

Claude Code 记忆系统：源码级深度研究报告

版本：v1.0 | 日期：2026-04-05 | 基于版本：Claude Code v2.1.88 源码分析

摘要

本报告基于 2026 年 3 月 Claude Code 源码泄露事件（v2.1.88，1,906 个文件，512,000+ 行 TypeScript 代码）后涌现的多方独立分析，对 Claude Code 的记忆系统进行源码级深度解读。

Claude Code 的记忆系统并非简单的"笔记功能"，而是一套**文件化连续性引擎**——它把长期协作所需的连续性拆成若干类 Markdown 工件，再用不同的写入节拍、召回策略和治理规则将其接回运行时。整套系统涵盖六层连续性架构（Auto Memory、Session Memory、KAIROS 日志、AutoDream、Agent Memory、Team Memory）、三路 prompt 注入机制、基于 Sonnet 的主动召回、四级渐进压缩管线，以及包含 freshness 标记、后校验和怀疑式写入在内的防漂移治理体系。

本报告从运行时全景出发，逐层拆解 prompt 拼接、记忆分类、多尺度架构、召回与注入机制、生产与治理流程、上下文压缩策略，最终提炼出一套可迁移到其他 AI Agent 系统的设计原则。

一、引言：为什么需要研究 Claude Code 的记忆系统

1.1 背景

2026 年 3 月下旬，Claude Code v2.1.88 的完整源码意外泄露。这批代码包含 1,906 个文件、超过 512,000 行 TypeScript 代码，构建在 **Bun** 运行时之上，使用 **React + Ink** 进行终端 UI 渲染。

泄露事件引发了技术社区的广泛分析。从 [db0.ai](#) 的源码级拆解，到 Ars Technica 的主流报道，再到中文社区叉烧、青稞等公众号的万字解析，多个独立来源对同一套代码形成了交叉验证。

1.2 研究价值

Claude Code 的记忆系统代表了当前 AI Agent 记忆工程的最成熟实践。它的价值不在于"有记忆"，而在于把以下能力工程化地整合到了一起：

- 主题化存储与索引注入
- 选择性召回与 token 预算控制
- 会话压缩与状态外化

- 即时提炼与延迟整固
- 权限约束与过期治理
- 团队同步与安全扫描

这使得 Claude Code 的记忆不再是"隐形成态", 而是一个本地可审计、可编辑、可回收、可纠偏的运行时子系统。

1.3 资料来源

本报告综合以下来源：

类别	来源	数量
中文源码深度分析	叉烧（记忆系统专题）、青稞（运行机制与 Memory 模块）	2 篇
英文技术分析	db0.ai 、ClawDecode、 dev.to （多位作者）	5 篇
主流媒体报道	Ars Technica	1 篇
二次消化稿	基于上述原文的架构级 digest	2 篇

二、运行时全景：Claude Code 不是聊天工具，而是一台会话机器

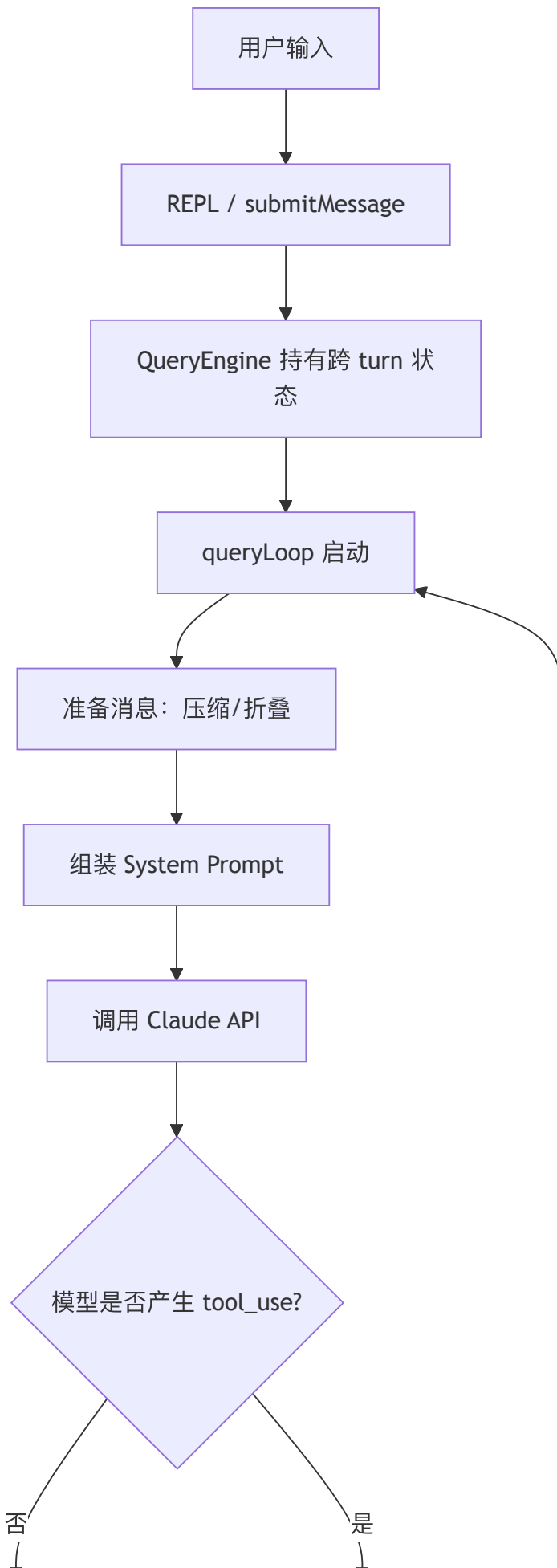
2.1 核心判断

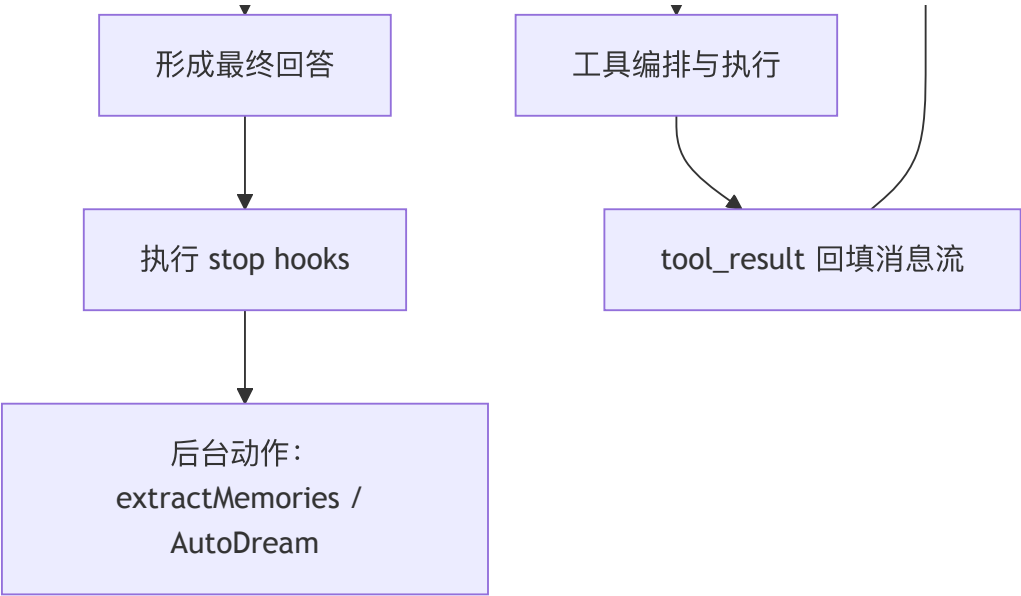
把 Claude Code 理解成"终端里能调工具的大模型"是不够的。更准确的定位是：

