

# Claude Code 源代码深度分析报告

来自：AI超元域

我的B站频道：<https://space.bilibili.com/3493277319825652>

项目：Claude Code CLI (Anthropic)

分析日期：2026-03-31

源码快照：2026年3月31日 (npm source map 泄露)

规模：约1,900个文件，512,000+行 TypeScript 代码

运行时：Bun | UI框架：React + Ink (深度定制fork) | CLI：Commander.js

## 目录

- 概述总结
- 整体架构
- 核心引擎
- 工具系统
- 权限系统
- 服务层
- UI 架构
- Bridge 桥接系统
- 支撑基础设施
- 安全性分析
- 设计模式与工程决策
- 结论

## 1. 概述总结

Claude Code 是 Anthropic 官方的 CLI 工具，用于在终端中与 Claude 交互。它是一个深度工程化的代理式开发者工具，能够编辑文件、执行命令、搜索代码库、以及协调多代理工作流。

核心技术亮点：

- 自定义 Ink fork — 包含 React reconciler、Yoga 布局引擎、双缓冲渲染、鼠标支持
- 5层错误恢复 — 核心查询循环中最大化会话存活率
- 43个工具 — 统一类型系统、逐工具权限模型、流式并行执行

- 多提供商 API — 支持 Anthropic 直连、AWS Bedrock、Azure Foundry、Google Vertex、Claude.ai
- 8种 MCP 传输协议 — 用于外部工具集成
- 编译时死代码消除 — 通过 Bun 的 `feature()` 标志清除15+个内部功能
- 纯 TypeScript 移植 — Yoga (C++)、nucleo/fzf (Rust)、syntect/bat (Rust)

## 2. 整体架构

### 2.1 顶层数据流

```
main.tsx (Commander.js CLI) --> REPL
  --> QueryEngine.submitMessage() --> query()
  --> callModel() --> StreamingToolExecutor
  --> Hooks --> Hooks
  --> Stop Hooks --> Hooks
```

### 2.2 目录结构

目录	文件数	用途
tools/	43个子目录	代理工具实现
commands/	~50个子目录	斜杠命令实现
components/	~140个文件	Ink UI 组件
hooks/	~85个文件	React Hooks
services/	~38个条目	外部服务集成
utils/	100+个文件	工具函数
bridge/	~30个文件	IDE 与远程控制桥接
constants/	~20个文件	配置常量
ink/	~90个文件	自定义 Ink 渲染引擎
state/	~8个文件	状态管理
skills/	~5个文件	技能系统
memdir/	~8个文件	持久化记忆
keybindings/	~14个文件	键盘绑定处理
vim/	~5个文件	Vim 模式模拟

### 2.3 技术栈

类别	技术
运行时	Bun

类别	技术
语言	TypeScript（严格模式）
终端 UI	React + Ink（深度定制 fork）
CLI 解析	Commander.js (extra-typings)
Schema 校验	Zod v4
代码搜索	ripgrep
协议	MCP SDK、LSP
API	Anthropic SDK
遥测	OpenTelemetry + gRPC
特性开关	GrowthBook
认证	OAuth 2.0、JWT、macOS Keychain

### 3. 核心引擎

#### 3.1 QueryEngine.ts（约46,000行）— LLM 会话引擎

`QueryEngine` 类管理整个会话的生命周期。每个会话一个实例，`submitMessage()` 是核心入口。

关键设计决策：

- AsyncGenerator 模式：`submitMessage()` 是 `async*` 生成器，流式输出 `SDKMessage`
- 提前持久化 transcript：用户消息在 API 调用前即写入，即使进程被杀也能恢复
- 权限拒绝追踪：包装 `canUseTool` 回调以拦截并上报拒绝
- 即发即忘 transcript：助手消息使用 `void recordTranscript()` 写入，避免阻塞

`submitMessage` 流程：

1. 提取配置，设置 CWD，确定模型和 thinking 配置
2. 获取系统提示词组件（含 coordinator 用户上下文）
3. 构建 `ProcessUserInputContext`，处理用户输入
4. 将用户消息持久化到 transcript（提前写入）
5. 处理孤立权限（每个引擎生命周期仅一次）
6. 产出 `buildSystemInitMessage`（工具、MCP 客户端、模型信息）
7. 进入 `query()` 循环，分发每种消息类型
8. 通过 `accumulateUsage` 追踪用量
9. 记录 transcript（助手消息即发即忘）
10. 检查预算限制、结构化输出重试次数、最大轮次
11. 产出最终结果消息（费用、用量、耗时）

