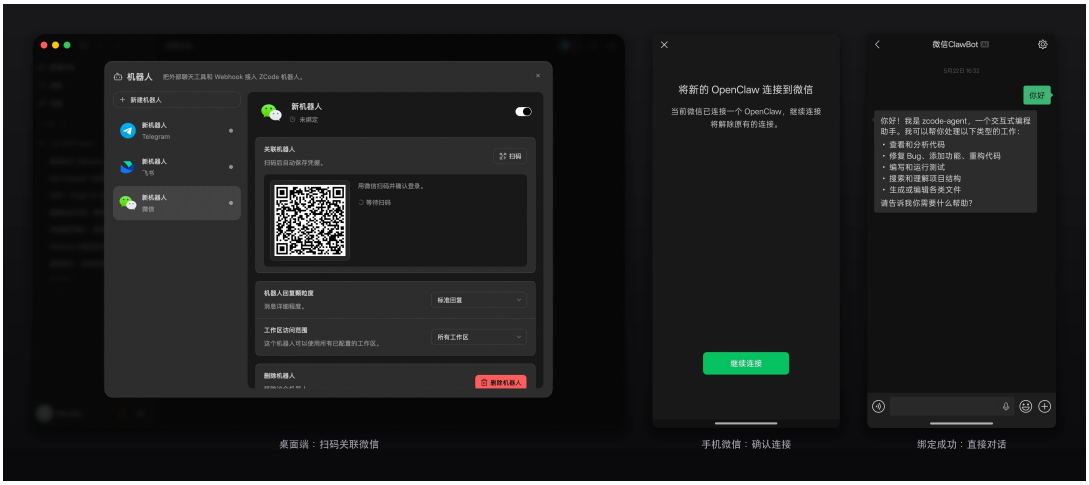
Bot Channel 把外部聊天工具接入 ZCode。配置完成后，可以从微信或飞书会话里打开当前工作区，查看任务进度，并继续给 Agent 发送指令。

入口：打开 **移动端远程控制** 弹窗，右侧就是 Bot Channel 入口；点击对应聊天工具开始接入，或进入底部 **机器人管理** 查看已有机器人。



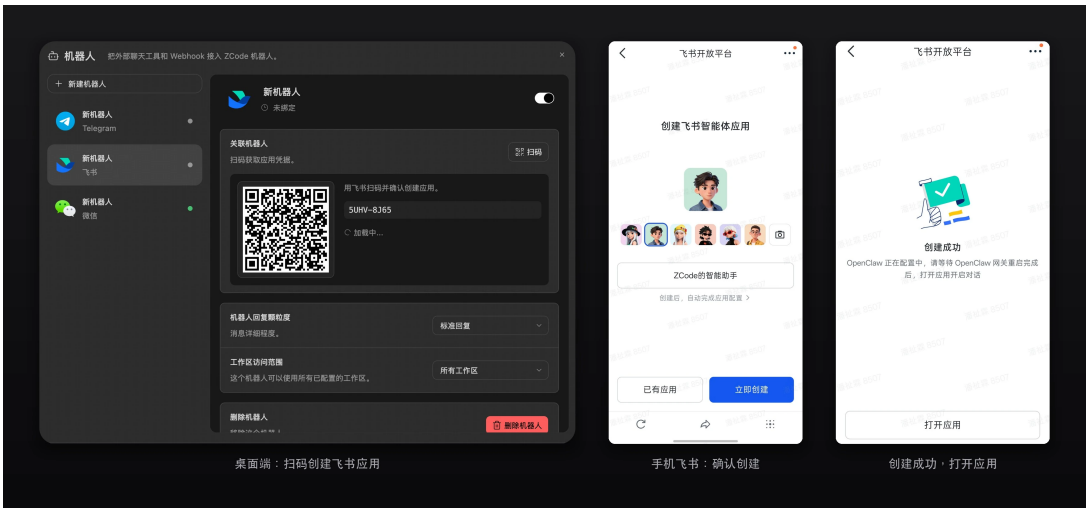
目前可创建的渠道：**微信**（扫码登录，自动绑定）、**飞书**（扫码创建应用，再通过消息绑定）。钉钉、Discord 和企业微信会在后续版本继续补齐。

微信配对流程：在机器人页选择 **微信**，用手机微信扫码并确认连接，凭据自动保存。绑定完成后，直接在微信里和这个 Bot 对话即可。



桌面端：扫码关联微信　　手机微信：确认连接　　绑定成功：直接对话

飞书配对流程（两步）：



桌面端：扫码创建飞书应用　　手机飞书：确认创建　　创建成功，打开应用

1. **创建应用**：在 ZCode 里选择 **飞书** 并扫码，手机会打开飞书开放平台，确认智能体应用的名称和头像后点击 **立即创建**，等待配置完成；
2. **绑定会话**：ZCode 生成一段限时有效的**绑定命令**（形如 `/bind C67356`），在飞书里向这个智能体发送该命令，机器人回复绑定成功并列出全部可用命令。

绑定成功后，飞书 Bot 会返回可用命令（查看状态、新建任务、切换项目、切换模型、切换运行模式、调整回复详细度等）。



桌面端：生成 /bind 绑定命令　　飞书里发送 /bind 完成绑定

管理机器人：创建后机器人出现在左侧列表，可以启用或停用、查看连通状态、设置回复颗粒度，并限制它可以访问的 workspace。常用配置包括：

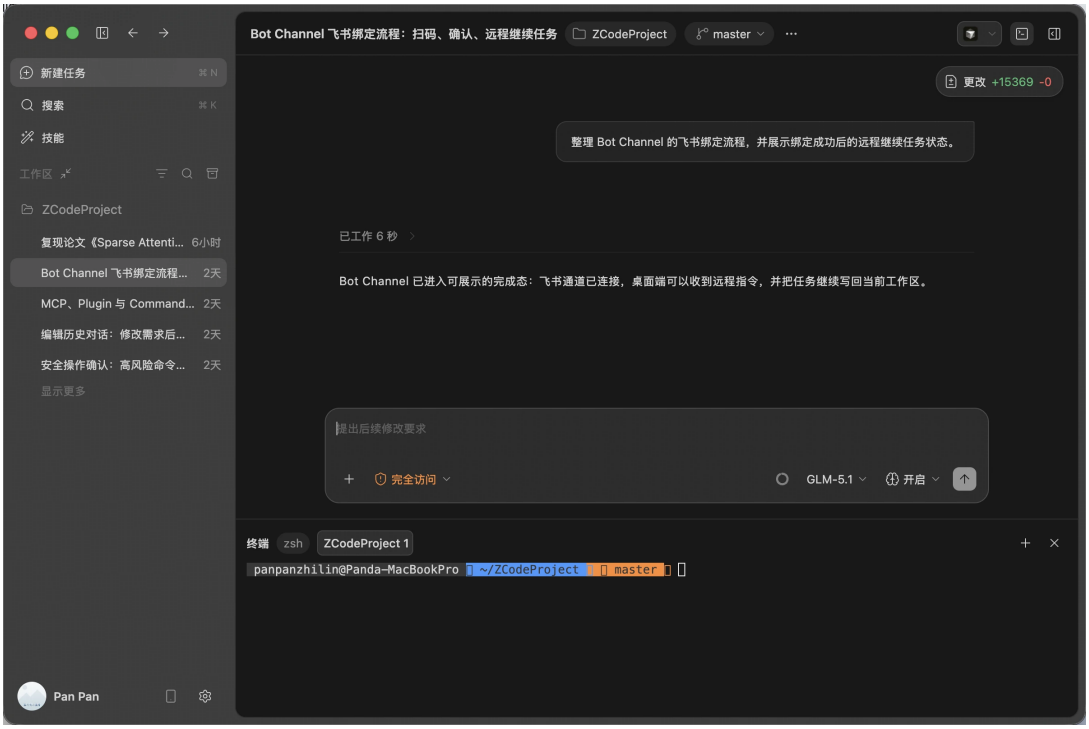


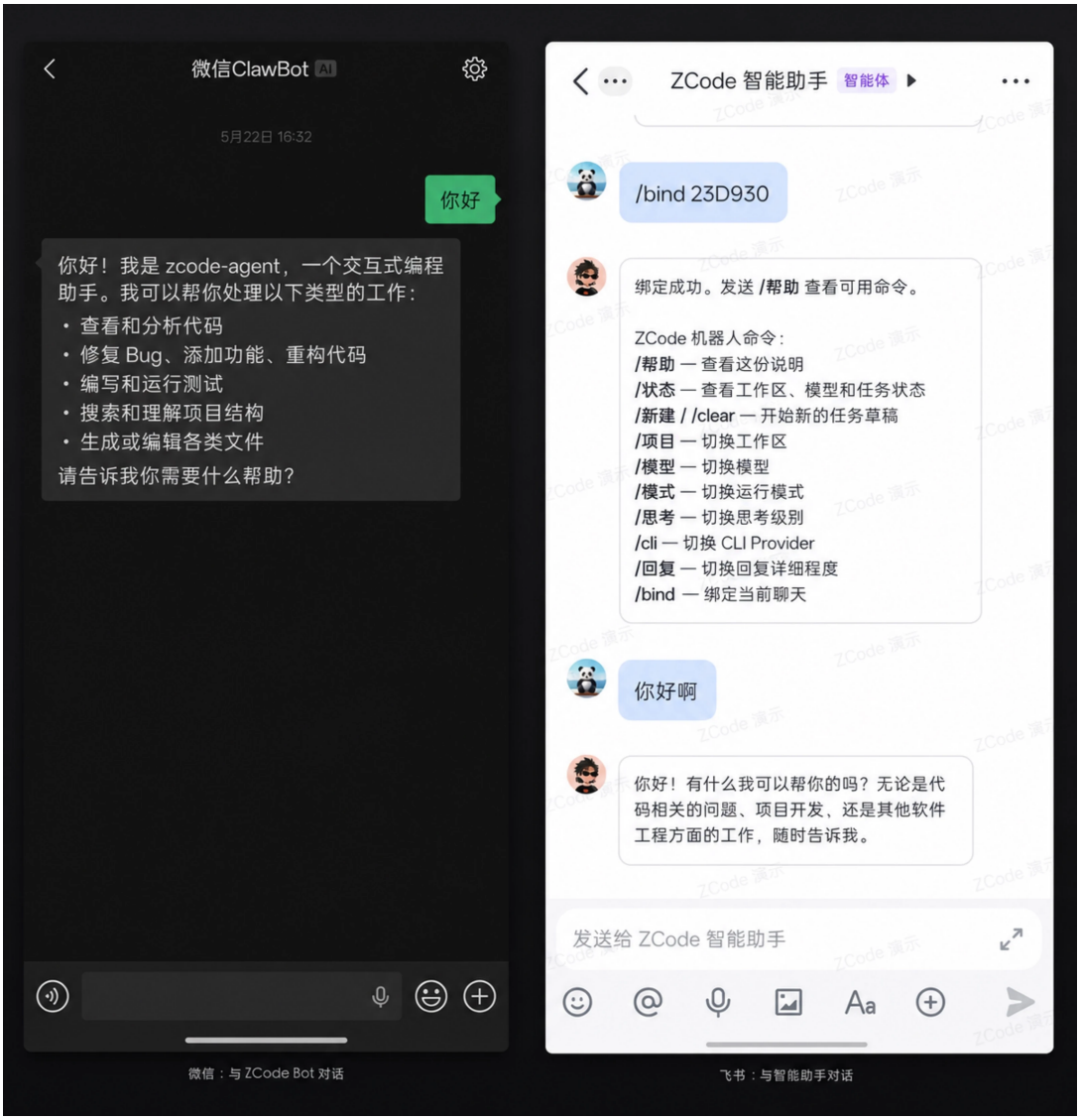
机器人管理：连通状态、回复颗粒度与工作区访问范围

- **关联机器人**：扫码或按提示完成凭据绑定，已连通的可随时解绑；
- **机器人回复颗粒度**：控制回复详细程度；
- **工作区访问范围**：限制机器人可打开的工作区；
- **删除机器人**：移除不再使用的渠道。

飞书的流式卡片回复：飞书和 Lark 渠道统一使用**流式卡片**回复，没有颗粒度选项（标准、完整、摘要这几档只对其他渠道生效）。实际效果是 Agent 的回答不再拆成一条条消息刷屏，而是在**同一张卡片里原地更新**；卡片内容按时间顺序穿插正文和工具调用，工具部分只显示折叠起来的摘要。

当 Agent 需要你做决定时（选择方案、授权操作、确认计划），当前卡片会先定格保留，用一张单独的卡片来问你；答复后，后续输出会开一张新卡片继续。





13.4 用扫码还是用 Bot

维度	Bot Channel	Remote Control
入口形式	常驻在微信 / 飞书会话里	手机扫码，用完即走
适合	长期反复回访同一个工作区	临时看一眼
相同点	两者操作的都是桌面端已有的会话，代码仍然跑在桌面端原本连着的那个环境里	

> **注意**：如果你要改变的是**代码运行的位置**，那属于远程开发（第 12 章），不是这两者。

13.5 本章小结

- Remote Control 是临时钥匙，Bot Channel 是常驻入口；二者都是控制界面，代码始终跑在桌面端连着的机器。
- 扫码地址等同临时凭证：用完要点停止，怀疑泄露就刷新二维码。

- 飞书用流式卡片，微信扫码即绑。

13.6 思考与练习

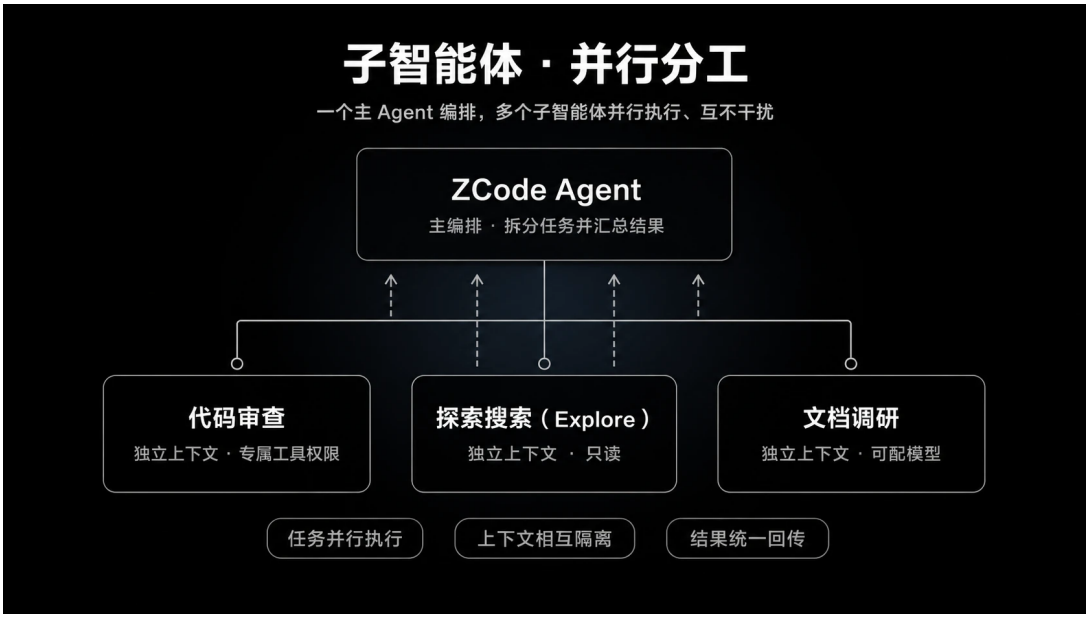
1. 用手机扫码接入一个正在运行的目标模式任务，追加一条指令并观察桌面端是否正确执行。
2. 配置飞书 Bot，用 `/bind` 完成绑定，然后列出它返回的全部可用命令。

第 14 章　子智能体：并行与分工

14.1 本章目标

- 理解两个内置子智能体的分工
- 创建自定义子智能体
- 掌握工具白名单与 MCP 的边界规则

14.2 概念



子智能体是主 Agent 可以在**独立上下文**中启动的专用 Agent，完成后把结果汇总回主对话。当主 Agent 判断任务需要独立上下文或并行调研时，会通过 Agent 工具启动子智能体。

ZCode 内置 **general-purpose（通用型）** 和 **Explore** 两个子智能体，另外可以在设置中创建自己的**用户级子智能体**。

14.3 内置：general-purpose

默认通用型子智能体，**拥有完整工具权限**，适合在独立上下文中处理需要读取、修改、运行命令或综合推进的任务。

适合交给它：

- 独立实现一个小功能或修复一个明确问题；
- 在另一个上下文里整理文件、运行验证命令，并回传结果；

- 把可并行的文档、代码或配置任务拆开处理。

14.4 内置：Explore

只读的文件搜索和代码库调研专家，适合大范围检索、调用链梳理、代码结构理解和证据收集。**不会创建、修改、移动或删除文件。**

适合交给它：

- 查找某个能力在哪些文件中实现；
- 梳理模块入口、调用链和关键依赖；
- 在改动前先调研相关代码和风险点；
- 并行搜索多个目录、命名方式或实现路径。

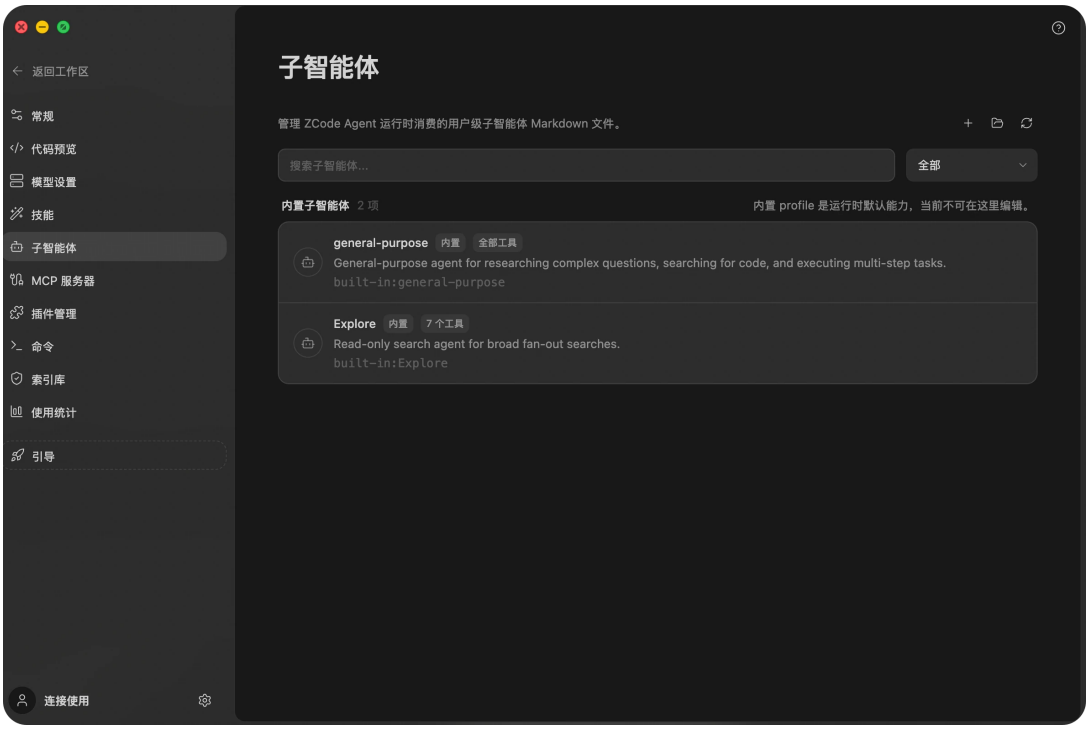
可以直接在提示词里要求：

```
请先用 Explore 调研这个模块的调用链，再总结修改入口和风险点。
```

```
请用只读方式搜索这个功能的实现位置，列出相关文件和证据。
```