Claude Code 是一个 query loop，外面包着 prompt 组装、上下文注入、工具编排、压缩治理、记忆读写、子代理和 stop hooks 的运行时宿主。

它真正维护的不是单条回复，而是一套能跨 turn 持续推进的执行状态。

2.2 queryLoop 生命周期





一个回合（turn）的定义不是"一问一答"，而是一个直到模型不再调用工具才真正收束的执行循环。

2.3 工具编排

Claude Code 内建约 **43 个工具**（含内部构建专用工具），涵盖文件读写、搜索、终端执行、Web 访问、子代理调度等。工具执行遵循以下策略：

执行模式	适用场景	说明
并行执行	只读工具（Read、Grep、Glob）	多个工具同时运行，不互相阻塞
串行执行	写入工具（Edit、Write、Bash）	按顺序执行，确保文件系统一致性
子代理隔离	Agent 工具	独立上下文，权限可定制

2.4 核心运行时组件

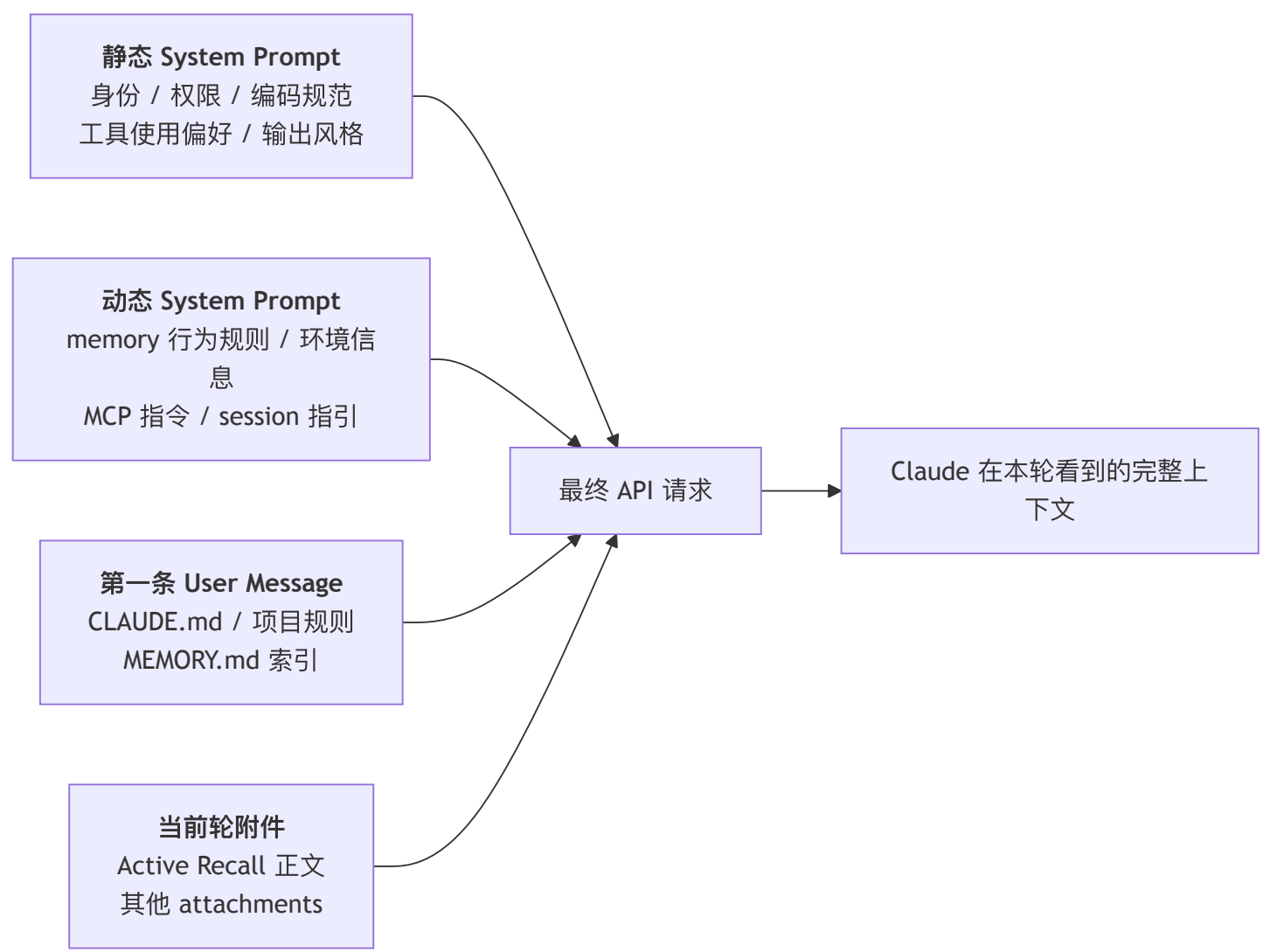
组件	职责	关键文件
REPL	接收用户输入，管理会话生命周期	—
QueryEngine	持有跨 turn 状态，驱动 queryLoop	src/query.ts
System Prompt	分层组装模型指令	src/constants/prompts.ts
Tool Orchestrator	管理工具权限与并发策略	—
Stop Hooks	回合结束后触发后台任务	—
Memory Runtime	记忆读写、召回、整固	src/memdir/

三、Prompt 的分层拼接：规则、索引与正文的三路注入

3.1 核心洞察

Claude Code 没有把所有上下文粗暴塞进一坨 system prompt。它至少把"模型看到的世界"拆成了三层，每层有不同的缓存语义和更新频率。

3.2 三层架构



3.3 三层对比

维度	静态 System Prompt	动态 System Prompt	第一条 User Message
典型内容	身份定义、 权限规则、 编码规范、 工具偏好、输出风格	Memory 行为规则、环境信息、MCP 指令、session 指引	CLAUDE.md、 项目规则、MEMORY.md 索引

维度	静态 System Prompt	动态 System Prompt	第一条 User Message
更新频率	极少变化 (跨版本稳定)	会话内基本稳定 (通过 memoize 冻结)	每个会话加载一次
缓存策略	全局共享缓存, 前缀复用率最高	session 内缓存, 不跨用户共享	不缓存, 但体积小
注入位置	system 消息开头	system 消息尾部	第一条 user 消息, 包裹在 <system-reminder> 中
工程意义	最大化 prompt cache hit rate	区分"通用规则"和"会话特定规则"	规则放规则层, 数据放数据层