3.2 query.ts（约1,730行） — Claude Code 的心脏

核心代理循环采用 `while(true)` 模式 + 显式 State 状态转换（非递归，避免长会话栈溢出）。

每轮迭代流程（14步）：

- 1. 技能发现预取 — 在模型流式输出期间并行运行
- 2. 查询链追踪 — 创建/递增 chainId/depth
- 3. 工具结果预算 — applyToolResultBudget 强制每条消息聚合大小
- 4. Snip 压缩 — 修剪过期历史（特性门控）
- 5. 微压缩 — 应用细粒度上下文编辑
- 6. 上下文折叠 — 投影折叠后的上下文视图（特性门控）
- 7. 自动压缩 — token 计数过高时主动压缩
- 8. 模型流式调用 — 调用 deps.callModel()，处理流式事件
- 9. 工具执行 — StreamingToolExecutor 在流式输出期间即开始执行
- 10. 错误处理 — 多层恢复级联
- 11. Stop Hooks — 后采样钩子、记忆提取
- 12. 工具结果 — 收集并作为附件注入
- 13. 工具池刷新 — 为新连接的 MCP 服务器刷新工具池
- 14. 最大轮次检查 — 超出则产出 max\_turns\_reached

5层错误恢复机制：

层级	策略	触发条件
第1层	上下文折叠排水	Prompt 过长（最廉价）
第2层	响应式压缩	完整会话摘要
第3层	最大输出升级	8K -> 64K tokens
第4层	多轮恢复	推动消息（最多3次）
第5层	模型回退	切换到备用模型

3.3 特性开关架构

机制	时机	方式
<code>feature()</code> (bun:bundle)	编译时	打包器死代码消除
Statsig/环境变量门控	运行时	每次查询快照一次

4. 工具系统

4.1 工具类型系统（Tool.ts，约29,000行）

每个工具遵循 `Tool<Input, Output, Progress>` 接口。 `buildTool()` 工厂函数应用安全默认值（fail-closed：`isConcurrencySafe=false`，`isReadOnly=false`）。

4.2 核心工具清单

工具	文件数	复杂度
BashTool	18	最高 — Shell 安全、沙盒、ML 分类器
AgentTool	17	第二高 — 多模式代理生成
FileEditTool	6	最严格的输入验证
FileReadTool	5	支持5种文件类型，token 限制
FileWriteTool	3	必须先读后写
GlobTool	3	基于 ripgrep 的模式匹配
GrepTool	3	3种输出模式，分页支持
SkillTool	4	技能到代理的桥接
MCPTool	4	运行时模板覆写

4.3 工具执行生命周期

```
1. Zod Schema 验证 (safeParse)
2. 验证 (tool.validateInput)
3. 验证 (工具元数据)
4. PreToolUse 钩子 (工具)
5. 钩子 (hooks -> tool.checkPermissions -> 钩子)
6. 调用 (tool.call)
7. PostToolUse 钩子
8. 验证 (工具元数据)
```

4.4 BashTool 安全层级

层级	组件	功能
第1层	bashSecurity.ts	命令注入、Zsh 漏洞、IFS 注入、Unicode 混淆
第2层	readOnlyValidation.ts	git/find/grep/ls 只读命令白名单
第3层	bashPermissions.ts	模式验证、路径约束、复合命令拆分
第4层	shouldUseSandbox.ts	macOS 沙盒判定
第5层	ML 分类器	基于推理的命令风险分类

4.5 FileEditTool 验证链（12步）

- 1. 团队记忆文件中的秘密检测
- 2. old\_string != new\_string 检查
- 3. 拒绝规则检查 + UNC 路径保护
- 4. 文件大小限制（1 GiB）

5. 编码检测 (UTF-8 / UTF-16LE)
  6. 文件存在性检查 (含相似文件建议)
  7. Jupyter notebook 重定向
  8. 必须先读后写强制执行
  9. 过期检测 (含内容比较回退)
  10. 引号规范化 (处理弯引号)
  11. 多重匹配检测 (除非 replace\_all 否则拒绝)
  12. 设置文件编辑验证
- 

## 5. 权限系统

### 5.1 三层处理器级联

```
权限 (hasPermissionsToUseTool)
  |-- allow --> 允许
  |-- deny  --> 拒绝
  |-- ask   --> 询问:
Coordinator 权限 (权限)
  --> 询问 (询问)
  --> Bash 权限 (询问)
Swarm Worker 权限
  --> 询问 --> 权限 leader
询问 (权限)
  --> React ToolUseConfirm 权限
  --> 200ms 询问
  --> Bridge/询问
```

### 5.2 权限上下文

`ToolPermissionContext` (DeepImmutable 不可变) :

- `mode` : default | auto | plan | acceptEdits | bypassPermissions | dontAsk | bubble
- `alwaysAllowRules` / `alwaysDenyRules` / `alwaysAskRules`
- `additionalWorkingDirectories` (额外工作目录)
- `shouldAvoidPermissionPrompts` (用于后台代理)