14.5 自定义子智能体（Beta）

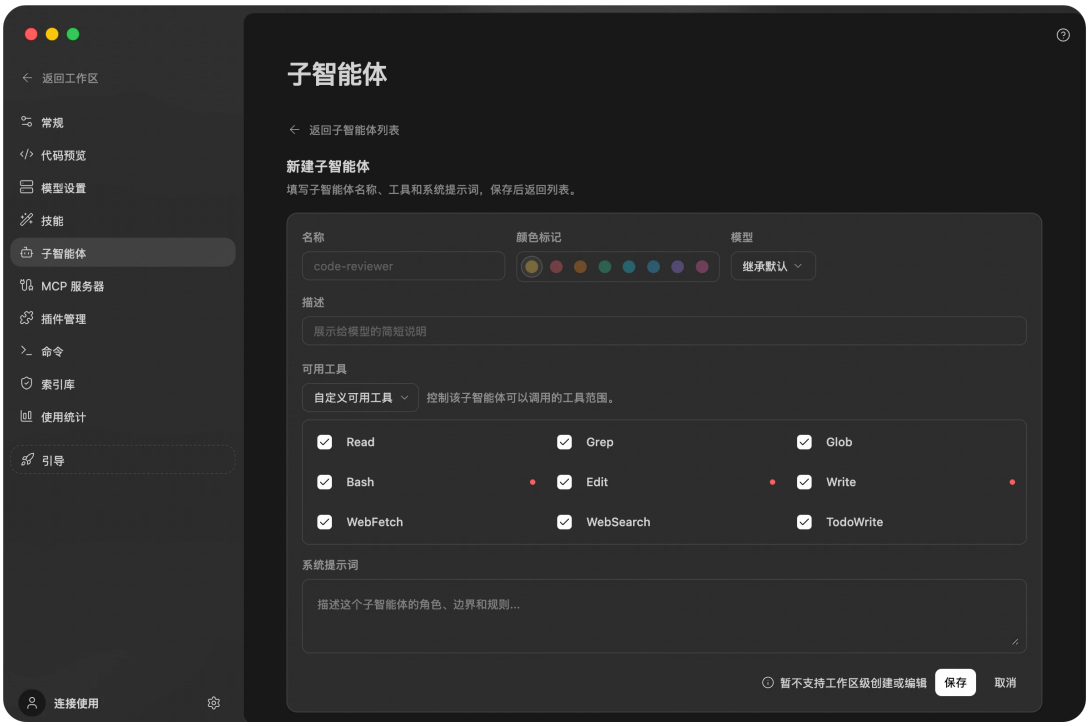
> **Beta**：用户级自定义子智能体正在灰度上线，能力范围后续可能调整。



在 **设置** → **子智能体** 中可以：

- **查看**可用子智能体（内置 `general-purpose`、`Explore`）；
- **新建**子智能体，填写名称、颜色、模型、思考强度、描述、可用工具和系统提示词；
- 对自己创建的子智能体进行**编辑、删除、启用或停用**（内置角色不能编辑正文或删除，但可以为它们单独指定模型和思考强度）。

新建表单字段：



字段	说明
名称	标识，例如 `code-reviewer`
颜色标记	在列表和会话中区分身份，仅作标识
模型	「继承默认」（跟随主 Agent 当前模型）或指定具体模型
思考强度	单独设置推理档位，**仅在指定了具体模型时生效**；选择「继承默认」时此项不参与
描述	展示给主 Agent 的简短说明。**描述越准确，越容易被自动选用**
可用工具	控制工具范围。「默认所有权限」继承全部工具；「自定义可用工具」逐个勾选
系统提示词	描述这个子智能体的角色、边界和规则

保存后，ZCode 把子智能体写入 `~/.zcode/agents/<name>.md`，运行时在下次运行时加载它。启用后可以让 Agent 自动选用，也可以在聊天输入框中用 `@` 引用它。

14.6 定义文件字段参考

子智能体定义文件是带 frontmatter 的 Markdown，**正文即系统提示词**。除表单字段外，手工编辑时还支持（camelCase，大小写敏感）：

字段	说明
`name` / `description`	**必填**，缺失时该定义文件会被忽略并产生诊断

字段	说明
`model`	指定具体模型；写 `inherit` 或不填表示跟随主 Agent 当前模型
`thoughtLevel`	思考强度档位（如 `high`）。**仅在同时配置了具体 `model` 时生效。**注意字段名不是 `reasoningEffort`——不认识的字段会被静默忽略，不报错
`color`	颜色标记（预设色）
`tools` / `disallowedTools`	可用 / 禁用工具列表
`maxTurns`	单次调用的最大轮数（正整数）
`injectAgentsMd`	是否注入 AGENTS.md，默认注入；设为 `false` 可关闭
`mcpServers`	声明依赖的 MCP 服务名列表（精确匹配）；声明的服务未连接时，调用会**直接失败**

> **注意（生效时机）**：修改定义文件、或在设置页调整模型 / 思考强度后，需要**新建会话**才会生效，已启动的会话不热更新。**例外**：切换主会话模型——未指定 `model` 的子智能体在后续调用中会立即跟随新模型。

示例定义文件：

```
---
name: code-reviewer
description: 审查当前改动，重点关注回归风险、测试覆盖与命名一致性
model: inherit
color: green
tools:
  - Read
  - Grep
  - Glob
maxTurns: 20
---

你是一名资深代码审查员。审查时：
1. 先读懂改动意图，再评价实现；
2. 只指出能给出依据的问题，不提风格偏好；
3. 输出按「阻断 / 建议 / 可选」三级分类，每条目附文件与行号。
```

14.7 工具与 MCP 边界

子智能体能用哪些工具，取决于 `tools` 字段的配置方式：

配置	结果

配置	结果
`tools` 留空或 `*`	继承主会话全部工具，**包括已连接的 MCP 工具**
自定义可用工具（设置页勾选）	列表里写了什么就只有什么。设置页勾选列表只包含内置工具（`Read` / `Grep` / `Glob` / `Bash` / `Edit` / `Write` / `WebFetch` / `WebSearch` / `TodoWrite`），因此**勾选自定义后 MCP 工具会全部不可用**（`WebSearch` / `WebFetch` 不是 MCP 工具，不受影响）
手工编辑 `tools`	可把 MCP 工具**全名**逐个写进 `tools`，格式为 `mcp__<服务名>__<工具名>`。通配写法（如 `mcp__server__*`）无效，会被静默忽略

另外两条边界：

- 子智能体只能看到主会话**启动时**已连接的 MCP 服务，会话中途新连接的服务对子智能体不可见（新建会话即可）；
- **子智能体内不能再派发子智能体。**

14.8 项目指令注入

自 **v3.7.1** 起，子智能体默认注入用户级 `~/.zcode/AGENTS.md` 与工作区 `AGENTS.md`，与主 Agent 保持一致；frontmatter 中设置 `injectAgentsMd: false` 可关闭。**内置的 Explore 默认不注入。** v3.7.1 之前的版本，子智能体不注入 AGENTS.md。

14.9 前台与后台执行

方式	行为	适合
前台执行	同时启动多个时并行跑，主任务等它们全部完成后再继续	结果马上就要用上的调研
后台执行	主任务不必等待，可以继续推进甚至先结束当前回合；跑完后自动回到主对话	耗时较长、不该打断当前思路的调研

> **注意**：出于安全考虑，后台运行的 `Explore` 子智能体**只有只读工具**（读取文件、按文件名查找、按内容搜索），不能修改任何东西。

14.10 范围与限制

- **仅限用户级**：当前 Beta 管理的是全局 / 用户级子智能体，存放在 `~/.zcode/agents/`。暂不支持在设置中创建或编辑**工作区 / 项目级**子智能体；
- **内置角色不可删除或停用**：`general-purpose` 和 `Explore` 不能编辑正文、删除或停用，名称也不能被复用；但可以在设置页为它们单独指定模型和思考强度；
- **后台执行由 Agent 自己决定**：你不需要手动切换，也没有单独的后台管理界面。

14.11 本章小结

- general-purpose 全能、Explore 只读调研，先判读写性质再选。
- 自定义子智能体的 `description` 决定它能否被自动选中，要写清"什么时候用"。
- 自定义工具白名单会让 MCP 工具全部失效，需要时手工写 `mcp__<服务名>__<工具名>`。

14.12 思考与练习

1. 创建一个 `test-writer` 子智能体，工具只给 `Read` / `Grep` / `Glob` / `Write`，用它为某个模块补测试。
2. 在同一个任务里让 Agent 并行派发两个 Explore 子智能体，分别调研前端路由与后端接口，观察结果汇总方式。

第三部分　扩展机制篇

ZCode 提供五类扩展机制，构成了从"一句话"到"一套团队工程规范"的完整阶梯：

机制	本质	复杂度	分发方式
Command	一段可复用的提示词	最低	文件 / 插件
Skill	一套可复用的工作方法	低	文件 / 插件
MCP	外部工具与数据源接入	中	配置文件 / 插件
Hooks	事件驱动的自动动作	高	配置文件 / 插件
Plugin	以上能力的打包单元	—	插件市场

第 15 章　Skill：可复用的工作方法

15.1 本章目标

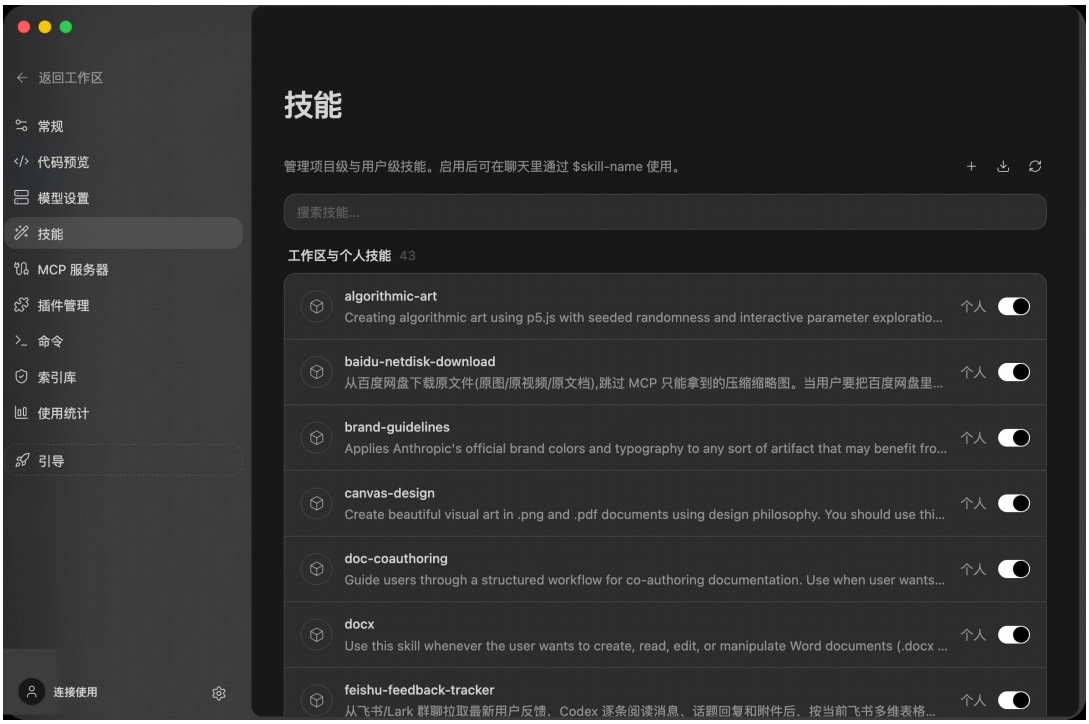
- 理解 Skill 的定位与目录结构
- 写出能被自动触发的 `SKILL.md`
- 排查"面板看得到、模型却不主动用"的问题

15.2 概念

Skill 是一组**可复用的工作说明**。通常用一个 `SKILL.md` 描述触发场景、处理步骤和输出要求，让 Agent 在遇到特定任务时按同一套标准执行。

与 Command 的区别：Command 保存"一段提示词"，Skill 保存"一套工作方法"（可含脚本、模板、示例文件、检查清单）。

15.3 管理技能



进入 **设置** → **技能** 后，可以按来源查看技能。列表展示技能名称、来源标签、描述和启用开关。可以：

- 通过搜索框按名称筛选；
- 使用右侧开关启用或停用；
- 点击右上角 **新建技能**，ZCode 会引导你在聊天中用 Agent 生成一个新技能。

15.4 创建技能

一个技能本质上是一个**目录 + 一个 `SKILL.md` 文件**，目录名就是技能名，聊天中也用这个名字引用。

ZCode Agent 的用户级技能目录：

```
~/.zcode/skills/<skill-name>/SKILL.md
```

最小可用示例：

```
---
name: code-review-checklist
description: Review code changes with a focused checklist for
correctness, regressions, tests, and maintainability.
---

# Code Review Checklist

Use this skill when reviewing a pull request, merge request, or
local diff.

Focus on correctness, regressions, missing tests, risky API changes,
and maintainability.
```

> **注意**：创建或修改后，回到 **设置** → **技能** 点击 **刷新**，确认技能出现在对应来源下，并保持开关开启。

15.5 格式限制

限制	后果
frontmatter 必须包含 `name` 和 `description`	缺失时该技能会被忽略，并在设置页的诊断里给出原因
`description` 上限 **1024 字符**	超出后**整个技能会被丢弃**（不是截断），诊断显示「description 超过 1024 字符」。长说明请放进正文
`SKILL.md` 正文超过 **100KB**	会被截断加载

15.6 技能数量与上下文占用

每轮对话会把所有**已启用**技能的元数据（名称 + 描述摘要，单条描述最多 250 字符）注入模型上下文，**正文只在技能被调用时按需加载。**

> **注意**：全部技能的元数据共享一个**固定预算**。安装大量技能导致超出预算时，注入会降级为只保留技能名，模型将难以判断何时该用哪个技能，自动触发率会明显下降。**建议只启用常用技能**——停用的技能不注入上下文、也不能被调用。

15.7 「面板里能看到，模型却不主动用」排查

`/` 面板与模型看到的是同一份技能清单，不存在"只给面板、不给模型"的开关。按以下顺序排查：

1. **元数据被降级**：技能装得太多、超出预算后模型只能看到技能名（描述丢失），自动触发率骤降——**停用不常用的技能**即可恢复；
2. **描述写得太泛**：`description` 没有写清"什么时候用"，模型无从判断。**触发条件写得越具体，自动触发越可靠**；

3. **在子智能体里**：子智能体定义文件若声明了自定义 `tools` 白名单，技能工具可能不在允许列表内，子智能体便无法调用任何技能——**删掉 `tools` 字段或把技能工具加入白名单**；
4. **所属插件被停用**：插件停用后其技能立即从会话移除，面板列表刷新可能略有延迟。

15.8 分发技能

ZCode **没有独立的技能市场。**想把一组技能分发给团队，可以把它们打包成**插件**（`skills/<技能名>/SKILL.md` 的**单层目录结构**），通过插件市场分发——插件市场支持自建源（GitHub 仓库、git URL 或本地目录）。

> **注意**：插件内的技能必须放在**单层目录**下，嵌套分组目录里的技能不会被 Agent 识别。

15.9 从外部 Agent 导入技能



如果已在 Claude Code、Codex CLI、OpenClaw、Augment、Windsurf 等工具里维护了技能，不必重新创建。在 **设置** → **技能** 页面右上角点击 **导入** 图标，ZCode 会自动扫描这些外部 Agent 的技能目录。

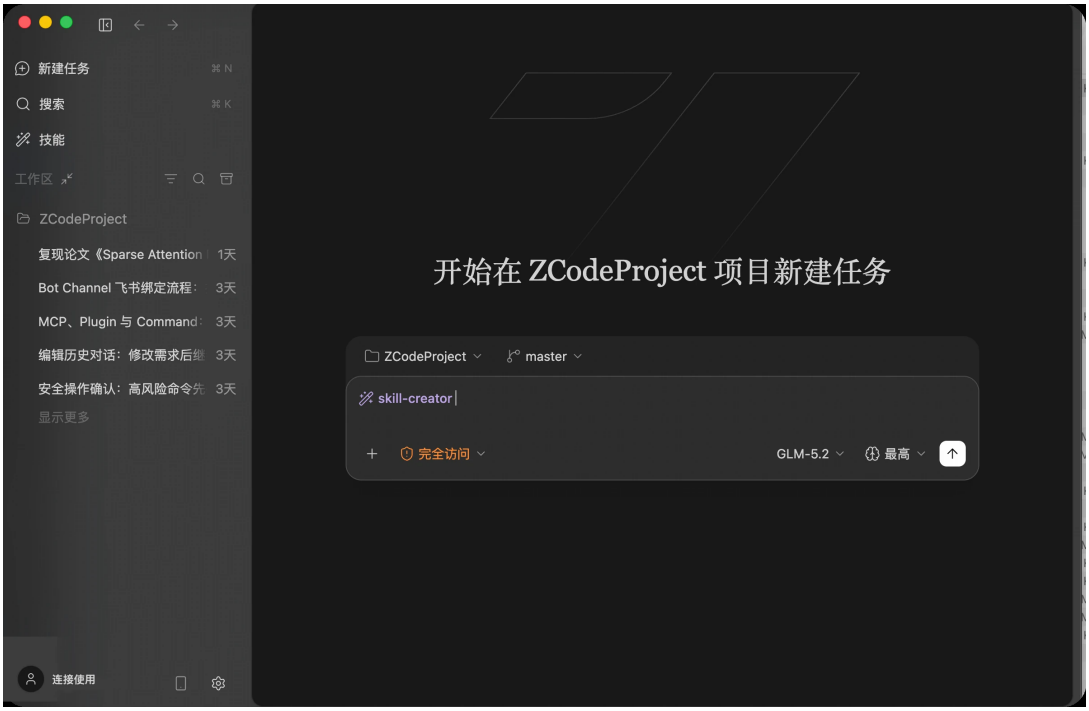
弹窗中可以：

- 按外部 Agent 分组查看检测到的技能，每个来源显示对应目录路径和可导入数量；
- 勾选需要的技能，或使用 **全选**；
- 选择 **导入方式**：

    - **软链**：创建指向外部技能目录的链接。ZCode 会跟随来源目录的后续变更，但依赖来源路径持续可用；
    - **复制**：复制成 ZCode 内的独立副本，与来源解耦；
- 选择 **导入目标**：导入到 **全局**（用户级）或当前 **项目**（仅当前工作区可用）。