关键设计原则：system prompt 负责告诉模型怎么使用记忆，user message 负责告诉模型当前有哪些记忆。这不是措辞差异，而是缓存语义、上下文成本和职责边界的差异。

3.4 静态 / 动态分割点

根据源码分析，system prompt 共计约 **914 行**，在此处分为静态和动态两部分。静态部分在所有用户、所有会话间共享前缀缓存；动态部分的"动态"并不意味着每轮都变，而是指它不能假设对所有用户、所有会话都相同。

四、记忆系统设计哲学：文件化连续性引擎

4.1 核心定位

Claude Code 把长期协作所需的连续性，拆成若干类 **Markdown** 工件，再用不同的写入节拍、召回策略和治理规则把它们接回运行时。

这套设计的关键不在"记住更多"，而在于四个"不"：

- 不依赖外部向量库——用 plain Markdown + LLM 推理替代 Vector DB + RAG
- 不把所有内容堆进单个大文件——按主题拆分 topic files
- 不把 transcript 直接等同于长期记忆——对话记录是原料，不是最终结构
- 不让每轮都把所有记忆全文灌回模型——先索引、再选择、再读正文

4.2 为什么是文件，而不是数据库

选择本地文件系统而非数据库（或向量库）带来四个直接收益：

- ① 可审计

用户能直接打开文件查看记忆内容
- ② 可编辑

模型可用现有的 Read/Edit/Write 工具原地维护
- ③ 可注入

MEMORY.md 索引可直接嵌入 prompt
- ④ 可纠偏

出错时人工可直接修订、删除或比对

这意味着 Claude Code 的 memory 不是"隐状态", 而是**显式工件** (Explicit Artifacts)。在代码协作场景中, 这一点尤其重要——协作知识本来就需要可见、可改、可追责。

4.3 什么该存 vs 什么不该存

应该存入 Memory	不应该存入 Memory
用户身份、技术背景、偏好	代码结构、文件路径、架构模式
用户对协作方式的反馈与纠偏	Git 历史和最近变更
项目目标、里程碑、非代码可见约束	临时修复步骤
外部系统入口（工单、仪表盘、文档）	已写在 CLAUDE.md 里的内容
难以从当前代码重建的协作线索	当前会话的临时状态

这条边界背后的核心原则是：

Memory 只记那些"难以从当前代码和 git 历史稳定推断出来"的协作线索。

能靠 git log 查到的，不存；能靠读代码推断的，不存；会快速过期的路径和函数名，不存。

五、四类语义记忆：按"未来如何使用"分类

5.1 分类哲学

Claude Code 把长期记忆拆成四种语义类型。这套分类最精妙之处在于：它不按"来源"分，而按"以后怎么用"分。

不是在说"这是谁说的", 而是在回答：

这条信息在未来的协作里，扮演什么角色。

5.2 四类记忆详解

user — 用户画像

记录用户是谁、擅长什么、偏好什么、怎样解释更容易懂。

```
----
name: 用户技术背景
description: 用户是资深 Go 开发者，首次接触 React 前端
type: user
----
```

用户有十年 Go 开发经验，但这是第一次接触本项目的 React 前端部分。

****How to apply:**** 前端概念解释时应类比后端模式，避免假设用户熟悉 React 生态。

feedback — 协作纠偏

用户对工作方式的明确反馈——既包括纠错，也包括对非显而易见做法的确认。

```
----
name: 测试策略偏好
description: 集成测试必须连接真实数据库，不要 mock
type: feedback
----
```

集成测试必须连接真实数据库，不要 mock。

****Why:**** 之前因 mock 与生产环境偏差导致迁移问题漏检，曾引发生产事故。

****How to apply:**** 涉及测试设计时，优先选择真实数据库集成测试。

project — 项目上下文

项目目标、背景、截止日期等无法从代码直接推断的约束。

```
---
name: Auth 中间件重写背景
description: auth 重写受法务合规驱动，非技术债务清理
type: project
---
```

Auth 中间件重写是法务合规要求驱动的，不是技术债务清理。

```
**Why:** 法务标记了旧 session token 存储方式不满足新合规要求。
**How to apply:** 范围决策应优先考虑合规需求，而非工程便利性。
```

reference — 外部资源入口

指向外部系统中信息位置的指针。

```
---
name: Pipeline Bug 追踪位置
description: pipeline 相关 bug 在 Linear 项目 INGEST 中追踪
type: reference
---
```

Pipeline 相关 bug 在 Linear 项目 "INGEST" 中追踪。

```
**How to apply:** 用户提及 pipeline 问题时，引导到 Linear INGEST 项目。
```

5.3 目录结构

```
~/ .claude/projects/<sanitized-project-root>/memory/
├─ MEMORY.md                               ← 索引文件 (≤200行 / ≤25KB)
├─ user_expertise_profile.md                ← user 类型
├─ feedback_testing_database_policy.md      ← feedback 类型
├─ project_auth_rewrite_compliance.md       ← project 类型
├─ reference_linear_pipeline_bugs.md        ← reference 类型
├─ reference_grafana_latency_dashboard.md
└─ team/                                   ← Team Memory 子目录
    └─ ...
```

5.4 MEMORY.md 索引示例

MEMORY.md 不存储记忆正文，只做轻量索引——每行不超过 150 个字符：

- [用户技术背景](user_expertise_profile.md) - 资深 Go, 首次接触 React 前端
- [测试策略偏好](feedback_testing_database_policy.md) - 集成测试用真实 DB, 不 mock
- [Auth 重写背景](project_auth_rewrite_compliance.md) - 法务合规驱动, 非技术债
- [Pipeline Bug 追踪](reference_linear_pipeline_bugs.md) - Linear 项目 INGEST
- [延迟监控面板](reference_grafana_latency_dashboard.md) - grafana.internal/d/api-latency

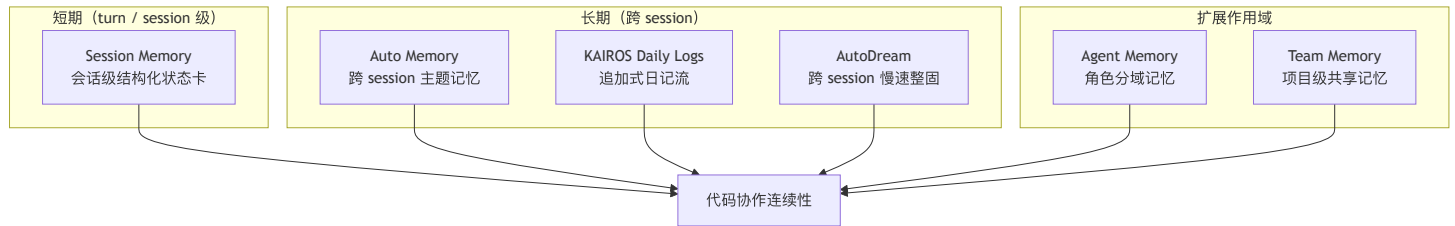
5.5 四类记忆对比

维度	user	feedback	project	reference
核心问题	用户是谁?	怎么协作?	为什么这样做?	去哪找信息?
典型触发	用户自我介绍、能力描述	用户纠正或确认做法	讨论项目背景、决策原因	提及外部系统位置
更新频率	低 (角色稳定)	中 (随协作深化)	高 (项目状态变化)	低 (系统位置稳定)
过期风险	低	低	高 (截止日期、里程碑)	中 (URL 可能变化)
推荐结构	事实 + How to apply	规则 + Why + How to apply	事实 + Why + How to apply	位置 + 用途