### 5.3 受保护路径

`.gitconfig`、`.bashrc`、`.zshrc`、`.profile`、`.mcp.json`、`.claude.json`、`.git/`、`.vscode/`、`.idea/`、`.claude/`

---

## 6. 服务层

### 6.1 API 服务 — 多提供商工厂

提供商	SDK	说明
Anthropic 直连	@anthropic-ai/sdk	默认，OAuth 或 API key
AWS Bedrock	@anthropic-ai/bedrock-sdk	动态导入
Azure Foundry	@anthropic-ai/foundry-sdk	动态导入
Google Vertex AI	@anthropic-ai/vertex-sdk	动态导入
Claude.ai	OAuth bearer	订阅用户访问

重试策略：

- 指数退避 + 抖动
- 529 仅对前台查询重试，防止级联放大
- 无人值守持久重试模式（429/529 无限重试）
- AWS/GCP 凭证刷新集成到重试循环中

6.2 MCP 服务

8种传输类型：stdio、sse、sse-ide、ws、ws-ide、http、sdk、claudeai-proxy

配置作用域：local → user → project → dynamic → enterprise → claudeai → managed

6.3 上下文压缩 — 三层

层级	触发条件	机制
微压缩	每轮	对单条消息进行细粒度缓存编辑
自动压缩	tokens > contextWindow - 13K	Fork 代理共享 prompt cache
完整压缩	手动 /compact	完整会话摘要

6.4 分析与认证

- 零依赖入口：事件排队直到 attachAnalyticsSink() 被调用
- PII 路由：\_PROTO\_\* 前缀 → 特权 BigQuery 列
- GrowthBook：磁盘缓存过期读取 + 远程刷新
- OAuth：PKCE (\$256) 授权码流程，支持 Claude.ai 和 Console 登录

7. UI 架构

7.1 渲染栈 — 自定义 Ink Fork (90+个文件)

组件	技术
React Reconciler	react-reconciler (Concurrent 模式)
布局引擎	Yoga (纯 TypeScript 移植)

组件	技术
渲染	双缓冲帧、基于差异的终端写入
输入	自定义 parse-keypress、鼠标 hit-testing
文本选择	Alt-screen 模式 + 搜索覆盖层

7.2 组件层级

```
cli.tsx
  main.tsx (窗口: MDM, Keychain, API 窗口)
    窗口 (Onboarding -> Trust -> MCP 窗口)
      App.tsx > FpsMetricsProvider > StatsProvider
        > AppStateProvider > MailboxProvider > VoiceProvider
          > REPL.tsx (~5,000 行 -- 窗口)
```

7.3 状态管理 — 极简自研（仅35行）

不使用 Redux 或 Zustand。createStore<T>() 返回 getState/setState/subscribe。AppState 约570行类型定义，通过 useAppState(selector) + useSyncExternalStore 实现最小化重渲染。

7.4 输入处理管线

```
窗口 stdin -> Ink parse-keypress
-> 窗口 (窗口, chord, 1000ms 窗口)
-> Vim 窗口 (normal/insert, d/c/y, h/l/w/b/e/$)
-> 窗口 (kill ring, yank, history, multiline)
```

8. Bridge 桥接系统

8.1 REPL Bridge（本地 CLI → Claude.ai）

版本	读取传输	写入传输
v1	WebSocket	HTTP POST 到 Session-Ingress
v2	SSE	CCR v2 /worker/* 端点

8.2 Remote Bridge

模式	隔离方式
single-session	在 cwd 中的单个会话
worktree	每个会话使用独立的 git worktree
same-dir	所有会话共享同一目录

8.3 消息协议与 IDE 集成

进站：control\_response（权限决策）、control\_request（服务器命令）、user 消息



出站：仅用户/助手消息，虚拟消息被过滤

IDE 通过 MCP sse-ide / ws-ide 传输连接，支持代码选择转发、文件 @提及、差异显示、权限回调。

## 9. 支撑基础设施

### 9.1 持久化记忆

类型	用途	示例
user	用户画像、偏好	深度 Go 经验，React 新手
feedback	行为指导	集成测试必须使用真实数据库
project	在进行的工作上下文	合并冻结从 2026-03-05 开始
reference	外部系统指针	Pipeline bugs 在 Linear 项目 INGEST 中

### 9.2 纯 TypeScript 原生模块移植

模块	替代目标	用途
yoga-layout	Meta Yoga (C++)	Ink 的 Flexbox 布局
file-index	nucleo (Rust)	fzf-v2 模糊搜索，270K+ 文件异步索引
color-diff	syntect+bat (Rust)	highlight.js 语法高亮 + 词级差异