导入的技能与自建技能一样可以启用、停用，并在聊天中通过 `$skill-name` 调用。

15.10 在聊天中使用



在输入框中输入 `$`，选择需要的技能；选中后技能以标签形式出现在输入框里，可继续补充具体需求。

```
$code-review-checklist 帮我 review 当前改动
```

```
$release-notes 根据这次提交写一版发版说明
```

除了 `$`，斜杠菜单里也有「技能」分组，输入 `/` 后同样能找到它们。

15.11 内置的配置向导技能

ZCode 自带一个 `zcode-configuration-guide` 技能，随应用安装、默认启用。它把技能、命令、MCP、Hooks、插件和 `AGENTS.md` 这六类扩展的配置位置、作用域和冲突规则整理在一处：

```
$zcode-configuration-guide 我想给这个项目加一个只在本仓库生效的命令，该放哪
```

配套还有几个诊断技能，专门排查扩展不生效的问题——技能没被识别、命令调不出来、MCP 连不上、Hooks 没触发、插件装不上。

15.12 同步到远程主机

用户级技能存在本机，远程工作区里的 Agent 默认看不到它们。连上 SSH 或 WSL 远程后，可以在工作区标题栏的 **同步** 下拉里选择 **同步 Skill**，把它们复制到远端（详见第 12 章）。

15.13 适合沉淀成 Skill 的内容

- 任务有固定流程（code review、接口排查、发版说明、测试报告）；
- 团队对输出格式有固定要求；
- 任务需要附带背景知识、检查清单、模板或示例；
- 同一个能力会在多个项目或多次对话里重复使用。

> **建议**：如果只是保存一句简单提示词，优先用 Command；如果需要一套完整工作方法，就更适合做成 Skill。

15.14 本章小结

- Skill = 目录 + `SKILL.md`，`name` / `description` 必填，描述上限 1024 字符（超出整体丢弃）。
- 元数据共享固定预算：技能装太多会导致自动触发率下降，**只启用常用的。**
- 描述要写清"什么时候用"，这是自动触发率的关键。

15.15 思考与练习

1. 为你的团队写一个 `api-change-checklist` 技能，描述中明确写出触发条件，观察它是否被自动调用。
2. 停用一半不常用技能，对比同一任务的自动触发率变化。

第 16 章　Command：可复用的提示词

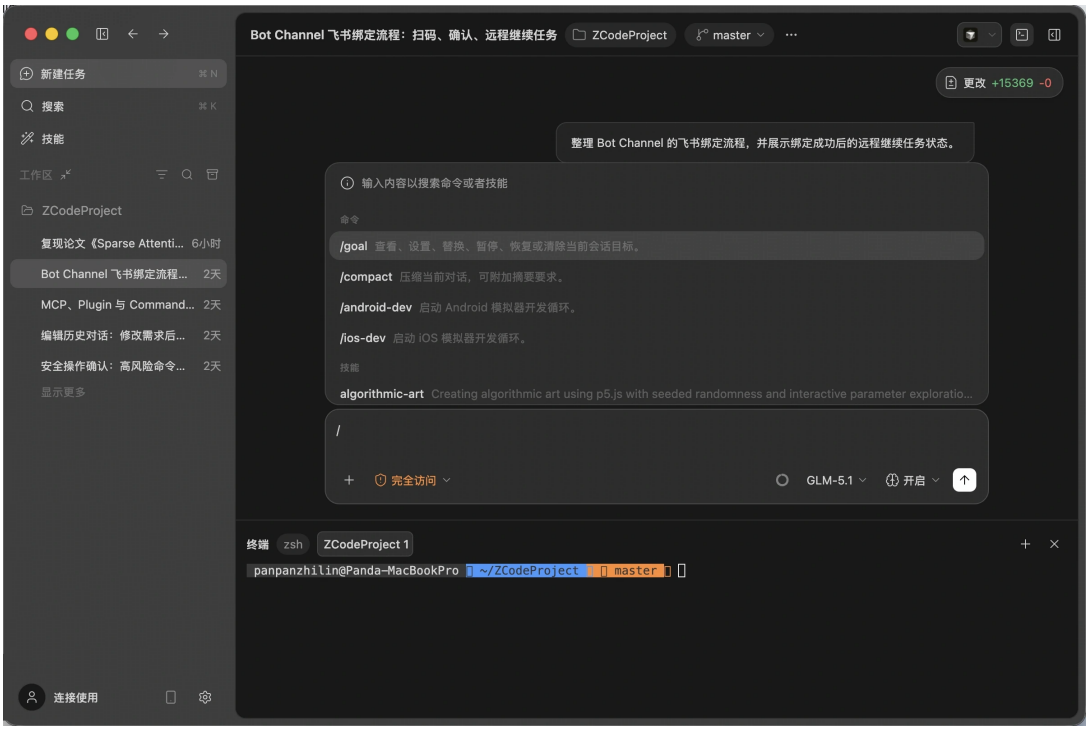
16.1 本章目标

- 掌握两个内置命令
- 创建自定义命令并理解存储位置
- 从外部 Agent 导入命令

16.2 概念

Command 用来快速调用 ZCode Agent 的内置能力，也可以保存常用提示词。把重复使用的提示词（代码审查、提交说明、发布检查、文件解释等）保存成命令后，就能通过 `/` 快速调用。

16.3 在对话中使用



1. 在输入框输入 `/`，打开命令面板。面板分为 **命令** 和 **技能** 两个分组，可继续输入关键字筛选；
2. 选择要使用的命令，或继续输入命令名筛选；
3. 如果命令需要参数，可以在命令后继续补充：路径、模块名、说明等。

> **提示**：需要调用"技能"时用 `$`，或在 `/` 面板的「技能」分组中选择。

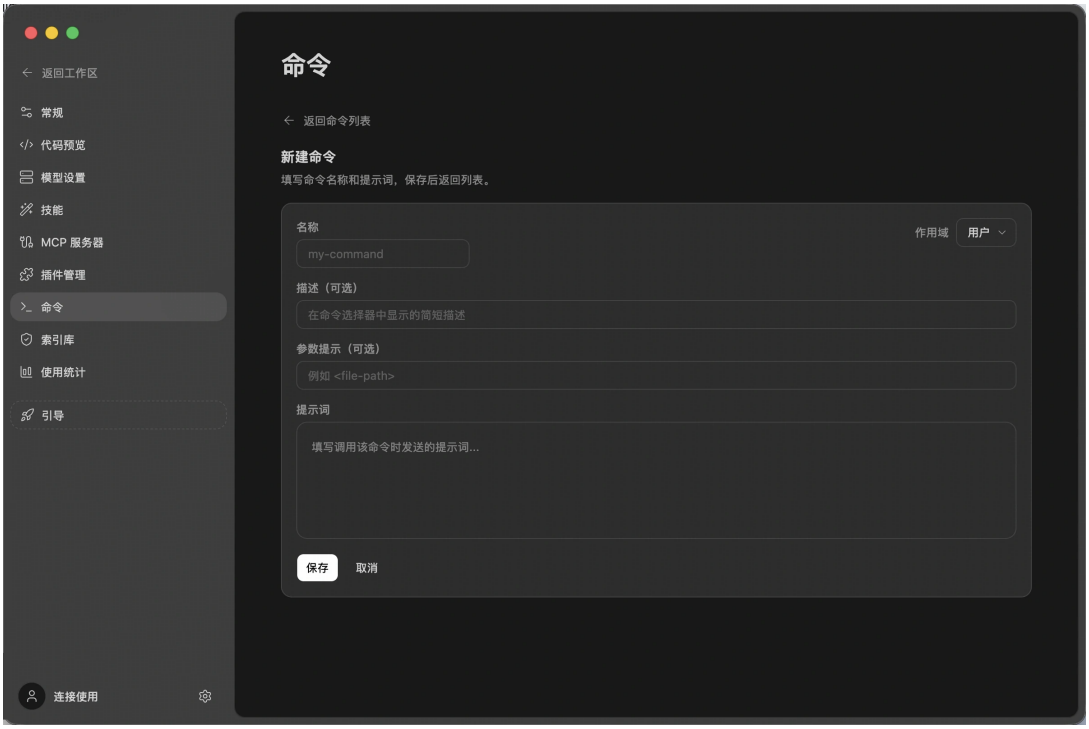
16.4 内置命令

ZCode Agent 当前**内置两个命令**：

命令	用途	适合
`/goal`	查看、设置、替换、暂停、恢复或清除当前会话目标	持续执行的长任务（详见第 5 章）
`/compact`	压缩当前对话上下文，保留关键信息	长对话继续推进时使用

16.5 自定义命令

创建入口：设置中进入 **命令** 页面，点击 **新建命令**。



字段	说明
作用域	**用户**（所有工作区可用）或 **工作区**（仅当前项目可用）
名称	命令名称，保存后可通过 `/command-name` 调用
描述	可选，显示在命令选择器中的简短说明
参数提示	可选，用来提示参数格式，例如 `<file-path>`
提示词	命令真正发送给 Agent 的提示词内容（核心）

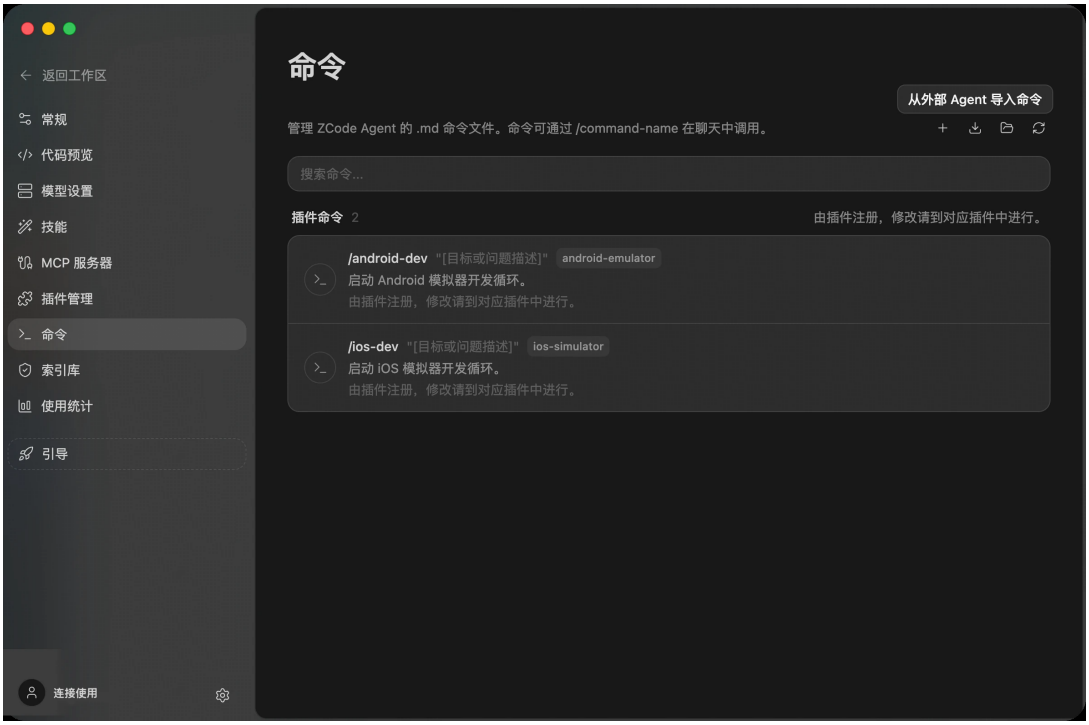
存储位置：自定义命令以 `.md` 文件形式保存——

- **用户级命令**：`~/.zcode/commands`
- **工作区级命令**：项目目录下（仅当前项目可用）

保存后在任务输入框中通过 `/command-name` 调用即可。

> **建议**：推荐通过 ZCode 设置页的 UI 新建和编辑命令。导入后的命令也可以继续修改名称、描述、参数提示或提示词内容。

16.6 从外部 Agent 导入命令



如果已在 Claude Code 等外部 Agent 中维护了一批命令，不需要重新创建：

1. 在设置中进入 **命令** 页面；
2. 点击右上角的 **从外部 Agent 导入命令**；
3. 选择要导入的外部 Agent 命令，确认后即可导入。

16.7 Command 与 Skill 的选择

场景	推荐
只保存一段简单提示词，或固定流程的提示词	**Command**
流程需要脚本、模板或示例文件	**Skill**

16.8 本章小结

- 内置命令只有 `/goal` 与 `/compact`；其余都是自定义的。
- 自定义命令 = 名称 + 提示词 + 可选参数提示，以 `.md` 存于 `~/.zcode/commands` 或项目目录。
- 一句话提示词用 Command，成套工作方法用 Skill。

16.9 思考与练习

1. 创建一个 `/commit-msg` 命令，让它按 Conventional Commits 规范生成提交说明，并限定作用域为工作区。
2. 在长对话中先执行 `/compact`，对比压缩前后 Agent 对早期约定的记忆情况。

第 17 章　Plugin：打包与分发能力

17.1 本章目标

- 浏览、安装、管理插件
- 搭建团队自己的插件市场
- 从零写出一个插件

17.2 插件里有什么

一个插件可以同时包含多种能力，ZCode 会根据插件目录内容自动识别组件：

组件	说明
Skill（技能）	教 Agent 如何完成特定任务的技能文件
Command（命令）	可通过 `/` 调用的快捷命令
Agent（子智能体）	随插件一起注册的子智能体
MCP 服务器	随插件注册的外部工具服务，出现在 MCP 列表的 **Plugin MCP 服务器** 分组
Hook（钩子）	在特定事件触发的自动化钩子

启用插件后，它的可运行组件（技能、命令、MCP 服务器）会注册到当前工作台；停用后一起停用。

17.3 浏览与安装

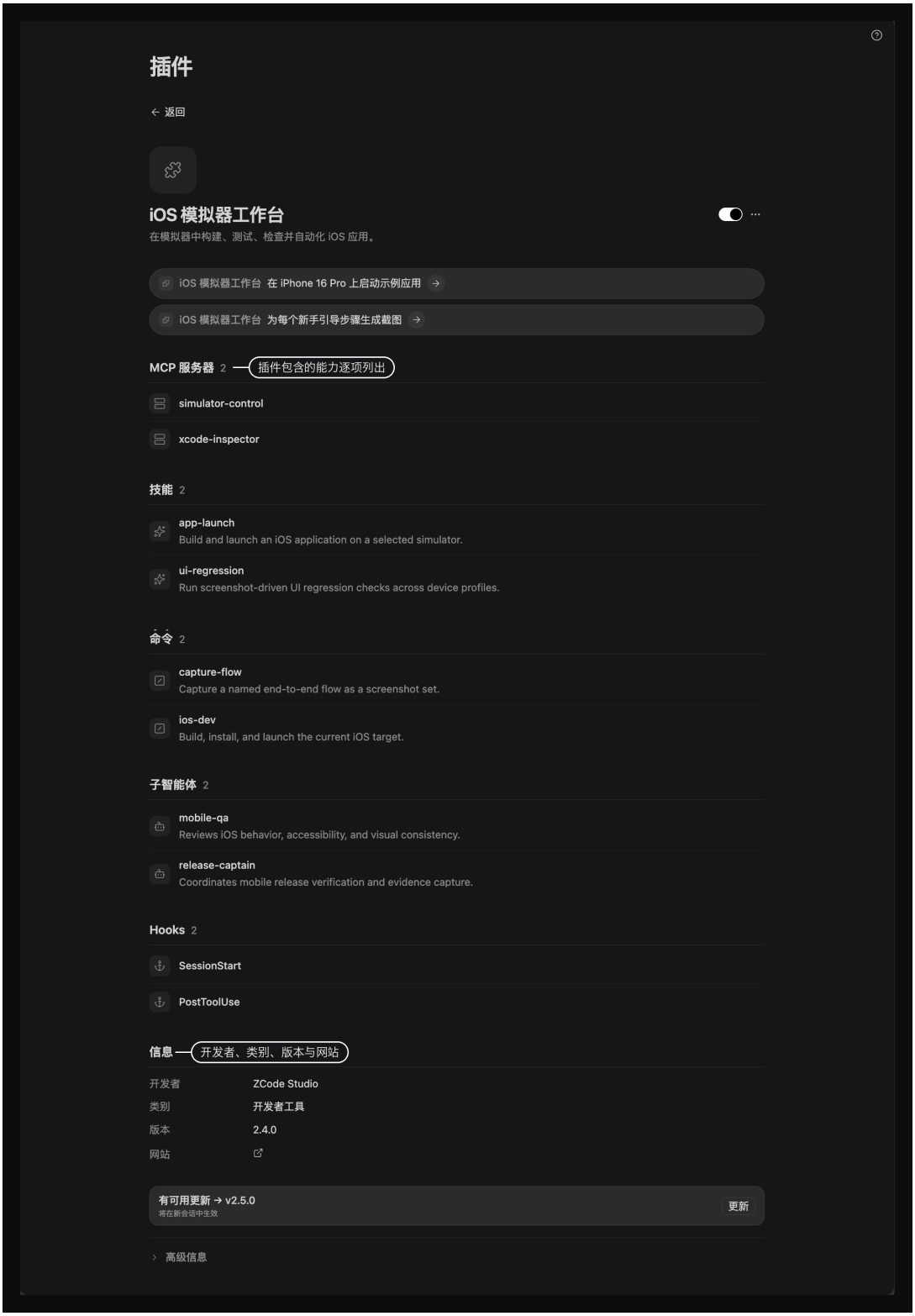


进入 **设置** → **插件** 就是插件商店。顶部搜索框可跨所有来源查找；下方分 **公开** 与 **个人** 两个分段：

- **公开**：ZCode 收录的插件目录，顶部精选推荐，其余按「开发者工具」「生产力」「实用工具」「指南」「模板」等类别分组；
- **个人**：你自己添加的插件市场。ZCode 已**预置 Claude Code 的插件市场**，不用自己添加即可浏览安装。

> **注意**：插件页需要在**打开工作区后**才可用。如果提示「打开一个工作区以管理插件」，先打开任意项目 / 工作区。公开目录来自 GitHub，网络访问不畅时目录、详情和安装都可能加载失败；列表没及时更新时点右上角 **刷新**。

点击卡片上的 **安装** 即可，状态依次变为「安装中…」和「已安装」，**新装插件默认启用**。点击任意插件可打开详情页，查看它实际包含的技能、命令、子智能体、MCP 服务器和 Hooks，以及开发者、类别、版本、网站等信息。



17.4 添加自己的插件市场



点右上角 **创建** → **添加插件市场**，指定一个来源：

- GitHub 仓库（例如 `owner/repo` 或其链接）；
- Git URL；
- 本地的市场清单文件或目录（可直接拖入文件夹，或点 **选择目录**）。

添加前 ZCode 会先校验市场。成功后它发布的插件以市场名称分组出现在 **个人** 分段。

点搜索框上方的**齿轮图标**可打开 **市场源** 面板，查看每个市场收录了多少插件、上次更新时间，并单独刷新或移除该市场。ZCode 自带的官方目录只能刷新，不会被移除。