六、六层连续性架构：从 turn 到 team 的多尺度系统

6.1 架构总览

Claude Code 的 Memory 不是一个单点存储，而是多层时间尺度的连续性工件体系。



6.2 第一层：Auto Memory — 跨会话持久化主仓

定位：项目级长期记忆目录，存储跨 session 稳定复用的协作线索。

关键澄清：Auto Memory 默认不在 **repo 工作树里**，而在用户目录下的全局 Claude 数据区，以项目路径派生的 slug 做隔离：

```
~/ .claude/projects/<path-hash>/memory/
```

不同 worktree 或子目录通常共享同一个项目 memory 目录。

结构特征：

- 按 topic file 拆分，不堆进一个总文件
- MEMORY.md 做索引，不存正文
- 每个 topic file 用 user / feedback / project / reference 做语义分类

6.3 第二层：Session Memory — 会话级结构化状态卡

定位：不是长期知识库，而是当前 session 的结构化摘要，服务于上下文压缩。

典型结构（基于 `~/ .claude/session-memory/config/template.md`）：

- Current State ← 当前进行到哪一步
- Task specification ← 任务目标
- Files and Functions ← 涉及的关键文件
- Workflow ← 操作顺序
- Errors & Corrections ← 遇到的问题
- Learnings ← 过程中学到的

触发条件：

条件	阈值
首次提取	上下文超过 10,000 tokens
后续更新	新增 ≥5,000 tokens 且 ≥3 次工具调用
单节上限	~2,000 tokens/section
总摘要上限	12,000 tokens

核心价值：

把压缩从一次性行为，变成可复用的中间工件。Session Memory 不是"又做了一个摘要"，而是给 compact 提供现成的低损替身。

6.4 第三层：KAIROS Daily Logs — 追加式日记流

定位：长生命周期场景下的高频记录模式（Agent SDK 守护进程模式）。

存储模式：

```
memory/logs/YYYY/MM/DD/YYYY-MM-DD.md
```

与 Auto Memory 的按主题拆分不同，KAIROS 使用**每天一个日志文件、按时间顺序 append-only**的方式记录。

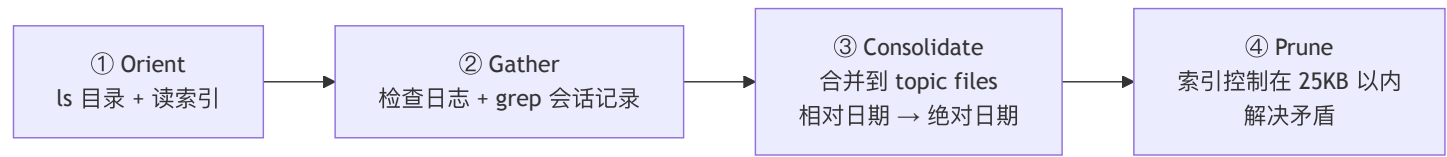
设计考量：

- 避免高频写入导致 topic file 冲突
- 充当"速记流"，后续由 /dream 统一蒸馏为结构化记忆
- 特别适合长时间运行的守护进程场景

6.5 第四层：AutoDream — 慢速整固

定位：不是增加新存储，而是在已有记忆文件上做 consolidation。

四个阶段（源自 src/services/autoDream/consolidationPrompt.ts）：



触发条件：距上次运行 ≥ 24 小时 且 累计 ≥ 5 个新 session。

与 extractMemories 的本质区别：

extractMemories 负责捕获新信号，AutoDream 负责把长期记忆库整理得还能继续用。**快路径负责写入，慢路径负责去噪。**

Anthropic 自己在 prompt 中这样描述 dream：

"You are performing a dream — a reflective pass over your memory files... synthesize into durable, well-organized memories."

6.6 第五层：Agent Memory — 角色分域记忆

自定义 Agent 可以带自己的 memory scope：

Scope	存储位置	用途
user	用户级全局目录	跨项目的用户偏好
project	项目目录下（可被版本控制共享）	项目特定的 Agent 知识
local	本机私有目录	不共享的本地状态

关键原则：角色 prompt 与角色记忆应该一起设计，而不是先有 agent、再外挂一层记忆。

6.7 第六层：Team Memory — 项目级共享记忆

定位：把连续性从个人延伸到协作。物理位置通常在 Auto Memory 的 team/ 子目录。

同步机制：

维度	实现
认证	Anthropic 1st-party OAuth (claude_ai_inference + claude_ai_profile scope)
作用域绑定	GitHub repo remote URL (origin)
效率优化	ETag + checksum，仅上传/下载变化的文件
冲突策略	Pull: server wins per-key; Push: local wins on conflict（不做内容合并）
安全扫描	40+ Gitleaks 规则（AWS、GitHub PAT、Anthropic keys 等），检出 secret 时阻止写入
大小限制	单文件 ≤250KB，PUT 请求体 ≤200KB