### 9.3 其他支撑系统

- CCR 上游代理：TCP-to-WebSocket CONNECT 中继，prctl 防 ptrace，Protobuf 分帧，全部 fail-open
- 伴侣精灵：18种物种、6种眼睛、8种帽子、5种稀有度，Mulberry32 PRNG 确定性生成
- 设置子系统：policySettings > projectSettings > localSettings > userSettings，安全字段受限
- Swarm 团队系统：InProcess/Tmux/iTerm 后端，Leader-Worker 权限委托
- 配置迁移：11个幂等迁移（模型别名、权限模式、设置路径）
- 优雅关闭：SIGINT/SIGTERM/SIGHUP 处理，30秒孤立检测，安全强制退出

## 10. 安全性分析

### 10.1 BashTool 安全（纵深防御）

层级	缓解的威胁
Shell 安全分析	命令注入（\$()、反引号）、进程替换
Zsh 专项防护	zmodload、syswrite、ztcp 利用

层级	缓解的威胁
输入消毒	IFS 注入、花括号展开、Unicode 空白字符混淆
NTLM 保护	UNC 路径在所有文件系统工具上阻止
只读白名单	Git 读取命令、docker inspect、gh 读取命令
macOS 沙盒	按命令可配置的沙盒
ML 分类器	基于推理的命令风险分类
路径验证	拒绝项目外的写入、危险文件路径

10.2 全栈安全措施

- 文件系统：必须先读后写、过期检测、设备路径阻止、UNC 保护、秘密检测
- 凭证：prctl 防 ptrace、token 仅堆内存、OAuth PKCE、JWT、macOS Keychain
- Prompt 注入：工具结果预算限制、结构化输出钩子、消息去重防重放
- PII：类型安全标记防误记、\_PROTO\_\* 前缀路由到特权存储

11. 设计模式与工程决策

11.1 架构模式

模式	应用位置	目的
AsyncGenerator	submitMessage()、query()	流式消息消费
Fail-closed 默认	buildTool()	安全优先的工具注册
依赖注入	QueryDeps、ToolUseContext	可测试性
编译时 DCE	feature() + bun:bundle	剥离内部功能
品牌类型	SessionId、AgentId	防止 ID 混淆
零依赖入口	Analytics index.ts	打破导入循环
双缓冲渲染	Ink Output 系统	无闪烁终端更新
Prompt 缓存稳定性	assembleToolPool() 排序	防缓存失效

11.2 性能优化

优化	影响
启动时并行预取	MDM、Keychain、API 预连接并行于模块评估
流式工具执行	模型输出期间即执行工具，降低延迟
惰性导入	重量级模块首次使用时加载

优化	影响
虚拟滚动	仅挂载可见范围内的消息
LRU 缓存 JSON 解析	50条目缓存
即发即忘 transcript	助手消息非阻塞写入
Blit 优化	仅写入变化的终端单元格

### 11.3 可靠性模式

模式	应用位置
断路器	自动压缩（最多3次连续失败）
Fail-open 设计	代理、语音、记忆、分析
指数退避 + 抖动	API 重试
级联放大防护	529 仅对前台查询重试
孤立进程检测	30秒间隔进程监控
优雅降级	所有支撑系统不致命

## 12. 结论

### 12.1 工程质量评估

- 架构：查询引擎、工具系统、权限、UI 和服务之间有清晰的分离。QueryDeps 注入模式和 ToolUseContext 提供了可测试、可组合的抽象。
- 性能：在每一层都有激进的并行化 — 启动预取、流式工具执行、并发工具批次、虚拟滚动和基于差异的渲染。
- 安全：Shell 执行的纵深防御（6+层）、fail-closed 默认值、NTLM 保护、过期追踪和 PII 安全分析。
- 可靠性：核心循环5层错误恢复、断路器、fail-open 支撑系统、带孤立检测的优雅关闭。
- 开发者体验：自定义 Ink fork 提供桌面级终端 UI，支持鼠标、Vim 和键盘 chord。

### 12.2 突出技术成就

1. 纯 TypeScript Yoga 移植 — 消除原生二进制依赖，实现跨平台布局
2. 流式工具执行 — 重叠模型推理和工具执行以降低延迟
3. 编译时特性门控 — 通过 Bun 的 feature() 系统实现内部/外部构建分离
4. 多提供商 API 抽象 — 跨5个云提供商的统一接口
5. 35行状态管理 — 证明复杂状态可以不依赖框架来管理

### 12.3 规模指标

指标	数值
总文件数	约1,900
总代码行数	512,000+
工具实现	43个
斜杠命令	约50个
UI 组件	约140个
React Hooks	约85个
服务模块	约38个
MCP 传输类型	8种
特性开关	15+（编译时）
伴侣精灵物种	18种

本报告基于2026年3月31日的源码快照生成。仅用于教育和安全研究目的。

来自：AI超元域 | [B站频道](#)