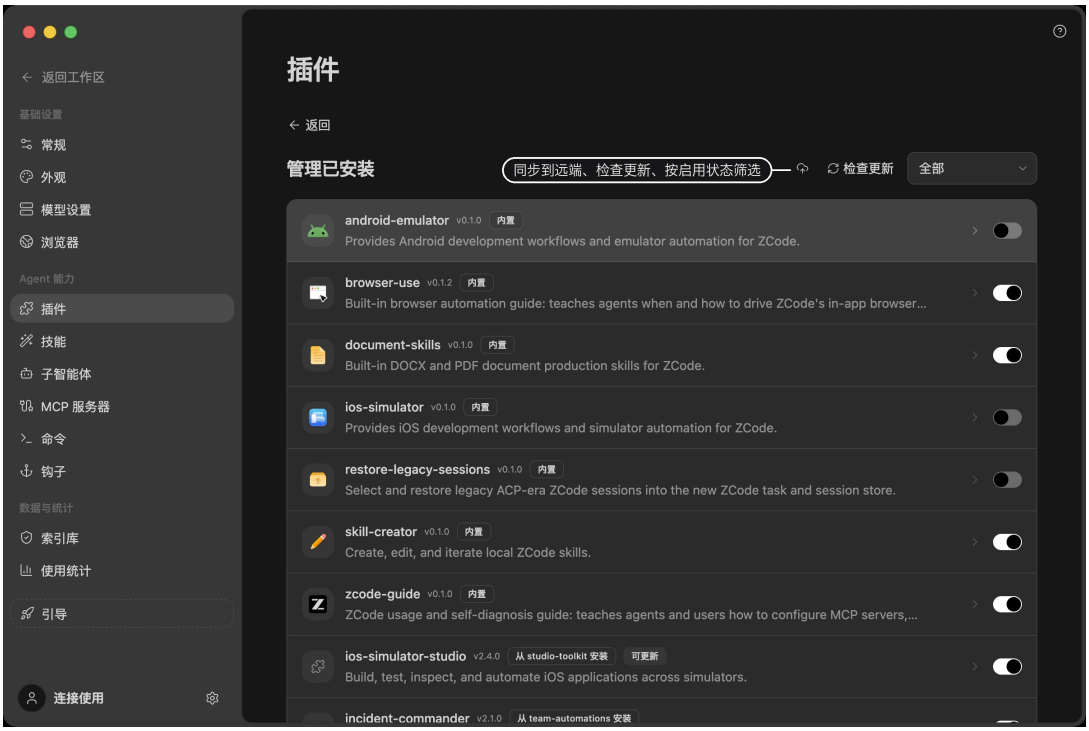


17.5 内置插件

插件	能力	默认状态
document-skills	生成 DOCX、PDF 等文档的内置技能	默认启用
skill-creator	创建和编辑你自己的本地技能	默认启用
android-emulator	Android 开发工作流与模拟器自动化	默认关闭
ios-simulator	iOS 开发工作流与模拟器自动化（macOS）	默认关闭
restore-legacy-sessions	迁移并恢复旧版本的会话	默认关闭

其中 `android-emulator` 和 `ios-simulator` 启用后，ZCode Agent 能直接驱动模拟器，把构建运行、安装启动、界面验证放进同一条对话流里。

17.6 管理插件



装过插件后，搜索框下方出现 **已安装** 图标带，点图标进入插件详情；点右侧齿轮图标进入 **管理已安装** 页面。

可以完成：

- 按启用 / 停用状态过滤；
- 查看来源标签（如 `内置` 或某市场名称）；
- 通过右侧开关启用或停用；
- 点 **检查更新**：注意版本对比口径——「**最新版本**」取自市场 `marketplace.json` 里该插件条目声明的 `version`，「**已安装版本**」取自插件自身的 `plugin.json`。**如果市场条目发版时忘了更新 `version`，即使插件代码已更新也不会提示可更新**；
- 点击插件条目查看组件并可卸载。内置官方插件同样可以卸载。

> **注意**：
>
> - 启用或停用插件后，ZCode 会自动刷新受影响的技能和会话让改动生效；停用后全部组件立即从会话中移除，再次启用即可恢复；
> - **停用 vs 卸载**：停用只是暂时不生效，随时可以开回来；卸载则从列表移除。内置插件卸载时会记录一条屏蔽标记，**应用升级后也不会把它装回来**——想重新使用需在市场里重新安装；
> - 远程工作区里本机插件默认不会跟过去，可用 **同步 Plugin** 搬过去（第 12 章）。

17.7 配置插件

点开插件详情，展开底部 **高级信息**，在 **配置** 区填写可填项——必填项标注「必填」，填完点 **保存配置**。

> **注意**：标记为敏感的项（例如 API 密钥）会提示「该值需要安全存储接入后才能配置」，**当前暂不支持在界面里直接填写**。遇到这类插件，按其说明在系统层面准备好密钥即可。

17.8 装好的插件怎么用

能力	在哪里用
技能	合适时机自动触发；或输入 `/` 从「技能」分组手动选用。在 **设置** → **技能** 的「Plugin 技能」分组统一查看
命令	输入 `/` 从「命令」分组选用，可输入关键词搜索
子智能体	会话中可被自动调度；在 **设置** → **子智能体** 的「插件子智能体」分组查看（只读）
MCP 服务器	在 **设置** → **MCP** 中显示为 **Plugin MCP 服务器**，随插件启停自动加载

17.9 开发自己的插件

只需会写 JSON 和 Markdown，不用改 ZCode 本体。

17.9.1 目录结构

```
my-plugin/
├── .zcode-plugin/
│   └── plugin.json     清单（唯一必需）
├── commands/           斜杠命令，每个一个 .md
├── skills/             技能，每个子目录含 SKILL.md
├── agents/             子智能体 .md
├── hooks/hooks.json    钩子
└── .mcp.json           MCP 服务声明
```

清单位置按优先级查找：`.zcode-plugin/plugin.json`（推荐）→ `.claude-plugin/plugin.json`（兼容 Claude Code）。

17.9.2 五种组件的写法

组件	格式与位置
命令	`commands/*.md`，YAML frontmatter + 正文；正文用 `$ARGUMENTS` 接收参数
技能	`skills/<名>/SKILL.md`，frontmatter 写清 `name` / `description`
子智能体	`agents/*.md`，frontmatter 必填 `name` / `description`，正文即 system prompt
Hooks	`hooks/hooks.json`，随插件启停（详见第 19 章）
MCP 服务	根目录 `.mcp.json` 或清单 `mcpServers`，键名自动加命名空间避免冲突

最小清单只需要一个 `name`：

```
{
  "name": "hello-world",
  "version": "0.1.0",
  "description": "我的第一个插件"
}
```

17.9.3 `plugin.json` 字段速查

字段	必填	含义
`name`	✅	插件名，须匹配 `^[a-z0-9][a-z0-9._-]{0,127}$`
`version`		版本号，缺省 `0.0.0`，建议语义化版本
`description`		一句话描述
`author`		作者，字符串或对象 `{ name, email, url }`
`homepage` / `repository`		主页与仓库地址
`license`		许可证，如 `MIT`
`keywords`		关键词数组
`commands` / `skills` / `hooks` / `mcpServers` / `agents`		各类组件声明，可写目录路径字符串、路径数组或内联对象
`dependencies`		依赖的其他插件，写 `name@market` 或同市场内裸 `name`

字段	必填	含义
`userConfig`		用户可配置项

> **注意**：清单里写了 `channels` / `lspServers` / `outputStyles` / `settings` 这几个字段时，当前运行时**仅登记、不执行**，会给出诊断提示，不影响其它组件加载。

`userConfig` 每一项都会出现在插件详情弹窗的 **配置** 区：

字段	含义
`type`	`string` / `number` / `boolean` / `directory` / `file`
`title`	界面显示的标题
`description`	配置项说明
`default`	默认值
`required`	布尔；是否必填，界面标「必填」
`sensitive`	布尔；敏感值，界面打码且暂不支持在界面直接填写

敏感配置（`sensitive`）的值可以在 MCP 声明里用 `${user_config.键}` 引用。

17.9.4 `marketplace.json` 字段速查

顶层字段

字段	必填	含义
`name`	✅	市场名，命名规则同插件名
`description`		市场描述
`plugins`	✅	插件条目数组
`pluginRoot`		解析各条目 `source` 时的基准目录，相对市场根目录
`allowCrossMarketplaceDependenciesOn`		允许跨市场依赖的市场名数组

`plugins[]` 每个条目

字段	必填	含义
`name`	✅	插件名
`source`	✅	插件代码在哪（见下）

字段	必填	含义
`description` / `version`		展示用描述与版本
`category` / `tags`		分类（字符串）与标签（字符串数组）
`dependencies`		依赖的其他插件
`strict`		布尔；对该条目做更严格的校验

`source` 的几种写法

写法	含义
`"./plugins/hello"`	最常用。相对市场根目录的子目录
`{ "source": "directory", "path": "/abs/path" }`	本地绝对路径目录
`{ "source": "github", "repo": "owner/repo", "path": "subdir", "ref": "main" }`	从 GitHub 仓库取，可指定子目录与分支
`{ "source": "git", "url": "https://...git", "path": "subdir", "ref": "..." }`	从任意 Git 仓库取
`{ "source": "file", "path": "..." }`	读取一个本地清单文件
`{ "source": "url", "url": "https://.../marketplace.json" }`	指向一个 JSON 文件的 HTTP 地址，可带 `headers`
`{ "source": "npm", "package": "..." }`	从 npm 包取

17.9.5 命令 `.md` frontmatter 速查

字段	必填	含义
`description`	✅	命令描述（或正文非空即可）
`argument-hint`		参数提示，如 `"[topic]"`
`allowed-tools`		逗号分隔，限制该命令可用的工具
`model`		覆盖默认模型
`skills`		逗号分隔，自动挂载的技能
`disable-noninteractive`		布尔；是否在非交互模式下禁用

正文里 `$ARGUMENTS` 代表用户传入的全部参数，`$1` / `$2` 代表位置参数。命令名取自文件名，须匹配 `^[a-z0-9][a-z0-9_:-]{0,63}$`。

17.9.6 技能 `SKILL.md` frontmatter 速查

字段	必填	含义
`name`	✅	技能名，缺省取所在目录名
`description`	✅	触发说明，写清「什么时候用」；最长 1024 字符
`when_to_use`		补充触发时机描述
`license`		许可证
`metadata`		对象，可放 `author` / `version` 等

其余非白名单字段（如 `homepage`）会被忽略，不影响加载。

17.9.7 完整 JSON 用例

`plugin.json`（全字段）

```
{
  "name": "ios-simulator",
  "version": "1.2.0",
  "description": "iOS 模拟器开发循环：技能 + 命令 + MCP + 钩子",
  "author": { "name": "你的名字", "email": "you@example.com", "url": "https://example.com" },
  "homepage": "https://example.com/ios-simulator",
  "repository": "https://github.com/your-team/ios-simulator",
  "license": "MIT",
  "keywords": ["ios", "simulator", "mobile"],
  "commands": "commands",
  "skills": ["skills", "extra-skills"],
  "agents": "agents",
  "hooks": "hooks/hooks.json",
  "mcpServers": ".mcp.json",
  "dependencies": ["skill-creator@zcode-plugins-official"],
  "userConfig": {
    "api_key": {
      "title": "API 密钥",
      "description": "访问第三方服务用",
      "type": "string",
      "required": true,
      "sensitive": true
    },
    "default_device": { "title": "默认设备", "type": "string", "default": "iPhone 16" },
    "max_retries": { "type": "number", "default": 3 },
    "verbose": { "type": "boolean", "default": false },
    "workspace_dir": { "type": "directory" },
    "config_file": { "type": "file" }
  }
}
```

`marketplace.json`（全字段 + 所有 source 写法）

```
{
  "name": "my-market",
  "description": "团队内部插件市场",
  "pluginRoot": "plugins",
  "allowCrossMarketplaceDependenciesOn": ["zcode-plugins-official"],
  "plugins": [
    { "name": "hello-world", "source": "./hello-world", "description": "打招呼插件", "version": "0.1.0", "category": "demo", "tags": ["starter", "demo"], "strict": true },
    { "name": "from-github", "source": { "source": "github", "repo": "your-team/another", "path": "plugins/x", "ref": "main" }, "dependencies": ["hello-world", "skill-creator@zcode-plugins-official"] },
    { "name": "from-git", "source": { "source": "git", "url": "https://git.example.com/x.git", "path": "sub", "ref": "v1.0" } },
    { "name": "from-dir", "source": { "source": "directory", "path": "/abs/path/to/plugin" } },
    { "name": "from-url", "source": { "source": "url", "url": "https://example.com/plugin-manifest.json", "headers": { "Authorization": "Bearer xxx" } } },
    { "name": "from-npm", "source": { "source": "npm", "package": "@scope/plugin" } }
  ]
}
```

设了 `pluginRoot` 后，相对 `source`（如 `"./hello-world"`）以它为基准解析，即 `plugins/hello-world`。

`.mcp.json`（stdio + http + sse）

```
{
  "mcpServers": {
    "ios-simulator": {
      "type": "stdio",
      "command": "node",
      "args": ["${CLAUDE_PLUGIN_ROOT}/dist/mcp/server.js"],
      "cwd": "${CLAUDE_PROJECT_DIR}",
      "env": {
        "IOS_SIM_ROOT": "${CLAUDE_PLUGIN_ROOT}",
        "IOS_SIM_DEVICE": "${user_config.default_device}"
      },
      "enabled": true,
      "timeoutMs": 60000
    },
    "remote-http": {
      "type": "http",
      "url": "https://mcp.example.com/api",
      "headers": { "Authorization": "Bearer ${user_config.api_key}" },
      "enabled": true,
```

```
      "timeoutMs": 30000
    },
    "remote-sse": {
      "type": "sse",
      "url": "https://mcp.example.com/sse",
      "headers": { "X-Token": "${user_config.api_key}" }
    }
  }
}
```

`type` 可省略——有 `command` 默认 `stdio`，有 `url` 默认 `http`。可用模板变量：

变量	含义
`${CLAUDE_PLUGIN_ROOT}`	插件根目录，亦可写 `${ZCODE_PLUGIN_ROOT}`
`${CLAUDE_PLUGIN_DATA}`	插件数据目录
`${CLAUDE_PROJECT_DIR}`	当前工作目录
`${user_config.键}`	引用 `userConfig` 中对应配置项的值

服务键名会自动加命名空间 `plugin:<插件名>:<服务名>` 避免冲突。

17.9.8 本地测试与团队分发

本地测试：

1. 本地建好插件目录，再写一份 `marketplace.json`，`plugins[].source` 用相对路径指向插件目录；
2. 打开 **设置** → **插件**，点右上角 **创建** → **添加插件市场**，填该目录的本地路径（本地路径需真实存在）；
3. 在 **个人** 分段找到它，点 **安装** 并启用，在会话里触发组件验证；改完代码回到 **市场源** 面板刷新该市场即可。

团队分发：把插件放进市场仓库的 `plugins/` 目录，根目录写 `marketplace.json` 列出条目，推到 GitHub。队友点 **创建** → **添加插件市场** 填仓库地址，即可一次拿到全部插件。

> **建议**：自带的官方插件是最好的范例（skill-creator 最简，ios-simulator / android-emulator 最完整）。从纯技能插件起步，跑通后再加命令、Hooks、MCP。
>
> **安全提示**：启用插件就是授予代码执行信任。已启用的第三方市场插件与官方插件一样，可以执行本地进程并读取继承的 Agent 环境变量。**启用前请审查来源、`hooks/hooks.json` 和脚本**，不信任时停用或卸载插件。

17.10 本章小结

- 插件是 Skill / Command / Agent / MCP / Hook 的打包单元，也是唯一的分发渠道（无独立技能市场）。
- 清单查找优先级：`.zcode-plugin/plugin.json` → `.claude-plugin/plugin.json`；`hooks/hooks.json` 是标准位置，会自动发现。

- 自建市场发版务必同步修改 `marketplace.json` 里的 `version`，否则客户端不会提示更新。

17.11 思考与练习

1. 从"纯技能插件"起步，做一个团队 `onboarding` 插件，内含 3 个技能，本地安装验证。
2. 故意只改 `plugin.json` 的 `version` 而不改 `marketplace.json`，观察客户端是否提示更新，并总结发版清单。

第 18 章　MCP：接入外部工具与数据

18.1 本章目标

- 新建并管理 MCP 服务器
- 理解配置文件的路径、作用域与读取优先级
- 掌握工作区 MCP 自动连接的安全含义

18.2 概念



MCP（Model Context Protocol）可以把文件系统、浏览器、记忆、数据库等外部能力接入 Agent。ZCode 统一管理 **ZCode Agent** 使用的 MCP 服务器配置。

配好常用 MCP 后，Agent 在长任务里可以更稳定地使用项目文件、浏览器自动化、记忆等工具。

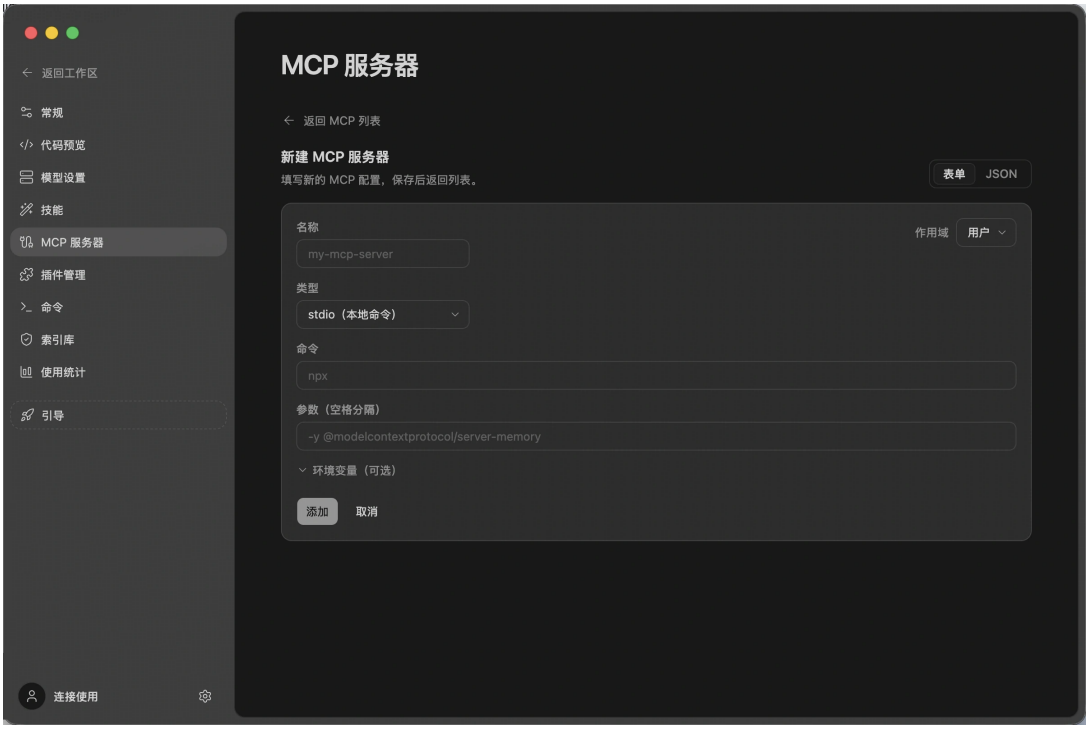
18.3 列表分组

分组	说明
已配置 MCP 服务器	手动添加的 MCP 服务，可直接编辑、删除或启用 / 停用
Plugin MCP 服务器	随插件一起安装的 MCP 服务，由对应插件统一管理