6.8 六层对比总览

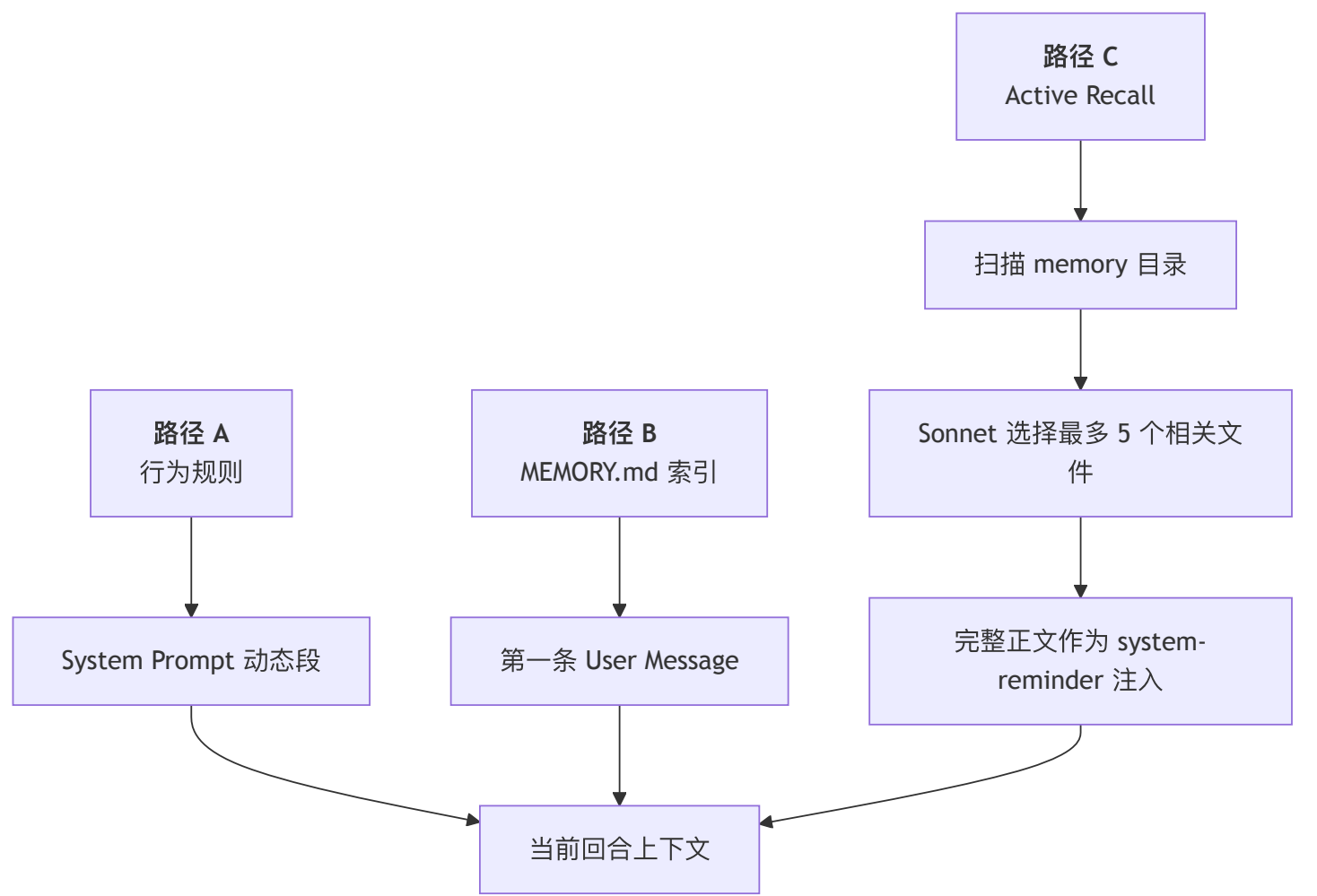
层级	时间尺度	存储位置	写入节拍	物理形态
Auto Memory	跨 session	~/.claude/projects/<slug>/memory/	回合后提炼 / 用户显式写入	topic Markdown files
Session Memory	当前 session	session 级目录	阈值触发自动更新	结构化摘要 summary.md
KAIROS Logs	跨 session	memory/logs/YYYY/MM/DD/	每次交互 append	每日日志文件
AutoDream	跨 session (整理)	同 Auto Memory	≥24h + ≥5 sessions	整理后的 topic files

层级	时间尺度	存储位置	写入节拍	物理形态
Agent Memory	按 scope	user/project/local 目录	Agent 运行时	topic Markdown files
Team Memory	跨协作者	memory/team/ + remote sync	手动或自动同步	共享 Markdown files

七、核心运行时机制：Active Recall 与三路注入

7.1 三条注入路径

很多人以为有了 MEMORY.md 就有了记忆系统——这远远不够。Claude Code 的记忆通过**三条独立路径**进入运行时，各有不同职责：



路径	注入内容	本质	更新频率
A	如何理解和使用记忆的规则	认知规则	session 内冻结（memoize）
B	记忆目录的总览索引	目录视图	session 内冻结（memoize）
C	与当前查询最相关的记忆正文	按需正文	每轮都执行

路径 A 提供"用法规则"，路径 B 提供"目录索引"，路径 C 提供"按需正文"。

路径 A 和 B 都通过 memoize 在 session 内只加载一次。路径 C 是会话中唯一动态感知新记忆的入口——这解释了为什么后台新写入的 memory 在同一会话里仍能被"想起来"：不是因为 system prompt 重新拼了，而是因为 recall 每轮都重新扫目录。

7.2 Active Recall：整个系统的灵魂

Active Recall 是让记忆从"静态笔记"变成"运行时能力"的关键机制。

当前用户输入

递归扫描 Auto Memory 目录
(最多 200 个文件)

读取 frontmatter 摘要
(前 30 行 / 文件名 +
description)

过滤 alreadySurfaced
(本 session 已注入的文件)

构建 manifest 提交给
Sonnet

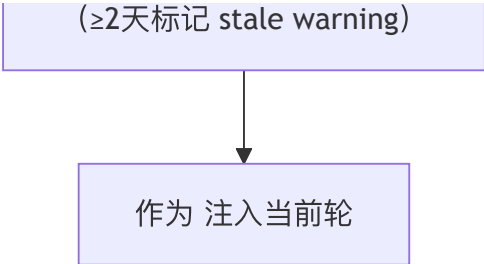
Sonnet 相关性选择
(选出最多 5 个 topic files)

后校验：确认文件真实存在
于候选集

读取完整内容

附加 freshness note





关键设计决策：

- 1. 先摘要后全文：不把几百个文件全文灌进模型，而是先用 frontmatter 缩小范围
- 2. 用便宜模型选择：相关性判断用 Sonnet 而非 Opus，控制成本
- 3. 会话内去重：已经在本 session 注入过的文件不再重复，避免浪费 token
- 4. 后校验防幻觉：模型选完文件名后再验证文件确实存在，防止 LLM 虚构不存在的记忆
- 5. **freshness** 标注：旧记忆自动附带"这可能过期了"的警告

7.3 预算与限制

参数	限制值	说明
目录扫描上限	200 个文件	超出则截断
Frontmatter 扫描	前 30 行/文件	只读 name + description
每轮选择上限	5 个 topic files	由 Sonnet 做相关性排序
单文件大小上限	4 KB (~200 行)	超出则截断
单轮注入预算	20 KB	所有召回文件合计
会话累计预算	60 KB (61,440 bytes)	防止长会话 token 爆炸
MEMORY.md 行数上限	200 行	索引保持稀疏
MEMORY.md 大小上限	25 KB (25,000 bytes)	控制注入成本
每行长度上限	150 字符	索引条目保持精简

7.4 扫描边界

Active Recall 并不扫描所有类型的记忆：

记忆类型	是否在 Recall 扫描范围
Auto Memory（项目级）	是（默认扫描目录）
Team Memory	是（作为 Auto Memory 子目录被递归扫到）

记忆类型	是否在 Recall 扫描范围
Agent Memory	否（不在默认扫描链路）
Session Memory	否（不参与文件级 recall）
KAIROS Logs	否（由 dream 整理后才进入 recall）