列表中每个 MCP 展示名称、来源、启动方式和命令。



18.4 新建 MCP 服务器



进入 **设置** → **MCP 服务器**，点击右上角 **新建 MCP 服务器**。**表单**模式适合快速填写常见 stdio 服务：

1. 选择**作用域**：**用户**（所有工作区可用）或 **工作区**（仅当前项目可用）；
2. 填写名称，例如 `memory`；
3. 类型选择 `stdio（本地命令）`，也支持 SSE 和 HTTP 类型的远程服务；
4. 填写命令，例如 `npx`，以及参数，例如 `-y @modelcontextprotocol/server-memory`；
5. 如服务需要密钥或路径，展开 **环境变量（可选）** 填写；
6. 点击 **添加** 后回到列表，确认开关处于启用状态。

其他两种录入方式：

- **远程服务**：类型选择 `HTTP` 或 `SSE`，填写服务地址；需要鉴权的展开 **请求头（可选）** 补充 Authorization 等信息；
- **完整配置**：切换到该模式直接粘贴 JSON，支持 `{"server-name": {...}}` 和 `{"mcpServers": {...}}` 两种常见结构。



需要 OAuth 授权的服务

有些远程 MCP 服务（例如 Figma 的官方 MCP）不接受固定密钥，而是要求你在浏览器里授权一次。为这类服务开启 OAuth 后：**设置** → **MCP** 里对应服务行会出现 **打开授权** 按钮，点击后用系统浏览器打开授权页面，你确认完，ZCode 自动重连并加载它的工具。ZCode 不会擅自弹出浏览器窗口。

> **注意**：
>
> - 这只适用于 **HTTP 和 SSE** 类型的远程服务，本地服务仍通过环境变量传凭据；
> - 如果新建会话时授权一直没完成，ZCode 只会把这一个服务标记为失败，不会拖住整个会话；
> - 换了服务地址或授权范围后需要重新授权一次。

18.5 从外部 Agent 导入



如果已在 Claude Code、Codex CLI、OpenCode 等工具中配置了 MCP 服务器，在 **MCP 服务器** 页面右上角点击 **导入** 图标，ZCode 会自动扫描并集中列出。

来源	配置文件
Claude Code	`~/.claude/settings.json`
Codex CLI	`~/.codex/config.toml`
OpenCode	`~/.config/opencode/opencode.json`
通用 `.agents`	`~/.agents/mcp.json`

操作步骤：

1. 进入 **MCP 服务器** 页面；
2. 点击右上角 **导入** 图标；
3. 在弹窗右上角选择导入的 **作用域**（全局或当前工作区）；
4. 勾选要导入的服务器，或点击 **全选**；
5. 点击 **导入** 完成，服务器出现在列表中并默认启用。

> **注意**：导入后的服务保存在对应作用域的 `.zcode` 配置文件中，可继续编辑、启停或删除，**原外部 Agent 的配置文件不会被修改**。

18.6 配置文件与默认读取路径

设置面板只是入口之一。ZCode 按固定的目录约定读取配置文件：

作用域	文件路径	配置键
用户（所有工作区可用）	`~/.zcode/cli/config.json`	`mcp.servers`
工作区（仅当前项目可用）	`<项目根>/.zcode/config.json`	`mcp.servers`
用户 · `.agents` 兼容	`~/.agents/mcp.json`	`mcpServers`
工作区 · `.agents` 兼容	`<项目根>/.agents/mcp.json`	`mcpServers`

> **注意**：用户级原生配置在 `.zcode/cli/` 目录下，工作区级直接在 `.zcode/` 目录下，两个文件同名为 `config.json` 但**层级不同**。

ZCode 原生配置的 MCP 部分形如：

```
{
  "mcp": {
    "servers": {
      "memory": {
        "command": "npx",
        "args": ["-y", "@modelcontextprotocol/server-memory"],
        "env": {}
      }
    }
  }
}
```

`.agents/mcp.json` 使用业界通用的 `{"mcpServers": {...}}` 结构，便于和其他支持 `.agents` 目录约定的 AI 工具共享同一份配置。

18.7 读取规则与优先级

1. **工作区优先加载**：打开工作区时先读工作区作用域配置，再读用户作用域，两边的服务都会出现在列表中；
2. **同作用域内 `.zcode` 强优先**：只要 `.zcode` 配置文件里读到了任何 MCP 服务，同作用域的 `.agents/mcp.json` 就会被**整体跳过，不做合并**。`.agents` 只在 `.zcode` 没有配置任何服务时才作为兜底来源生效；
3. **启用状态记录在配置对象内**：列表里停用某个 MCP 时，ZCode 会向该服务的配置对象写入 `"enable": false`；没有这个字段即视为启用；
4. **面板写入永远落在 `.zcode`**：通过设置面板新增或编辑的服务，始终写回对应作用域的 `.zcode` 原生配置文件，`.agents` 文件不会被修改。

> **注意**：如果你在 `.agents/mcp.json` 里维护了一批服务，又在设置面板里新建了一个服务，原来 `.agents` 里的服务会因为 `.zcode` 优先规则而**整体失效**。此时把 `.agents` 里的配置合并进 `.zcode` 配置文件即可。

18.8 工作区 MCP 会自动连接（安全须知）

写在项目配置里的 MCP 服务，在会话启动时会和用户级服务一样**自动连接**，不需要每次手动授权。这让团队把 MCP 配置提交进仓库后，成员克隆下来就能直接用。

> **注意（安全）**：**打开一个项目，就等于连上了它配置里写的所有 MCP 服务**，而 MCP 服务能执行命令、读写你的文件、访问网络。**打开来路不明的仓库之前，建议先看一眼它的** `.zcode/config.json`，确认里面的 MCP 配置你都认得。
>
> 同名服务以你自己的配置为准，项目配置覆盖不了它。

18.9 推荐配置

官方文档推荐的三个智谱 MCP 服务：

服务	能力
`zai-mcp-server`	视觉理解：让 Agent 分析图片、截图和界面信息
`web-search-prime`	联网搜索：让 Agent 获取最新资料和外部信息
`web-reader`	网页读取：让 Agent 解析网页正文、结构和关键信息

> **建议**：团队内如果有统一 MCP 规范，建议用 **用户** 作用域共享通用服务，再按项目需要在 **工作区** 作用域补充专用服务，避免配置互相污染。

18.10 本章小结

- MCP 配置有"用户 / 工作区"两个作用域、"`.zcode` / `.agents`"两套文件格式，读取规则是**工作区优先 + 同作用域 `.zcode` 强优先不合并。**
- 面板写入永远落在 `.zcode`，所以"手工维护 `.agents`" 与"面板新建"混用会导致 `.agents` 整体失效。
- 工作区 MCP 自动连接是便利性也是风险点：打开陌生仓库前先看 `.zcode/config.json`。

18.11 思考与练习

1. 在用户作用域添加一个 `memory` MCP（stdio），在工作区作用域添加一个项目专用 MCP，验证两者是否同时出现在列表。
2. 构造一个"`.agents` + 面板新建"混用的场景，复现 `.agents` 整体失效，然后合并修复。

第 19 章　Hooks：事件驱动的自动化

19.1 本章目标

- 理解 Hook 的执行时机与事件模型
- 掌握输入输出契约与退出码语义
- 写出第一个插件 Hook 并验证

19.2 概念

Hook 用来在特定事件时机自动执行动作——在会话开始时注入团队约束、在工具调用前做安全检查、在模型准备结束时校验产出。

Hook 本质上是一个**本地子进程协议**：ZCode 向进程的 stdin 写入一行 JSON，进程通过退出码和 stdout JSON 返回结果。Hook 不会获得可直接调用 ZCode 模型的内部对象。

> **安全提示**：Hook 会执行本地代码。启用第三方插件前请审查其来源、`hooks/hooks.json` 和脚本。

本章怎么读：只想在设置页里配几条 Hook，看「执行顺序与事件」「配置来源」「在设置页创建和维护 Hook」即可；「快速上手」及之后是面向开发者的部分（输入输出契约、退出码、插件打包）。

19.3 执行顺序与事件

```
新 session → SessionStart
用户提交 → UserPromptSubmit → 主模型
主模型请求工具 → PreToolUse → 需要确认时 PermissionRequest → 执行工具
工具成功 → PostToolUse；工具失败 → PostToolUseFailure
主模型准备结束 → Stop → 结束，或注入反馈后继续主模型
```

事件	matcher	主要用途与效果
`SessionStart`	匹配 `source`，常见值 `startup` / `clear` / `compact`	首轮模型请求前初始化环境、注入项目约束或操作说明
`UserPromptSubmit`	不参与过滤（填了也会执行）	模型调用前补充上下文，或阻断本次用户请求；**不能改写原始 prompt**
`PreToolUse`	匹配工具名	允许、询问或拒绝工具；可完整替换工具输入，替换后重新校验 schema
`PermissionRequest`	匹配工具名	只在权限结果需要询问时触发；可允许、拒绝、更新输入或权限规则
`PostToolUse`	匹配工具名	工具成功后追加模型可见上下文；**不能替换工具输出**
`PostToolUseFailure`	匹配工具名	工具失败后追加恢复建议、诊断或重试约束
`Stop`	不参与过滤（填了也会执行）	模型准备结束时检查结果；返回 `block` 可让现有主模型循环继续，**最多连续 3 次**

19.4 配置来源

来源	适用场景	生效方式
`~/.zcode/cli/config.json`	当前用户的所有工作区	必须在该文件中设置 `hooks.enabled: true`
插件 `hooks/hooks.json`	随插件安装和分发	标准位置自动发现，随插件启停；不需要在 manifest 中重复声明同一文件
`.agents/settings.json` / `.claude/settings.json`	迁移旧配置	**只读展示，不直接执行**；需在设置页显式导入到 `.zcode`

> **注意（重要）**：**项目级 Hook 当前不会执行。** 出于安全策略，写在 `<workspace>/.zcode/config.json` 或 `<workspace>/zcode.json` 里的 hooks 配置在当前版本会被**整体忽略**（无论 `hooks.enabled` 是否为 true，日志中记为 `config_project_hooks_ignored`），设置页也已隐藏工作区作用域的配置入口。需要团队共享的 Hook，请改用**插件分发**，或让成员各自配置用户级 Hook。

执行顺序：**user Hook → 已启用插件 Hook**；同一来源内按数组顺序执行。

> **注意**：每个 session 启动时会捕获一份 Hook 配置快照。修改文件、在设置页保存或启停插件后，请**新建 session 验证**；已启动的 session 不保证热更新。

用户配置示例：

```
{
  "hooks": {
    "enabled": true,
    "timeoutMs": 60000,
    "maxOutputBytes": 32768,
    "events": {
      "PreToolUse": [
        {
          "matcher": "Write|Edit",
          "hooks": [
            {
              "type": "process",
              "command": "node",
              "args": ["scripts/check-write.mjs"],
              "enabled": true,
              "timeoutMs": 10000
            }
          ]
        }
      ]
    }
  }
}
```

19.5 快速上手：第一个插件 Hook

下面的插件在新 session 启动时给模型补充一条团队约束（依赖系统 PATH 中可用的 `node`）。

目录结构

```
context-guard/
├── .zcode-plugin/
|   └── plugin.json
└── hooks/
    ├── hooks.json
    └── context.mjs
```

`plugin.json`（`hooks/hooks.json` 是标准位置会自动加载，manifest 不必再写 hooks 字段）

```
{
  "name": "context-guard",
  "version": "0.1.0",
  "description": "在会话开始时注入团队开发约束"
}
```

`hooks/hooks.json`

```
{
  "description": "Context Guard hooks",
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup|clear|compact",
        "hooks": [
          {
            "type": "process",
            "command": "node",
            "args": ["${ZCODE_PLUGIN_ROOT}/hooks/context.mjs"],
            "timeoutMs": 5000
          }
        ]
      }
    ]
  }
}
```

`hooks/context.mjs`

```
let raw = "";
process.stdin.setEncoding("utf8");
for await (const chunk of process.stdin) raw += chunk;

const input = JSON.parse(raw);
process.stderr.write("[context-guard] " + input.hook_event_name +
"\n");
process.stdout.write(JSON.stringify({
  hookSpecificOutput: {
    hookEventName: input.hook_event_name,
    additionalContext: "本项目提交前必须运行类型检查和受影响测试。"
  }
}));
```

验证步骤

1. 在 **设置** → **插件** 中点右上角 **创建** → **添加插件市场** 加入本地来源，安装并启用插件；
2. 到 **设置** → **Hooks** 确认插件 Hook 以只读条目出现，事件、matcher、命令和来源路径正确；
3. 新建 session 并发起请求；首轮模型应能看到注入的团队约束；
4. 调试时把日志写到 **stderr**，stdout 只输出协议结果，避免诊断文字破坏 JSON。

19.6 执行器、超时与 matcher

类型	语义	建议
`process`	`command` + `args[]`，直接按 argv 执行，**不经过 shell**；只支持同步	边界清楚，跨平台更稳定，优先用于 Node、Python 或二进制脚本
`command`	把完整字符串交给系统 shell；可设置 `shell`、`async`、`timeout`	适合兼容现有 Claude Marketplace 插件；注意 Windows、macOS、Linux 的 shell 和引用差异

- `timeoutMs` 单位是毫秒；兼容字段 `timeout` 单位是秒。两者同时存在时**优先** `timeoutMs`；
- 根级默认超时 **60000 ms**，默认 stdout 上限 **32768 bytes**；
- 单条 Hook 可写 `enabled: false`；runtime 会真正跳过，不只是界面置灰；
- `command` 的 `async: true` 是 fire-and-forget：当前事件立即继续，后台 stdout **不能**阻断、改输入或注入上下文；超时、取消、完成和失败仍记录生命周期；
- `statusMessage` 当前会保存并在设置页展示，但还不是运行时的实时状态提示。

matcher 规则：

- 缺省、空字符串或 `*`：匹配全部；
- 只含字母、数字、下划线和 `|`：按精确名称列表匹配，例如 `Write|Edit`；
- 包含其他字符：按 **JavaScript 正则**处理；非法正则不执行该 matcher，并产生诊断；

- 工具事件匹配实际工具名，并兼容 `Agent` / `Task` alias；
- `SessionStart` 匹配 `source`；`UserPromptSubmit` 和 `Stop` **不使用** matcher 过滤。

19.7 stdin 输入契约

ZCode 向每个 Hook 写入「一行 JSON + 换行」。同一份输入同时保留 ZCode camelCase 字段与 Claude Code snake_case alias，旧插件可以继续读取 snake_case。

```
{
  "session_id": "session-123",
  "transcript_path": "/tmp/zcode-hook/transcript.jsonl",
  "cwd": "/workspace/demo",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "src/index.ts",
    "content": "..."
  },
  "tool_use_id": "tool-123"
}
```

事件	重点字段
`SessionStart`	`source`，可选 `agent_type`、`model`
`UserPromptSubmit`	`prompt`
`PreToolUse`	`tool_name`、`tool_input`、`tool_use_id`
`PermissionRequest`	`tool_name`、`tool_input`，有真实数据时包含 `permission_suggestions`
`PostToolUse`	完整结构化 `tool_response`，以及工具名、输入和调用 ID
`PostToolUseFailure`	字符串 `error`、`is_interrupt`，以及工具字段
`Stop`	`stop_hook_active`、`last_assistant_message`

> **注意**：`transcript_path` 指向本次 Hook 可读的临时 JSONL 文件。ZCode 会在 Hook 完成后清理临时目录，不要把它当作长期存储；插件持久化数据请写入 `ZCODE_PLUGIN_DATA`。

19.8 stdout、退出码与常用返回值

- stdout 为空表示成功且无附加效果；
- 非 JSON stdout 只作为诊断，**不进入模型上下文**；
- 只有去除前导空白后以 `{` 开头的合法 JSON 才按协议解析；
- 未知字段会忽略；已知字段类型错误或事件名不符会让当前 Hook 可恢复失败，不影响后续 Hook。

注入上下文（推荐使用 `hookSpecificOutput`，事件归属最清楚；也兼容顶层 `additionalContext` / `additional_context`）：

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "只修改与当前任务相关的文件。"
  }
}
```

PreToolUse：修改或拒绝工具调用

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "已重定向到允许目录",
    "updatedInput": {
      "file_path": "generated/index.ts",
      "content": "..."
    },
    "additionalContext": "文件已被重定向到 generated 目录。"
  }
}
```

> **注意**：`updatedInput` 是**完整替代对象**，不是局部 patch；ZCode 会用工具 schema 重新校验。拒绝时返回 `permissionDecision: "deny"` 和 `permissionDecisionReason`。多个 Hook 聚合时，**deny 优先于 ask，ask 优先于 allow**。

PermissionRequest：自动允许或拒绝权限询问

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "deny",
      "message": "生产目录只能在发布流程中修改"
    }
  }
}
```

允许时把 `behavior` 改为 `allow`，并可在 `decision` 中返回 `updatedInput`、`updatedPermissions`；历史字段 `permissionUpdates` 也兼容。

> **注意**：显式 deny 规则、Plan 模式写入禁令和工具硬限制**不能被 Hook allow 绕过**。

UserPromptSubmit：阻断本次模型请求

```
{
  "continue": false,
  "reason": "请先提供工单号",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "该请求被团队策略阻断。"
  }
}
```

Stop：让主模型继续一轮