7.5 为什么 Active Recall 是灵魂

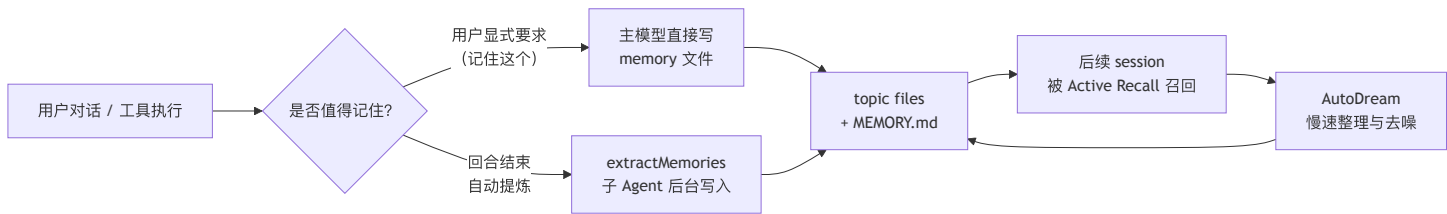
索引负责发现，正文负责解释。

这套方案让记忆行为更像"查档案"而不是"背全文"。它把检索成本从 $O(n \times \text{文件大小})$ 降到了 $O(n \times \text{摘要大小}) + O(k \times \text{文件大小})$ ，其中 $k \leq 5$ 远小于 $n \leq 200$ 。

八、记忆的生产与治理：快写、慢整、防漂移

8.1 记忆生产的三条路径

Claude Code 的记忆不是靠单一机制产生的，而是通过三条不同节拍的路径协同：



路径一：用户显式要求

最直观的方式。用户说"记住我偏向 tabs"或"记住测试不要 mock"，主模型直接写出一个 topic file 并更新 MEMORY.md 索引。

路径二：extractMemories — 回合后自动提炼

这是更有代表性的生产方式。一次 queryLoop 结束后，stop hooks 触发 extractMemories：

1. 后台 fork 一个受限子 Agent
2. 读取最近若干条对话
3. 扫描现有 memory 文件
4. 判断哪些信息值得长期保存
5. 写入或更新 topic files + MEMORY.md

Claude Code 在每轮结束后做一轮"值不值得长期记住"的后台整理。

重要边界： 这条后台自动提炼链路可能受编译开关或运行时 feature flag 控制，不是所有环境都默认开启。

路径三：AutoDream — 跨 session 离线整固

详见第六章 6.5 节。

8.2 extractMemories 子 Agent 权限模型

记忆提取子 Agent 并不拥有全部工具权限。这是一个非常成熟的安全设计：

工具类别	权限	说明
Read / Grep / Glob	可用	需要读取对话和现有记忆
Bash	只读	防止副作用
Edit / Write	仅限 memory 目录	不能修改工作区代码
Agent	不可用	防止递归 fork

"长期记忆维护"是高价值但高风险动作，因此必须缩权限。

子 Agent 的工作策略是**两轮对话**（2-turn strategy）：

- **第一轮：** 读取当前记忆文件和最近对话
- **第二轮：** 写入或更新记忆

这种设计让 extractMemories 共享主 queryLoop 的 prompt cache，降低推理成本。

8.3 extractMemories vs AutoDream 对比

维度	extractMemories	AutoDream
触发时机	每个回合结束后	≥24h 且 ≥5 新 sessions
信息范围	最近若干条消息	多个 session + 已有全部记忆
处理深度	即时提炼，预算有限	四阶段深度整理
操作类型	新增 / 更新 topic files	合并重复 / 修正矛盾 / 删除过期 / 精简索引
运行身份	受限子 Agent（后台 fork）	独立 consolidation Agent
核心价值	捕获新信号	降噪 + 纠偏
一句话定位	快速记笔记	隔段时间再整理

8.4 防漂移三重机制

长期记忆系统最危险的问题不是忘记，而是把旧内容误认成当前状态。Claude Code 用三重机制应对：

机制一：Freshness Warning — 基于时间的过期标记

- 基于文件 mtime 计算年龄：
- 今天 / 昨天 → 视为新鲜，不加警告
 - ≥2 天 → 自动附加 stale warning

明确告诉模型：**memory 是时间点快照，不是实时状态**。尤其涉及以下内容时更不能盲信旧记忆：

- 文件路径和函数名
- Feature flag 状态
- 代码行为和 file:line 级引用

机制二：后校验 — 防止模型幻觉选择不存在的记忆

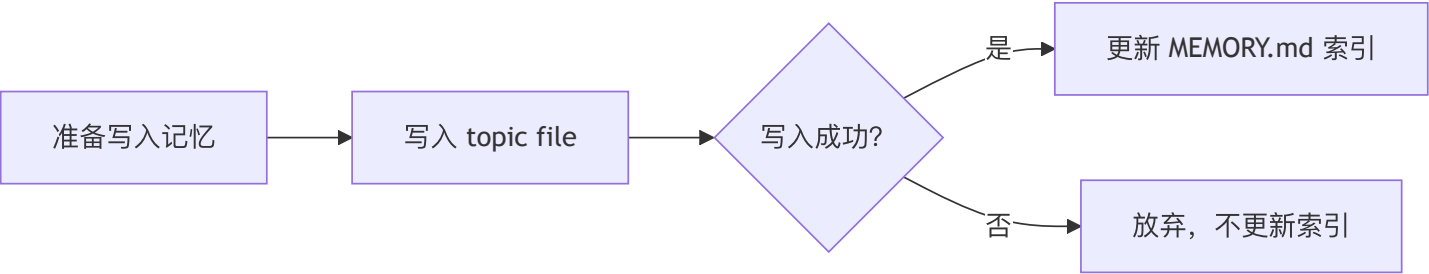
Active Recall 在模型选完文件名后，再校验一次，确保这些文件真的存在于候选集中。

LLM 负责排序和判断，确定性代码负责兜底与边界。

机制三：Skeptical Memory Architecture — 写成功后才更新索引

又称"Strict Write Discipline"：Agent 只在确认 topic file 写入成功后，才更新 MEMORY.md 索引。

这防止了"索引指向不存在文件"的 context pollution——如果写入失败但索引已更新，模型就会引用一个幻影记忆。



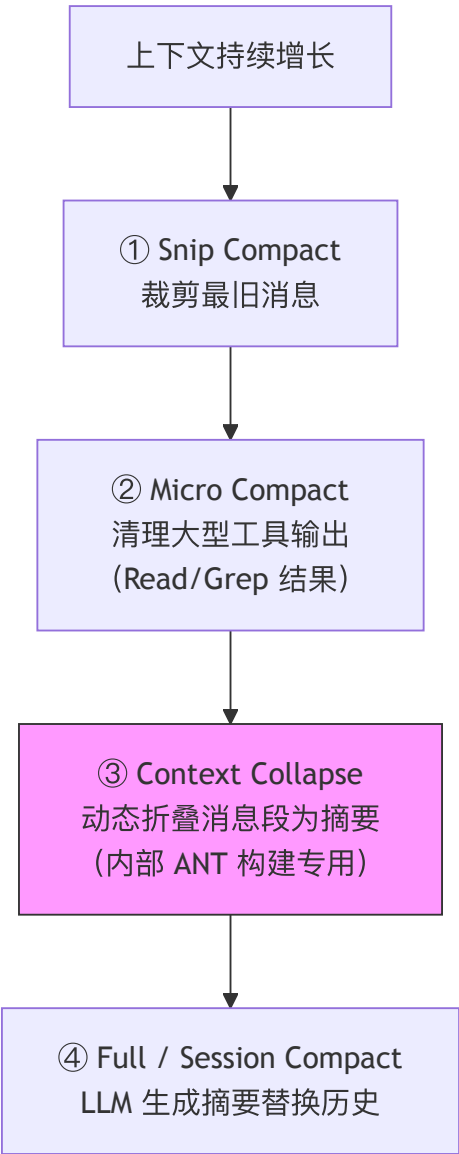
"Don't trust what you remember you wrote — verify against what actually exists on disk."

九、上下文窗口管理：四级渐进压缩

9.1 问题背景

长会话的上下文会不断膨胀。Claude Code 不是等到溢出才处理，而是通过**四级渐进压缩管线**逐步管理。

9.2 四级压缩管线



注：Context Collapse（第三级）为内部构建专用特性，外部发布版本中通过 DCE（Dead Code Elimination）移除。

9.3 各级详解

第一级：Snip Compact

最轻量的一种方式——直接裁剪最旧的消息。无需 LLM 调用，纯机械操作。

第二级：Micro Compact

清理对话中的大型工具输出（Read、Grep 结果等），保留结构但删除正文。

两条触发路径：

路径	触发条件	机制
时间触发	上次消息 >60 分钟前（cache 过期）	本地清理旧工具输出
缓存触发	prompt cache 可用时	通过 <code>cache_edits</code> API 从服务端缓存删除

时间触发路径保留最近 5 个工具结果，其余清空。

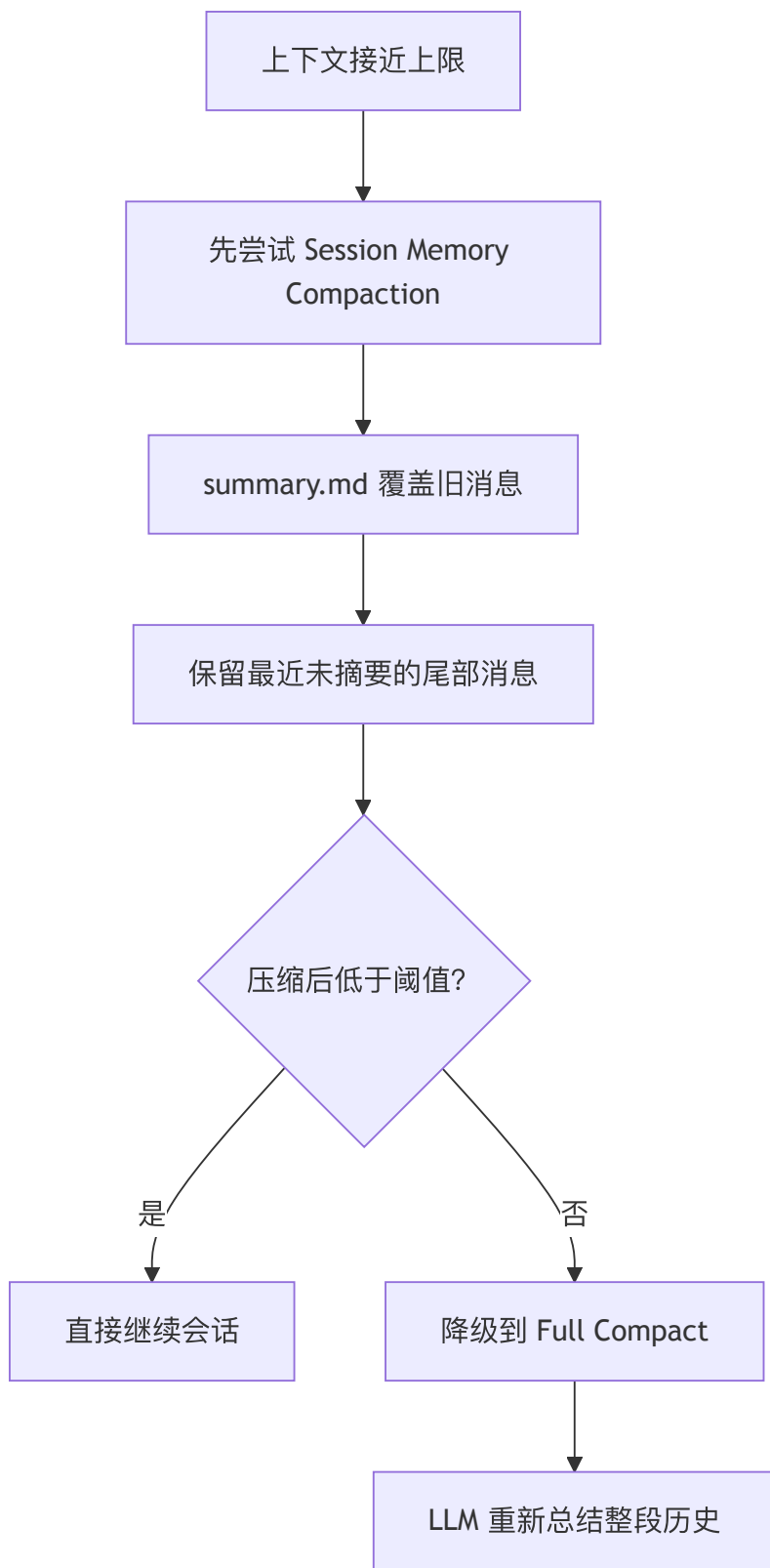
第三级：Context Collapse（内部专用）

动态将消息段折叠成摘要。标记为 `HISTORY_SNIP` / `CONTEXT_COLLAPSE` / `REACTIVE_COMPACT`，在外部构建中通过 DCE 移除。

第四级：Full / Session Compact

最重的操作——调用 LLM 重新总结整段历史。

Session Memory 在此发挥关键作用：Claude Code 先尝试用 `summary.md` 覆盖已摘要过的旧消息，保留最近未摘要的尾部。只有压完仍超阈值，才回退到 Full Compact。



能复用现成状态卡片的地方，就不要每次都重新发起一次昂贵总结。

9.4 关键阈值

常量	值	说明
AUTOCOMPACT_BUFFER	13,000 tokens	触发自动压缩的缓冲区
WARNING_BUFFER	20,000 tokens	发出上下文警告的缓冲区
MAX_SUMMARY_TOKENS	20,000 tokens	摘要最大 token 数
Session Memory 首次提取	10,000 tokens	开始维护 Session Memory 的阈值
Session Memory 更新间隔	5,000 tokens + 3 tool calls	后续更新条件

十、高级特性与未公开系统

10.1 KAIROS 守护进程

KAIROS 是 Claude Code 的 **Assistant Mode** —— 一个即使终端关闭也持续运行的守护进程。

核心机制：

- 使用 <tick> prompts 定期检查是否需要执行动作
- 支持 PROACTIVE flag，允许主动发起操作
- 记忆使用 append-only 日志而非直接修改 topic files
- 通过 /dream 将日志蒸馏为结构化记忆

10.2 多 Agent 协调

Claude Code 支持两种多 Agent 模式：

模式	机制	适用场景
Subagent	嵌套任务，同进程内 fork	单任务分解（如 Explore、Plan）
Teams	tmux + 共享消息总线 + Unix Domain Sockets	并行工程任务