```
{
  "decision": "block",
  "reason": "还没有给出测试命令和结果，请补齐后再结束。"
}
```

> **注意**：`decision: "block"` 需要带 `reason` 或 `additionalContext` 才会续跑。为兼容旧 ZCode 配置，也接受 `continue: true` 且带 `additionalContext`。连续续跑达到 **3 次**后会强制结束，防止无限循环。

退出码

退出码	含义
0	成功，解析 stdout
2	阻断快捷方式；在可阻断事件中产生 block / deny，在 Stop 中产生继续一轮的反馈
其他非零	当前 Hook 可恢复失败，记录诊断，turn 不会因此整体崩溃

19.9 插件发现、变量与安全边界

- manifest 查找优先级：`.zcode-plugin/plugin.json` → `.claude-plugin/plugin.json`；
- 标准 `hooks/hooks.json` 自动加载；manifest 的 `hooks` 还支持相对 JSON 路径、inline 对象或二者数组。**不要让 manifest 再指向同一个标准文件**，否则会记录重复诊断并跳过重复项；
- 插件进程可读取 `ZCODE_PLUGIN_ROOT`、`ZCODE_PLUGIN_DATA`、`ZCODE_PLUGIN_ID`、`ZCODE_PLUGIN_NAME`；兼容变量 `CLAUDE_PLUGIN_ROOT`、`CLAUDE_PLUGIN_DATA` 也会注入；
- 命令、参数里的插件路径变量会在执行前替换。长期数据写入 `ZCODE_PLUGIN_DATA`，不要写回安装目录；
- 插件 Hook 在设置页**只读**，启停跟随插件本身。

19.10 在设置页创建和维护 Hook

1. 打开 **设置** → **Hooks**；
2. 选择 user 或 workspace scope，新增事件、执行类型、matcher、命令 / 参数、超时、async 和状态文案；
3. 已有 `.zcode` Hook 支持查看、编辑、删除和单条启停。编辑时不能直接改变 scope；需要删除后在目标 scope 重建。

Hook 不生效时的检查顺序：

1. Hook 是否写在了**项目级配置**里（当前版本不执行）；
2. 配置来源里 `hooks.enabled` 是否为 `true`；
3. 插件是否处于启用状态；
4. **新建一个 session**——Hook 配置在 session 启动时形成快照，修改配置或启停插件不会影响已启动的 session。

仍有问题时查日志：`~/.zcode/cli/log/zcode-<日期>.jsonl` 中会记录 Hook 的执行失败（`hook.run.failed`，含来源和耗时）与被忽略的项目级配置（`config_project_hooks_ignored`）。

19.11 本章小结

- Hook 是"一行 JSON 进、退出码 + JSON 出"的子进程协议，七个事件覆盖会话全生命周期。
- **项目级 Hook 当前不执行**，团队共享要改用插件分发。
- 配置快照在 session 启动时捕获，任何改动都要新建 session 验证。
- 幂等与安全：deny 优先于 ask 优先于 allow；Stop 的 block 最多续跑 3 次。

19.12 思考与练习

1. 写一个 `PreToolUse` Hook：把对 `config/prod/` 的写入自动 deny，并给出理由。
2. 写一个 `Stop` Hook：若本轮没有输出测试命令与结果，就返回 block 让 Agent 补齐。

第四部分　工程实践篇

前三部分讲的是"能力"，本部分讲的是"治理"：如何把风险管住、把成本看清、把问题反馈到正确的位置。

第 20 章　安全操作确认与权限治理

20.1 本章目标

- 掌握四档执行模式与四类决策选项
- 理解权限请求的阻塞语义
- 建立团队的权限使用规范

20.2 核心机制

ZCode 把 ZCode Agent 的权限控制放在任务界面里。执行模式选择器位于输入框附近，用户可以根据任务风险决定 Agent 的执行方式。

核心原则是：**在允许执行前，先看清 Agent 准备做什么。**对于命令、文件改动、网络访问、脚本执行等操作，ZCode 会在任务里展示对应内容；如果进入高权限或全自动模式，工具栏会持续显示风险提示。

20.3 四档执行模式



模式	说明	适合场景
变更前确认	每次修改文件或执行命令前都先确认（默认档位）	关键代码、生产配置等高风险改动
自动编辑	文件编辑自动执行，命令等操作仍需确认	常规迭代开发
计划模式	先制定计划，确认后再开始实施	重构、迁移、长任务

模式	说明	适合场景
完全访问	尽量自动执行，减少确认	已确认上下文可信、希望连续执行的任务

聊天输入框聚焦时按 `Shift + Tab` 可快速循环切换。

20.4 权限请求的工作流程

1. **触发确认**：Agent 发起需要授权的操作时，当前任务会暂停，输入区也进入阻塞状态，避免继续追加新的执行要求；
2. **弹出请求**：界面展示 Agent 计划执行的具体命令、文件改动或工具调用内容；
3. **用户决策**：确认后 Agent 才继续执行；拒绝后会中止本次操作或回到可调整状态；
4. **任务同步**：权限请求属于当前任务。切换任务后再回来，确认状态仍会保留；侧边栏也会显示等待确认状态。

20.5 四类决策选项

选项	说明	建议场景
允许	仅授权 Agent 执行当前这一次操作	临时或不确定的单次任务
始终允许	授权当前及后续同类型操作，无需再次询问	操作安全可信且需要高频执行，例如常规编译
拒绝	禁止 Agent 执行当前操作	发现指令、路径或风险不符合预期
始终拒绝	禁止当前及后续同类型操作	明确不希望 Agent 执行的某类操作

部分场景下还会出现 **允许本会话**、**始终允许本项目** 等更细粒度的选项。

20.6 提问超时自动继续

除了权限确认，Agent 有时会主动问你一个选择题（比如"用方案 A 还是方案 B"）。这类提问默认带 **5 分钟倒计时**，快到时间时面板上会显示剩余时长。到点还没人回答，Agent 会按自己的判断挑一个方向继续推进，这一步在历史记录里标记为「未回答，已自动继续」。

> **注意**：只要你有任何动作——鼠标移到问答面板上、开始选择、点击秒数、或在侧栏点「停止计时」——倒计时就会**永久停止**，安心思考即可。
>
> 不希望它自动继续，可以在 **设置** → **常规** → **提问自动继续** 关掉，之后所有提问都会一直等你回答。**这个开关不可追溯**：关闭期间产生的提问，即使之后重新打开也不会恢复计时。
>
> 倒计时只作用于这类普通提问，**权限请求和计划审批本来就会一直等待**，不受影响。

20.7 常见需要确认的场景

场景	示例
运行第三方脚本	Agent 尝试运行项目中的 Python、Shell 或 Node.js 脚本
网络请求	Agent 需要通过 `curl` 或其它方式访问外部 API
文件改动	Agent 创建、编辑、删除或重命名文件。ZCode 会在任务结果中展示文件改动摘要，并提供打开和撤销入口
系统级命令	执行可能修改系统配置、安装依赖或删除文件的命令

20.8 团队权限规范建议

1. 在允许执行前，仔细阅读即将执行的命令、路径和文件名；
2. **不确定时优先用「允许」，不要直接用「始终允许」**；
3. "始终允许"和完全访问模式都会减少后续确认，请只在确认安全可信时使用；
4. 大改动优先使用"计划模式"，先确认方案，再开始实施；
5. 涉及生产配置、数据库、发布流程时，把约束写进 `AGENTS.md`，并用 Hook（第 19 章）在工具调用前做机器校验；
6. 打开陌生仓库前先检查 `.zcode/config.json` 里的 MCP 配置（第 18 章）。

20.9 本章小结

- 四档模式管"自主程度"，四类决策管"单次授权范围"。
- 权限请求会阻塞输入区并随任务保留状态；普通提问有 5 分钟自动继续倒计时。
- 团队规范的核心：默认最小授权，机器校验用 Hook，人工把关留给高风险场景。

20.10 思考与练习

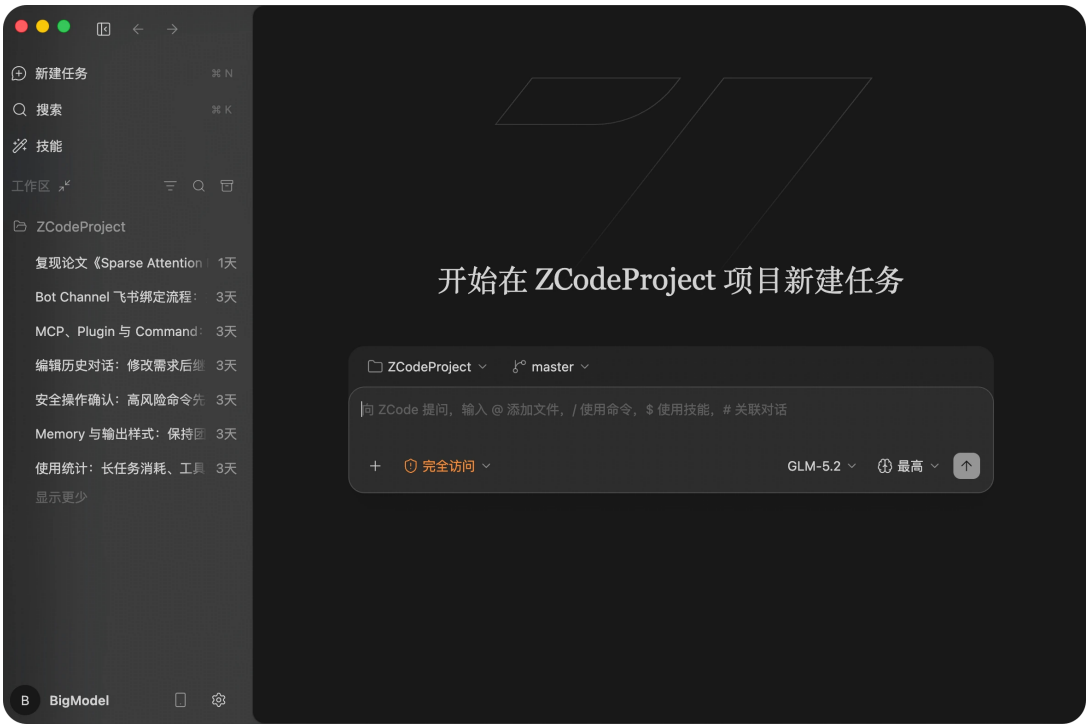
1. 为你的项目写一份"权限使用规范"，明确哪些场景必须用计划模式、哪些可以用完全访问。
2. 在设置里关掉"提问自动继续"，观察长任务中提问等待对你的工作节奏有何影响。

第 21 章　智能体开发环境工具集

21.1 本章目标

- 熟练使用命令中心、终端与后台命令
- 掌握预览面板对各类内容的呈现方式
- 会用"页面元素选成上下文"

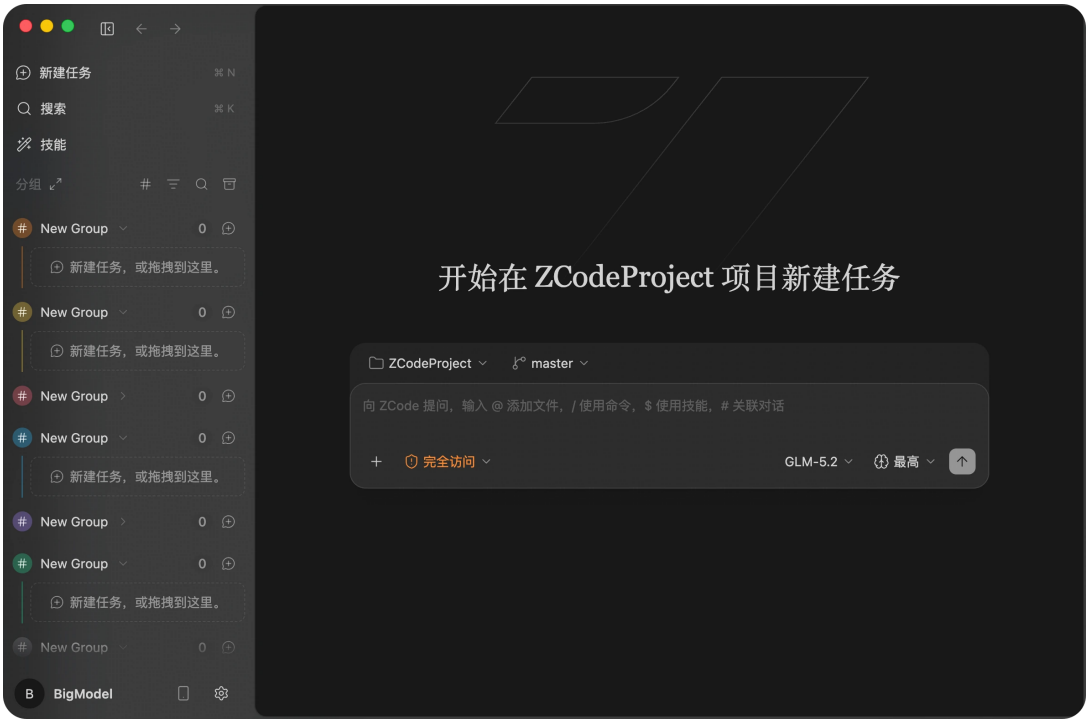
21.2 工作区与任务列表



左侧边栏跟随当前工作区展示任务列表。顶部是 **新建任务**、**搜索**、**技能** 三个入口，下面按工作区归类列出每个对话任务。

每一行显示：任务标题、相对时间（如「1 天」）、运行中 / 未读 / 失败等状态点；任务产生改动后还会显示该任务的 `+/-` **变更行数**，方便快速判断 Agent 在哪些任务里改了多少内容。





21.3 归档任务



不再活跃的任务可以归档。打开归档视图后看到所有已归档任务的扁平列表，每行显示标题、相对时间、变更统计和所属工作区，并提供 **恢复** 和 **删除** 操作。

21.4 命令中心



按 `Command + K` / `Ctrl + K`（或点击顶部搜索入口）打开命令中心，在一个面板里快速触达常用操作和导航。它同时覆盖**命令、对话和文件**三类搜索：

新建对话、打开文件夹、搜索文件、设置、上一个 / 下一个对话、查找、返回 / 前进、切换主题、MCP 服务器、个性化、切换终端等都能直接执行。

21.5 终端与后台执行

右上角的终端按钮或 `Command + J` / `Ctrl + J` 打开命令行面板。可以在同一工作区里运行构建、测试、启动服务、查看日志，再把结果继续交给 Agent 分析。命令行面板也集成在预览面板中，可以边看预览边执行命令。

后台执行长命令：启动开发服务器、跑一整套测试这类耗时命令，Agent 可以放到后台执行——命令继续跑，当前回合不必等它结束，你可以接着提问或让 Agent 做别的事，需要时再回来查看输出。子智能体同样支持后台执行。

21.6 增强的 Find 和 Grep

在大仓库里搜索容易慢，而且结果取决于你机器上装的是哪个版本的 `find` 和 `grep`。ZCode 自带了更快的替代品，Agent 默认就用它们，搜得更快，各台机器上的表现也一致。

> **注意**：这项设置在 **设置** → **常规** → **增强 Find 和 Grep**，默认开启。改完要**新建会话才生效**；Windows 上只替换 `grep`，`find` 保持系统原样；**远程工作区（WSL / SSH / Docker）暂不适用。**

21.7 文件与内容预览



点击文件树里的文件会在右侧预览面板打开。除代码外，几类常见格式也能直接看：

内容	能力
PDF	翻页、跳页和缩放（`Command` / `Ctrl` + 滚轮可按指针位置连续缩放），文字能选中复制。**只读**，不提供编辑和批注
HTML 文件	文件树里右键选 **用内置浏览器打开**，在浏览器面板新开标签渲染（该菜单项只在本地工作区出现）
Mermaid 图表	回复里的 `mermaid` 代码块直接渲染成图，点右上角放大按钮或双击打开大图，支持缩放、拖拽平移，`Esc` 关闭。消息还在输出时先显示源码，写完才切成图
图片	Markdown 图片可点开预览，连续多张自动排成图集，用左右箭头或方向键翻看，支持缩放、拖拽和下载当前这张
表格	Markdown 表格自带工具栏：复制成 Markdown、下载成 CSV、打开更大的预览窗口（滚动时表头吸顶）。很宽的表格还能展开滚动区域横向占用更多空间，再点一次收回

21.8 把页面元素选成上下文



打开浏览器标签、输入页面地址后，点击工具栏里的**元素选择按钮**进入点选模式。点击页面里的按钮、卡片、标题或其他元素，ZCode 会采集页面标题、URL、元素文本、选择器、位置和安全的 HTML 摘要，并把它加入当前聊天。

这样你可以直接对 Agent 说「修改刚才选中的卡片」或「优化这个按钮文案」，不用描述它在页面的哪个位置。



21.9 三类远程接入的分工

类别	解决什么
远程开发（第 12 章）	决定代码和命令在哪台机器上跑
Remote Control（第 13 章）	人不在电脑前时的临时操作入口
Bot Channel（第 13 章）	人不在电脑前时的长期回访入口

后两者控制的都是桌面端已有的会话。

21.10 本章小结

- 任务行的 `+/-` 变更行数是判断"Agent 在哪个任务里动了手"的最快指标。
- 终端 + 后台命令 + 预览面板构成"边看边跑边问"的闭环。
- 元素选择把"页面上的这个东西"变成可直接引用的上下文。

21.11 思考与练习

1. 用元素选择选中页面上一个按钮，让 Agent 优化它的文案，并观察它采集了哪些信息。
2. 让 Agent 后台启动开发服务器，同时继续问它一个无关问题，体验"当前回合不必等待"。

第 22 章　使用统计与额度管理

22.1 本章目标

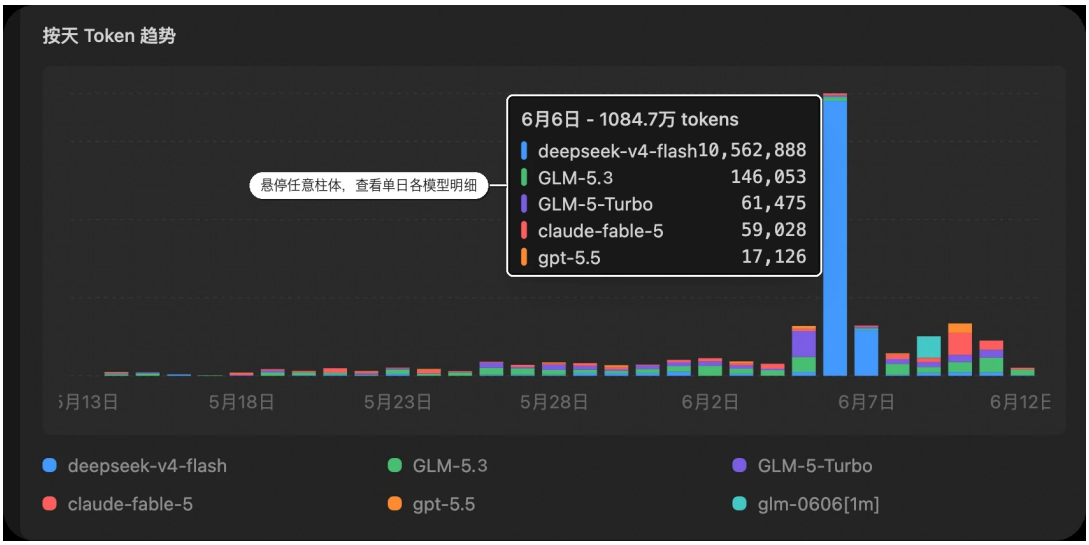
- 读懂应用用量与编程套餐两类统计
- 掌握额度重置卡的获取与使用规则
- 建立成本观测习惯

22.2 两类统计

类别	数据来源	用途
应用用量	本地 ZCode 会话记录	回看自己最近完成了多少任务
编程套餐	Z.ai / BigModel Coding Plan 远端数据	查看套餐额度、模型消耗和工具调用



22.3 应用用量

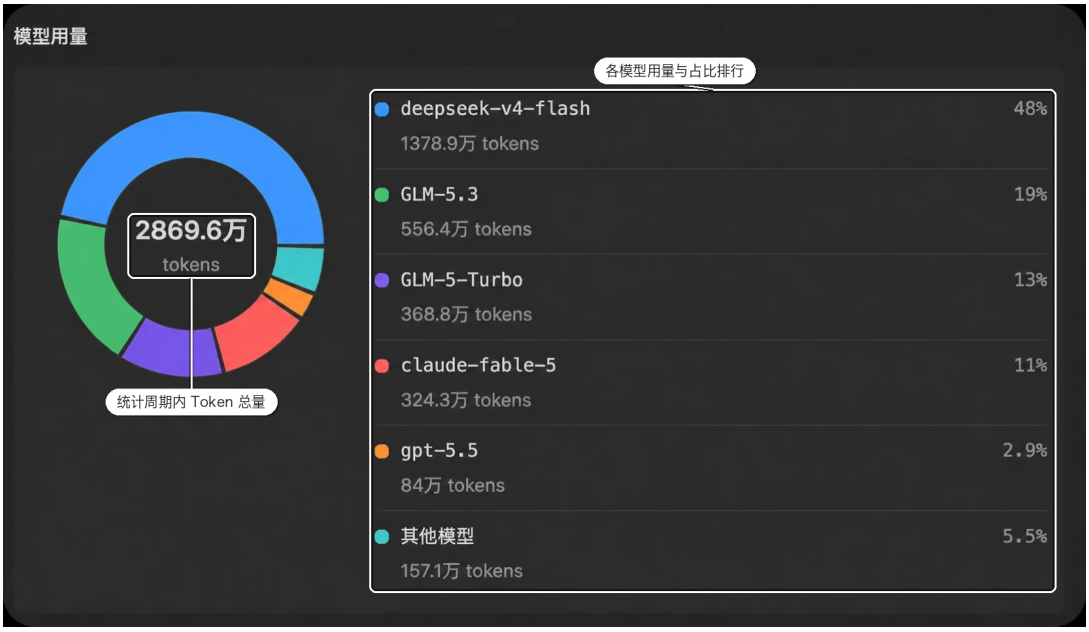


可以看到：

- Token 用量、会话数、消息数和活跃天数；
- 当前连续使用天数、最长连续使用天数和峰值时段；
- 按天 Token 趋势、活跃热力图和模型用量排行。

按天 Token 趋势图支持悬停查看每日各模型的用量明细；下方的模型用量环图汇总各模型的 Token 占比，方便确认主力模型的实际消耗分布。

右上角可以切换 **全部时间**、**最近 30 天**、**最近 7 天**，用于快速对比近期使用情况。



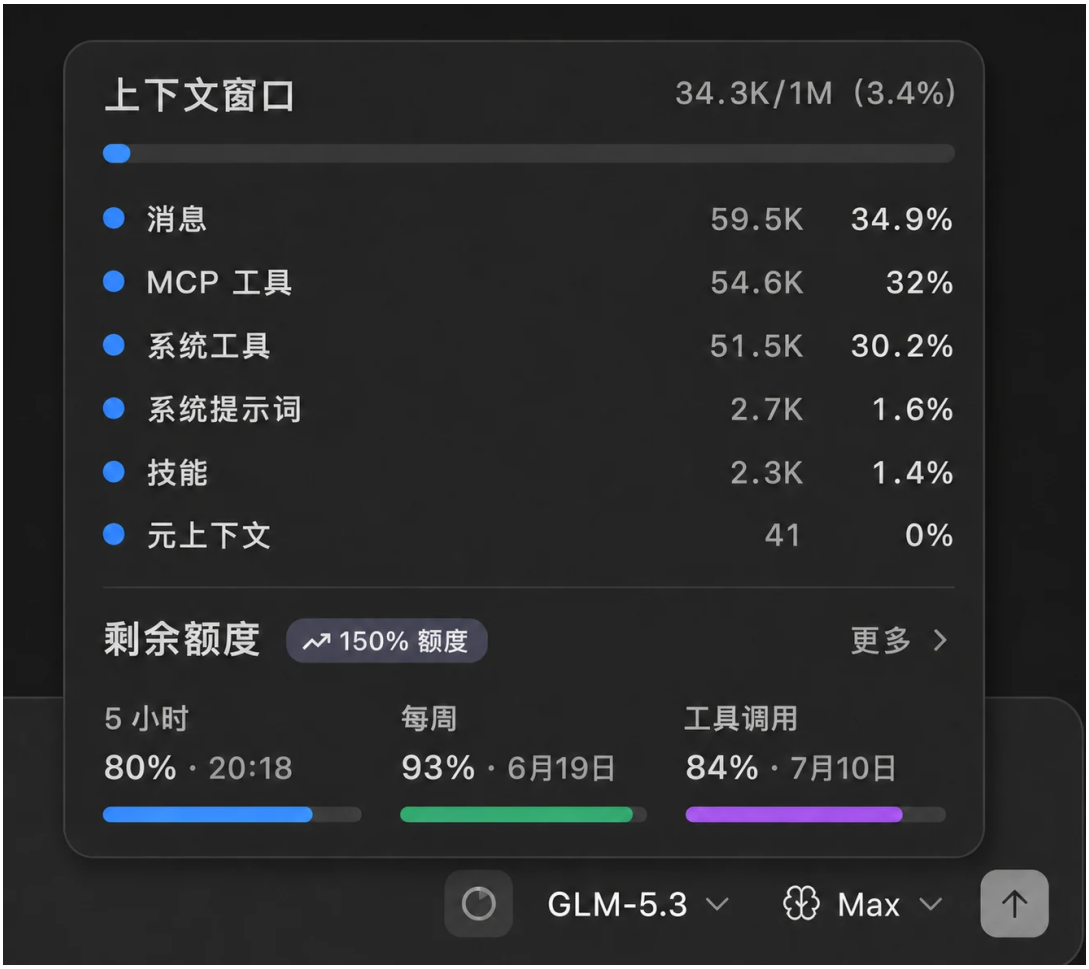
22.4 编程套餐



这里展示：

- **5 小时 prompt 池**、**每周额度**、**MCP 每月额度**等套餐状态；
- GLM-5.3、GLM-5.3-Flash 等模型的 Token 消耗；
- 联网搜索 MCP、网页读取 MCP 等工具调用次数。

点击或展开额度信息时，还可以查看当前上下文窗口的组成、消息和工具调用占比，以及 5 小时、每周、工具调用等剩余额度。


上下文窗口
34.3K/1M (3.4%)
消息 59.5K 34.9%
MCP 工具 54.6K 32%
系统工具 51.5K 30.2%
系统提示词 2.7K 1.6%
技能 2.3K 1.4%
元上下文 41 0%
剩余额度
150% 额度
更多
5 小时
80% · 20:18
每周
93% · 6月19日
工具调用
84% · 7月10日
GLM-5.3
Max

22.5 额度重置卡


模型设置
管理自定义模型供应商，配置后可在聊天时选择使用。
智谱
BigModel
添加供应商
BigModel 已启用
连接方式 个人套餐
GLM Coding Max V2 150% 配额
到期 2027年7月15日 · 管理 · 解绑
续期
有重置机会时，这里显示次数与剩余有效期
剩余额度 1 次重置额度 剩余 4 小时 44 分
点击即可重置对应额度
5 小时剩余
100%
每周剩余 重置
99% 8月20日
MCP 额度
100% 9月15日
模型列表
GLM-5.3 1M
GLM-5-Turbo 200K
添加模型



5 小时额度或每周额度用完后，不一定要干等下一个周期。ZCode **3.8.1 及以上版本**会在端内显示当前可用的 **5 小时 / 每周额度重置机会**：有重置机会时，直接在端内点击即可重置对应额度。GLM Coding Plan 其他活动发放的重置卡也会在端内一并显示，使用方式相同。

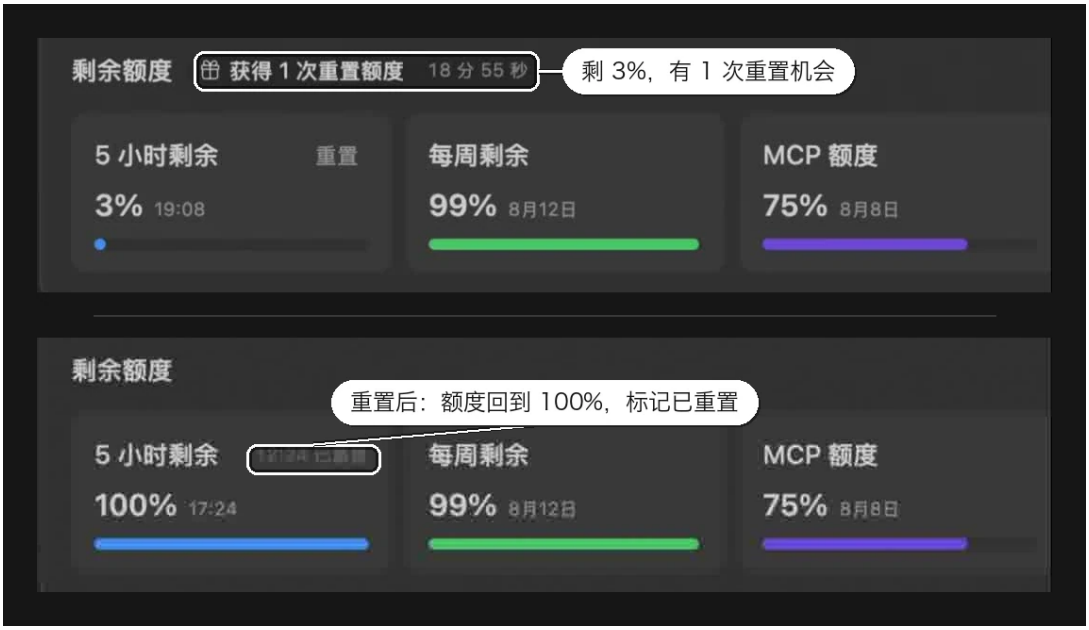
> **注意（获取条件）**：重置卡只会下发给**登录 ZCode 并连接了当前账号 Coding Plan** 的用户。如果没有登录账号、只用 API Key 接入，或者登录的账号没有开通 Coding Plan（即使配置了 API Key），**都不会收到重置卡**。

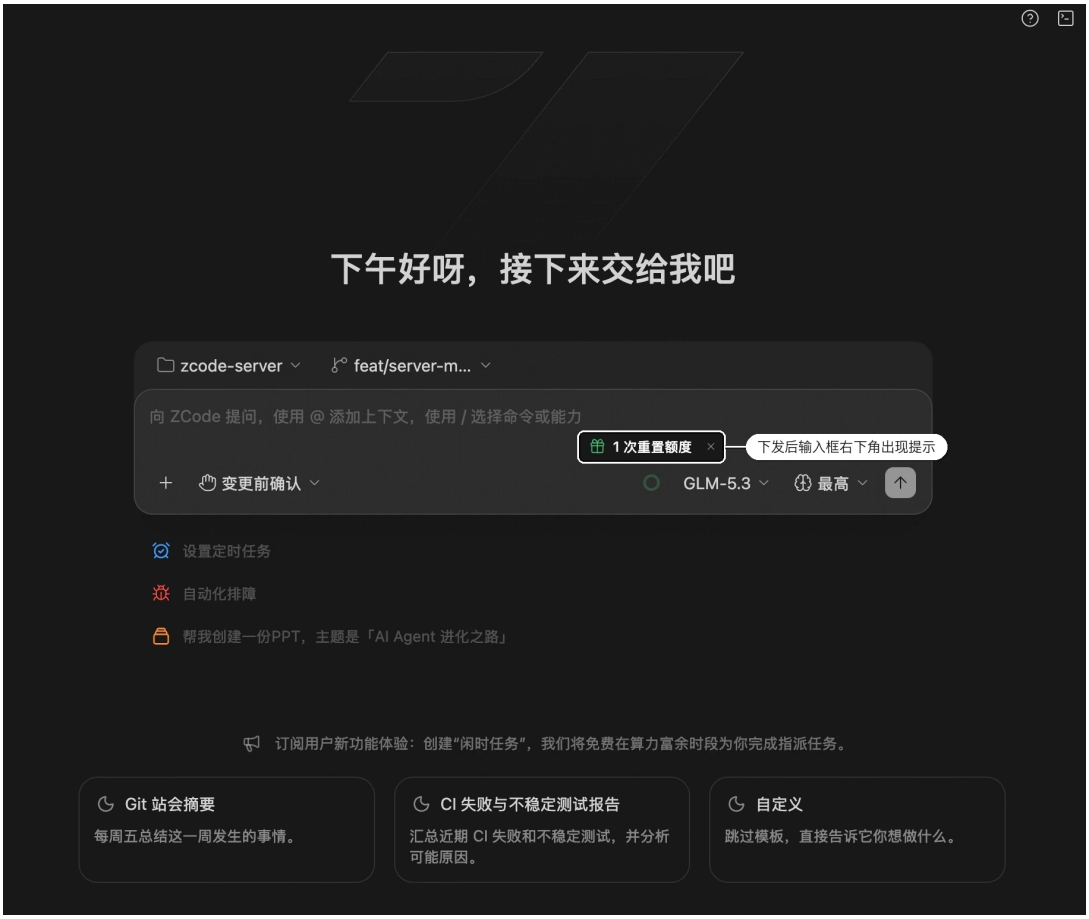
闲时自动下发：ZCode 会在**闲时**自动向 Coding Plan 用户下发 5 小时额度重置卡，需同时满足三个条件：

1. 当前处于闲时时段；
2. 5 小时用量超过阈值；
3. 当日下发次数未达上限。

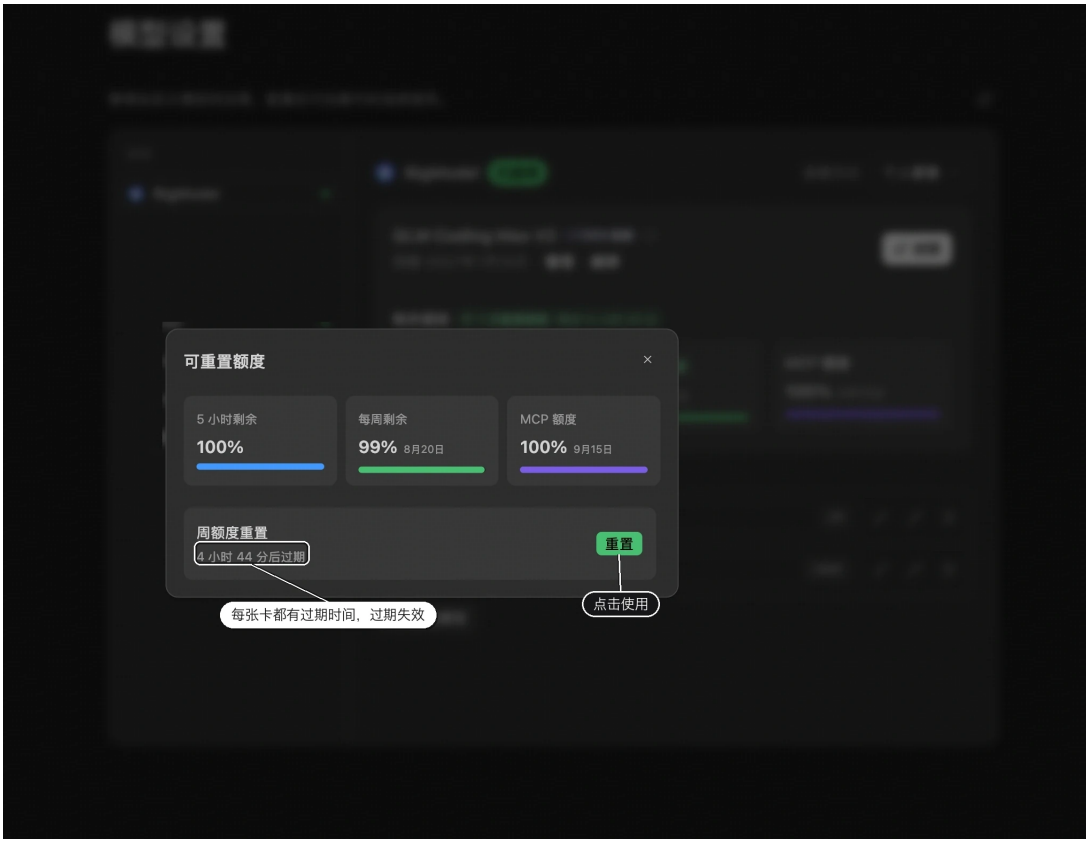
> **注意**：重置卡的闲时时段与**闲时任务**的闲时范围**不是同一套**；具体时段和用量阈值都是动态调整的参数，以端内实际显示为准。下发后，输入框右下角会出现重置额度的提示。

使用规则：





- 重置卡有**有效期**，必须在有效期内使用，过期自动失效；
- 点击重置按钮会弹出「可重置额度」面板，其中列出每张卡的过期时间；
- **有多张时，会自动使用最早获得的那张**；
- 使用统计的「编程套餐」页同样会显示可用的重置按钮，以及累计获得的重置次数；
- 点击「重置」后，对应额度立即恢复到 100%，并在卡片上标记「已重置」。



22.6 成本观测建议

习惯	做法
每周看一次模型用量排行	确认主力模型是否符合预期，避免误用高价模型跑批量任务
长任务前先看剩余额度	避免目标模式跑到一半额度耗尽
非紧急工作提交闲时任务	全程不消耗套餐额度（第 10 章）
开启 Memory 后关注增量	Memory 会产生额外的模型请求
以服务端账单为准	后台 Token 统计与实际消耗可能存在偏差，月底核算以 API 服务商账单为准

22.7 本章小结

- 应用用量看"我做了多少"，编程套餐看"还剩多少"。
- 重置卡只对"登录 + 已连 Coding Plan"的用户下发，有有效期、先获得的先用。
- 闲时任务是订阅用户最直接的省钱手段。

22.8 思考与练习

1. 对比"最近 7 天"与"最近 30 天"的模型用量排行，找出消耗增长最快的模型并分析原因。
2. 把一个本打算现在跑的分析任务改为闲时任务，记录它节省的额度。

第 23 章　反馈、排错与最佳实践

23.1 本章目标

- 选择正确的反馈渠道并提供有效信息
- 掌握日志打包方法
- 建立一套可复用的日常工作流

23.2 四种反馈渠道

渠道	适合
用户反馈表（飞书）	一次性提交完整问题信息
应用内反馈	主动提交建议或从报错提示快速上报
GitHub Issues（`zai-org/feedback`）	需要追踪进展、与团队公开讨论的问题
加入社群（优先推荐飞书群）	直接与官方团队交流

官方社群入口（官方文档提供的官方渠道）：

渠道	二维码
飞书群（推荐，官方团队驻群响应）	

渠道	二维码
微信群（用户交流群）	
微信公众号（获取最新动态）	

渠道	二维码
小红书社群	

23.3 应用内反馈入口

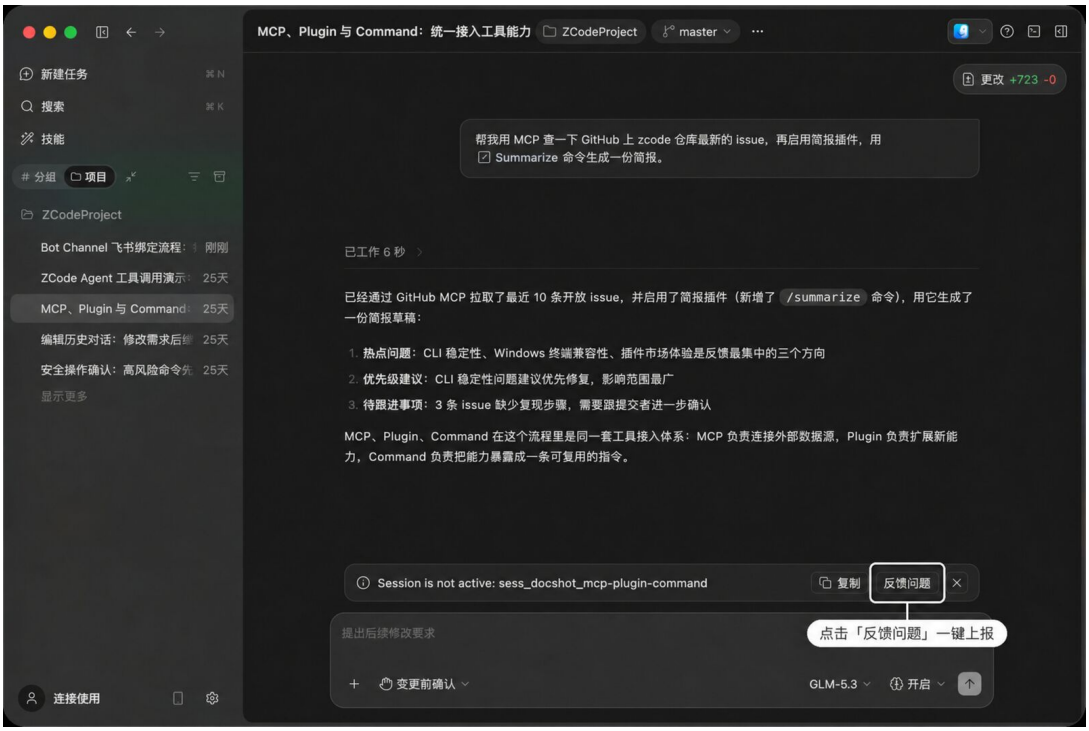
点击右上角的反馈入口，展开统一菜单：



项	作用
问题上报	打开「提交反馈」面板，提交 Bug、异常表现，支持附带截图与日志

项	作用
给产品提需求	希望支持的新能力或体验优化，直达产品团队
用户社群	获取飞书 / 微信社群入口
产品文档	跳转到当前文档站
导出日志	手动导出本地日志压缩包

报错提示入口：如果界面出现错误提示，可以直接点击提示条右侧的 **反馈问题**。这种入口会自动带入报错摘要、现场截图和当前会话信息，适合反馈初始化失败、运行时报错等问题。



23.4 提交反馈需要填什么



- **问题描述**：说明遇到的问题与复现步骤，描述越具体，定位越快；
- **截图**：支持粘贴、拖拽或选择文件，最多 5 张；从报错入口进入时会自动带上现场截图；
- **联系方式**：留下邮箱等，便于需要更多信息时联系；
- **日志**：点击 **上传诊断日志** 后底部日志上传选项默认开启，也可手动关闭。上传的日志已做**脱敏处理**。

工单 ID：在「提交反馈」面板左下角点击 **我的反馈**，可以查看每条反馈的处理进度：状态分 **已提交、开发中、已解决** 三种，每条反馈生成全局唯一的**工单 ID**（形如 `fb-20260712-9f21`），点击右侧图标即可复制。



> **建议**：工单 ID 关联着问题描述、截图、诊断日志与环境信息。在社群沟通或跟进进展时**务必附上工单 ID**，开发同学可凭此直接检索到完整上下文，显著缩短定位、复现与修复周期。

23.5 GitHub Issue 提交格式

- **标题**：一句话概括问题，例如「macOS 上启动后白屏」「Bot @ 触发任务无响应」；
- **使用环境**：操作系统及版本、ZCode 版本、所接入的模型通道；
- **复现步骤**：分步骤说明如何稳定触发；
- **期望表现 vs 实际表现**：分别描述；
- **日志与截图**：附上日志压缩包，有截图、录屏一并上传。

23.6 如何打包日志

推荐优先使用内置的 **导出日志** 功能；导出失败再手动定位。

系统	步骤
macOS	右上角统一反馈入口 → **导出日志** → 保存压缩包 → 上传。失败时 Finder 选择"前往 → 前往文件夹..."，输入 `~/.zcode` 后手动压缩 `logs` 文件夹
Windows	右上角统一反馈入口 → **导出日志** → 保存 → 上传。失败时文件资源管理器地址栏输入 `%USERPROFILE%\.zcode` 回车，选中 `logs` 文件夹右键"发送到 → 压缩(zipped)文件夹"

此外，Hook 相关的诊断日志在 `~/.zcode/cli/log/zcode-<日期>.jsonl`（第 19 章）。

23.7 排错速查表

症状	优先检查
终端能联网、ZCode 连不上模型	设置 → 常规 → HTTP 代理是否填写（留空不读系统环境变量）；改完是否重启应用
模型返回 400 参数错误	思考强度档位是否与第三方部署不匹配（如 DeepSeek V4 选最高档）
Coding Plan 额度没被消耗	OpenAI 接口地址是否用了通用端点而非 `/api/coding/paas/v4`
Skill 面板看得到但模型不用	技能是否过多导致元数据降级；`description` 是否写清触发条件；子智能体是否限制了 `tools`；插件是否停用
Hook 不生效	是否写在项目级配置（当前不执行）；`hooks.enabled` 是否为 true；插件是否启用；是否新建了 session
MCP 服务消失	是否在 `.agents/mcp.json` 维护后又用面板新建（`.zcode` 强优先会整体跳过 `.agents`）
Wiki 生成失败	模型响应是否缺标准文本字段；是否超时（180s 固定上限）；重试次数是否为 0
定时任务没跑	电脑是否睡眠；是否开启"保持唤醒"；任务数是否已达 20 条上限
闲时任务创建按钮置灰	是否为有效个人版 Coding Plan 订阅；是否选中了对应模型连接；当日创建数是否达上限
远程连接失败	看连接日志最后一条 `ERROR`：主机不可达 / 端口错误 / 凭据错误 / 远端缺工具 / 容器未运行

23.8 一套可复用的日常工作流

场景 A：接手陌生项目

1. 打开工作区 → 生成 **Wiki**（第 8 章），通读主执行链路；
2. 开启 **Memory**（第 9 章），让 Agent 逐步积累项目事实；
3. 写一份 `AGENTS.md`（第 4 章），把团队约定固化；
4. 用 **Explore 子智能体**（第 14 章）做改动前调研；
5. 用 "仅显示变更文件 + 添加到聊天"（第 7 章）做提交前自查。

场景 B：实现一个新功能

1. 先用**计划模式**让 Agent 出方案，人工校准；
2. 切到**自动编辑**，让 Agent 实现；
3. 用**内置浏览器**（第 6 章）让它自己验证前端效果；
4. 用 `/goal` 设定"测试全部通过 + 无 lint 报错"的可校验目标（第 5 章）；

5. Review 变更（第 11 章），不满意就用"对话 + 文件重置"回滚。

场景 C：日常维护自动化

1. 用模板创建"工作日晨会动态""周五发布简报"定时任务（第 10 章）；
2. 把非紧急的重活提交为闲时任务，节省额度；
3. 每周看一次**使用统计**（第 22 章），核对主力模型消耗。

场景 D：团队能力沉淀

1. 把固定流程写成 **Skill**（第 15 章）；
2. 把高频提示词写成 **Command**（第 16 章）；
3. 用 **Plugin**（第 17 章）打包 Skill + Command + MCP + Hooks；
4. 推到自建市场仓库，团队一次添加即可共享；
5. 用 **Hooks**（第 19 章）把"提交前必须跑类型检查""生产目录禁止写入"变成机器约束。

23.9 本章小结

- 反馈要带工单 ID 与日志；日志优先用内置导出，失败再手动打包 `~/.zcode/logs`。
- 排错先分清"网络 / 模型 / 配置 / 权限 / 生命周期"五类根因。
- 四个场景工作流可以作为团队落地 ZCode 的起点。

23.10 思考与练习

1. 按"场景 A"完整跑一遍你最不熟悉的项目，记录每一步的输出质量。
2. 按"场景 D"为你的团队做一个插件，并写出它的发版检查清单。

附录

附录 A　快捷键表

A.1 全局操作

Windows	Mac	描述
`Ctrl+N`	`Command+N`	新建任务
`Ctrl+B`	`Command+B`	切换侧栏
`Ctrl+J`	`Command+J`	切换终端面板
`Ctrl+Alt+B`	`Option+Command+B`	切换右侧面板
`Ctrl+K`	`Command+K`	打开命令中心
`Ctrl+F`	`Command+F`	查找对话
`Ctrl+[`	`Command+[`	后退到上一个任务或工作区

Windows	Mac	描述
`Ctrl+]`	`Command+]`	前进到下一个任务或工作区
`Ctrl+Shift+[`	`Shift+Command+[`	切换到上一个对话
`Ctrl+Shift+]`	`Shift+Command+]`	切换到下一个对话

A.2 快速切换菜单

Windows	Mac	描述
`Ctrl+M`	`Control+M`	打开模型选择菜单
`Shift+Tab`	`Shift+Tab`	切换执行模式（聊天输入框聚焦时）
`Ctrl+T`	`Control+T`	打开 Thought Level 选择菜单
`↑` / `↓`	`↑` / `↓`	在 Slash 或 Mention 面板中切换选项
`Enter` / `Tab`	`Enter` / `Tab`	选中当前高亮项
`Esc`	`Esc`	关闭当前面板或菜单

> **说明**：官方文档只收录最常用的 16 条，未堆砌系统原生菜单中的低频项。

附录 B　配置文件与目录速查

B.1 全局目录结构