Teams 模式下，多个 Claude Code 实例可以协调工作，类似多工程师并行开发。

10.3 Anti-Distillation 机制

源码中发现了 ANTI_DISTILLATION_CC feature flag —— 注入**伪造的工具定义**，防止竞争对手从 API 日志中提取训练数据。

10.4 内部代号一览

代号	对应功能
memdir	记忆系统的内部名称
Chicago	Computer Use (@ant/computer-use-mcp)
Tengu	下一代功能集代号
Fennec	Opus 4.6 模型
Capybara	Claude 4.6 变体
Numbat	测试用模型
KAIROS	守护进程 / Assistant Mode
Undercover Mode	内部构建自动激活的代码 (外部构建中 strip)

10.5 UltraPlan 与 Bridge Mode

特性	说明	状态
UltraPlan	Opus 级模型起草高层计划 (10-30 分钟运行)	未公开
Bridge Mode	从手机/浏览器通过 Anthropic Dispatch 远程控制	未公开
Coordinator	WebSocket 编排多 worker 并行工程任务	未公开

十一、关键数字速查表

11.1 Memory 系统参数

参数	值	来源
MEMORY.md 行数上限	200 行	db0.ai / 源码
MEMORY.md 大小上限	25,000 bytes (25KB)	db0.ai / 源码
MEMORY.md 每行长度	≤150 字符	ClawDecode
CLAUDE.md 大小上限	40,000 字符	db0.ai
CLAUDE.md import 深度	5 层	db0.ai

参数	值	来源
Topic file 扫描上限	200 个文件	db0.ai
Frontmatter 扫描范围	前 30 行/文件	青稞
Active Recall 每轮选择	≤5 个文件	db0.ai / 多方验证
单文件大小限制	4 KB (~200 行)	db0.ai
单轮注入预算	20 KB	db0.ai
会话累计注入预算	60 KB (61,440 bytes)	db0.ai

11.2 Session Memory 参数

参数	值
首次提取阈值	10,000 tokens
更新间隔	≥5,000 新 tokens 且 ≥3 次工具调用
单节上限	~2,000 tokens
总摘要上限	12,000 tokens

11.3 Compact 参数

参数	值
AUTOCOMPACT_BUFFER	13,000 tokens
WARNING_BUFFER	20,000 tokens
MAX_SUMMARY_TOKENS	20,000 tokens
Micro Compact 时间触发	>60 分钟无活动
Micro Compact 保留数量	最近 5 个工具结果

11.4 AutoDream 参数

参数	值
时间间隔	≥24 小时
session 间隔	≥5 个新 session

参数	值
索引上限	25 KB
锁文件	.consolidate-lock

11.5 Team Memory 参数

参数	值
单文件上限	250 KB
PUT 请求体上限	200 KB
Secret 扫描规则	40+ Gitleaks 规则

11.6 系统规模

指标	值
源码版本	v2.1.88
文件数量	1,906
代码行数	512,000+
System Prompt 行数	~914
内建工具数量	~43
工具定义文件	~29,000 行
运行时	Bun (TypeScript)
UI 框架	React + Ink

十二、可迁移的设计原则

从 Claude Code 的记忆系统中，可以提炼出以下通用原则，适用于任何需要长期记忆的 AI Agent 系统：

原则一：规则、索引、正文不要混在同一个注入通道

行为规则走 **system prompt**，索引走首条消息，正文走动态附件。

混在一起会导致缓存失效、成本失控和职责模糊。

原则二：长期记忆要按主题拆文件，不要堆单文件

一个主题一个文件，才能被高效检索、去重、替换和纠偏。

单文件模式在记忆积累后无法定向更新，也无法做选择性召回。

原则三：索引和正文要分层

先用摘要做选择，再按需读正文。

这是控制 token 成本的核心——检索先看摘要，不先把全文都塞进模型。

原则四：即时写回与延迟整固要并存

快路径负责捕获新信号，慢路径负责降噪纠偏。

只有即时写回会积噪；只有延迟整固会丢时效。两者并存才能兼顾连续性和可靠性。

原则五：记忆必须自带防漂移机制

任何会老化的知识，如果不主动提醒"这只是过去时的快照"，就迟早会变成误导。

过时但说得很像真的信息，比完全没有信息更危险。

原则六：权限边界不能省

让模型能写长期记忆，不等于让它任意写整个工作区。

记忆维护是高价值高风险操作，必须约束到专用目录和受限权限。

原则七：连续性不能只靠 transcript

对话记录是原料，不是最终记忆结构。

Claude Code 把连续性拆成 transcript、session summary、auto memory、active recall、autodream、team memory 六类工件协同工作。这些工件一起运转，才构成稳定连续性。

原则八：成熟记忆系统的关键不是"记得多"，而是"在对的时候取回对的内容"

检索能力比堆更多笔记更重要。

Active Recall 机制表明：记忆系统的核心价值不在存储规模，而在运行时的精准召回。

附录

附录 A：源码文件路径索引

文件路径	功能说明
src/query.ts	queryLoop 核心逻辑
src/constants/prompts.ts	System Prompt 组装 (getSystemPrompt)
src/memdir/memdir.ts	记忆系统核心 (memdir)
src/memdir/memoryTypes.ts	记忆类型定义 (TYPES_SECTION_COMBINED)
src/memdir/memoryAge.ts	Freshness 计算 (memoryAgeDays)
src/services/extractMemories/extractMemories.ts	回合后自动提炼
src/services/autoDream/autoDream.ts	跨 session 整固 (含 .consolidate-lock)
src/services/autoDream/consolidationPrompt.ts	AutoDream 四阶段 prompt
src/services/compact/autoCompact.ts	上下文压缩阈值 (THRESHOLDS)
src/utils/attachments.ts	记忆注入逻辑 (memoryHeader)
src/utils/undercover.ts	Undercover Mode 逻辑

附录 B：参考资料来源列表

来源	标题/主题	日期	语言
叉烧 (微信公众号)	Claude Code 源码阅读万字解析记忆系统	2026-03	中文
青稞 (微信公众号)	ClaudeCode 运行机制与 Memory 模块	2026-03	中文
db0.ai	How Claude Code Memory Works	2026-03-26	英文
db0.ai	Memory Consolidation vs AutoDream	2026-03-30	英文
ClawDecode (Avery Chai)	Source Analysis: Dream Mode	2026-03-31	英文
Ars Technica	Source Leak: AutoDream and Memory	2026-04-01	英文
dev.to (cristian)	Memory Architecture Nobody Documented	2026-04-01	英文
dev.to (ishaaan)	Every System Explained	2026-04-01	英文
dev.to (shekharp)	Memory Architecture and MCP	2026-04-02	英文

如果只记一句话：Claude Code 的 Memory 不是"给模型多塞一点用户笔记"，而是把规则、索引、动态召回、会话摘要、即时写回、离线巩固、共享同步和过期治理**一起工程化**，从而把连续性从聊天记录里拆出来，做成一个真正的 runtime 子系统。