```
~/.zcode/
├── AGENTS.md                          # 用户全局项目指令（第 4 章）
├── agents/<name>.md                   # 用户级自定义子智能体（第 14 章）
├── commands/                          # 用户级自定义命令 .md（第 16 章）
├── skills/<skill-name>/SKILL.md       # 用户级技能（第 15 章）
├── cli/
|   ├── config.json                    # 用户级 MCP / Hooks 配置（第 18、19 章）
|   ├── memories/projects/<p>/memory/  # Memory 记忆文件（第 9 章）
|   └── log/zcode-<日期>.jsonl         # Hook 诊断日志（第 19 章）
├── v2/
|   ├── config.json                    # 供应商连接参数（第 3 章）
|   └── repo-wiki/<hash>/wiki.json     # 仓库 Wiki 生成结果（第 8 章）
└── logs/                              # 应用日志（反馈时打包，第 23 章）
```

B.2 工作区（项目）目录

```
<项目根>/
├── AGENTS.md                          # 工作区项目指令（第 4 章）
├── .zcode/config.json                 # 工作区级 MCP 配置（第 18 章）
└── .agents/mcp.json                   # .agents 兼容的 MCP 配置（第 18 章）
```

B.3 MCP 配置键与优先级

作用域	文件	键
用户	`~/.zcode/cli/config.json`	`mcp.servers`
工作区	`<项目根>/.zcode/config.json`	`mcp.servers`
用户（兼容）	`~/.agents/mcp.json`	`mcpServers`
工作区（兼容）	`<项目根>/.agents/mcp.json`	`mcpServers`

优先级：**工作区优先加载；同作用域内 `.zcode` 强优先于 `.agents`（不合并）；面板写入永远落在 `.zcode`。**

B.4 Hooks 配置来源

来源	文件	生效条件
用户级	`~/.zcode/cli/config.json`	需 `hooks.enabled: true`
插件	`<插件>/hooks/hooks.json`	标准位置自动发现，随插件启停
旧配置迁移	`.agents/settings.json`、`.claude/settings.json`	**只读展示，不直接执行**，需在设置页导入
项目级	`<workspace>/.zcode/config.json`	**当前版本不会执行**

B.5 插件目录与清单位置

```
my-plugin/
├── .zcode-plugin/plugin.json   ← 优先
├── .claude-plugin/plugin.json  ← 兼容回退
├── commands/*.md
├── skills/<名>/SKILL.md        ← 必须单层目录
├── agents/*.md
├── hooks/hooks.json
└── .mcp.json
```

B.6 常用端点地址

场景	BigModel	Z.ai
Coding Plan · OpenAI	`https://open.bigmodel.cn/api/coding/paas/v4`	`https://api.z.ai/api/coding/paas/v4`
Coding Plan · Anthropic	`https://open.bigmodel.cn/api/anthropic`	`https://api.z.ai/api/anthropic`
资源包 / 余额 · OpenAI	`https://open.bigmodel.cn/api/paas/v4`	`https://api.z.ai/api/paas/v4`

附录 C　术语表

术语	含义
ADE（Agentic Development Environment）	智能体开发环境。以 AI Agent 为执行主体、以任务为最小工作单位的开发环境
ZCode Agent	ZCode 自研的默认智能体，针对 GLM-5.3 系列深度适配
Long Horizon Task	长程任务。跨度长、步骤多、需要多轮工具调用的开发任务
工作区（Workspace）	ZCode 当前处理的项目目录；Agent 在此读取文件、执行命令、管理变更
任务（Task）	推动 Agent 工作的基本单位，承载对话、上下文、权限与 Git 状态
目标模式（Goal）	用 `/goal` 为会话设定可校验目标，Agent 每轮自动校验并迭代直至完成
执行模式	四档权限策略：变更前确认 / 自动编辑 / 计划模式 / 完全访问
推理强度（Thought Level）	控制模型思考深度的档位，对应 GLM-5.3 的 Flexible Effort
Skill	用 `SKILL.md` 描述的可复用工作方法，通过 `$` 调用

术语	含义
Command	保存的常用提示词，通过 `/` 调用；内置只有 `/goal` 与 `/compact`
Plugin	把 Skill / Command / Agent / MCP / Hook 打包的扩展单元
MCP（Model Context Protocol）	接入外部工具与数据源的标准协议，支持 stdio / SSE / HTTP
Hooks	在会话与工具调用生命周期事件上自动执行本地进程的机制
子智能体（Subagent）	主 Agent 在独立上下文中启动的专用 Agent，完成后汇总结果
Memory	项目级长期记忆，由 Agent 自动提炼，存于本机、按项目隔离
Wiki	基于代码库自动生成的架构导读，结论带源码位置
闲时任务	不排时间表、排队等算力富余时免费执行的一次性任务
Remote Control	手机扫码临时接入桌面端窗口的控制界面
Bot Channel	把微信 / 飞书等聊天工具接入 ZCode 的长期入口
AGENTS.md	ZCode 持续读取的项目指令文件（用户全局 + 工作区两个来源）
Flexible Effort	GLM-5.3 的弹性推理强度能力，对应产品中的推理强度开关
BYOK（Bring Your Own Key）	自带 API Key 接入第三方模型服务

附录 D　综合实战：一个完整项目的全流程演练

本附录把全书能力串成一条完整链路。任务设定：**为现有 Web 项目增加"导出订单 CSV"功能，并保证测试通过。**

D.1 阶段一：理解项目（第 7、8、14 章）

1. 打开工作区，生成仓库 Wiki，重点读"主执行链路"与"数据与状态流"；
2. 开启 Memory（设置 → 常规），让后续会话自动带入项目事实；
3. 写一份 `AGENTS.md`：技术栈、包管理器、验证命令、禁止改动目录；
4. 派发 Explore 子智能体调研：

```
请先用 Explore 调研这个模块的调用链，再总结修改入口和风险点。
```

D.2 阶段二：定方案（第 4 章）

切到**计划模式**，提出需求：

```
为订单列表页增加"导出 CSV"按钮。
范围：只修改前端页面与一个新增的工具函数文件。
要求：不新增第三方依赖，复用现有的日期格式化工具。
请先给出方案，不要改文件。
```

人工校准方案后再进入下一阶段。

D.3 阶段三：实现（第 4、5 章）

切换到**自动编辑**，并设定可校验目标：

```
/goal 实现导出订单 CSV 功能，pnpm test 全部通过且 pnpm lint 无报错
```

观察摘要面板每轮的清单项与校验结论。

D.4 阶段四：验证（第 6、21 章）

1. 让 Agent 用内置浏览器打开订单页，点击导出按钮，确认下载行为与文件内容；
2. 用元素选择选中导出按钮，追加要求："把按钮文案改为'导出 CSV'并加到筛选栏右侧"；
3. 在终端后台跑一次完整测试，把输出交给 Agent 分析。

D.5 阶段五：审查与收敛（第 7、11 章）

1. 文件树切到"仅显示变更文件"，核对改动范围；
2. 把变更文件"添加到聊天"，让 Agent 自查并生成提交说明；
3. 若某轮改错，用"对话 + 文件重置"回滚；
4. 检查 Git 图谱确认分支关系，提交代码。

D.6 阶段六：沉淀与自动化（第 10、15—19 章）

1. 把"新增页面功能"的流程沉淀为 Skill；
2. 把"生成提交说明""跑提交前检查"沉淀为 Command；
3. 配置一个 `PostToolUse` Hook：文件写入后自动跑 lint；
4. 用插件把上述能力打包，推到团队市场仓库；
5. 创建一个每周五的"发布简报"定时任务，打开"保持唤醒"；
6. 把"补齐全项目 JSDoc"提交为闲时任务，不消耗套餐额度。

D.7 阶段七：复盘（第 22、23 章）

1. 查看使用统计：本次任务消耗多少 Token、主力模型是谁；
2. 对比闲时任务与实时任务的额度消耗；
3. 若遇到缺陷，带上日志与工单 ID 提交反馈。

D.8 演练检查清单

- [ ] Wiki 已生成，主执行链路已核对
- [ ] `AGENTS.md` 已写入技术栈、验证命令、禁改目录
- [ ] 方案经计划模式人工校准
- [ ] `/goal` 目标可校验（有命令可验证）
- [ ] Agent 已用浏览器自行验证前端效果
- [ ] 变更文件范围符合预期
- [ ] 测试与 lint 均通过
- [ ] 能力已沉淀为 Skill / Command / Plugin
- [ ] 成本已在统计页核对

后记

ZCode 的价值不在于"AI 能写多少代码"，而在于**把一个开发目标从需求推进到验证收口的整个过程组织起来。**

用好它的关键有三点：

1. **说清楚**：目标、范围、约束、验收标准，四样都写全；
2. **管得住**：按风险选择执行模式，用 Hooks 做机器校验，用 Git 做最终防线；
3. **沉下来**：把反复使用的流程变成 Skill、Command、Plugin，让团队能力可复用、可分发。

官方文档持续更新，本书以 https://zcode.z.ai/cn/docs/welcome 为准绳编写；实际使用时请以端内显示与官方最新文档为准。

> 愿你在 ZCode 里，把时间花在真正重要的判断上。