

tokenscope

看清 AI 编程工具的 token
到底花在了哪里

1%

一次请求里，你自己打的字只占这么多

39%

模型回答只占 8% 的量，却吃掉这么多钱

5×

读一个文件，因为之后聊了 5 轮，被收费 5 次

目录

1	它解决什么问题	为什么日志看不出来，代理能
2	安装	一条命令，可选精确分词
3	60 秒体验	不联网、不花钱，先看有没有用
4	接入你的 agent	四种场景，照抄命令即可
5	看结果	终端报表与浏览器看板
5b	还能用多久	额度、速率与主动预警
6	配置单价	让成本一列有意义
7	隐私与凭证	工具在你磁盘上写了什么
8	命令速查	全部子命令与常用参数
9	排错	最常见的四类问题

1 它解决什么问题

你付给 AI 编程工具的钱，绝大部分花在你看不见的地方。一次请求里，你输入的需求 大约只占 **1%**，剩下的是工具说明、历史对话、读进来的文件 —— 全都是框架替你发的，而且每一轮重发一次。

三个具体发现

你打的字只占 1%

其余 99% 是 system prompt、工具定义、历史对话和工具返回内容。

输出量少但贵

模型回答只占 8% 的 token，却花掉 39% 的钱 —— 输出单价是输入的 5 倍。

同一份内容被反复收费

读一个文件花 657 token，但读完又聊了 5 轮，实际付了 3300。

第三条最有用，因为它推翻了一个常见判断：多数人以为省钱要换便宜模型，实际上 及时关掉用不上的文件、或者打开缓存，效果大得多。

为什么不解析日志

各框架的日志格式互不相同，而且多数只记最终 token 总数，不记构成。但所有 agent 都得走 HTTP 调模型 —— 在中间插一个反向代理，一套实现覆盖全部框架，还能拿到 **完整请求体**，这才有可能拆解构成。

代理是完全透明的

状态码、响应头、响应体字节都不改动，SSE 流逐块实时透传不缓冲。统计走旁路，解析失败也不影响 agent 正常工作。这一点由测试保证，包括「流式响应字节完全一致」。

支持哪些工具

任何认 base URL 的 agent 都可以，已验证：Claude Code、Aider、Cline、Continue、Kiro CLI。上游可以是云端 API（Anthropic、OpenAI、公司自建网关）也可以是本地 Ollama / LM Studio。多个 agent 可以同时统计，报表里自动分开。

2 安装

需要 Python 3.10 或更高版本。建议用虚拟环境，避免污染系统环境。

```
python3 -m venv .venv
. .venv/bin/activate
pip install -e .
```

装完确认一下：

```
tokenscope --version
```

可选：精确分词

```
pip install tiktoken
```

不装也能用

估算只决定各成分的**相对占比**，绝对值一律以上游返回的真实 usage 为准，再等比缩放。所以启发式带来的系统性偏差在占比上基本抵消。装了 tiktoken 精度更高，启动时面板会显示当前用的是哪种。

让命令全局可用

虚拟环境里的命令默认不在系统 PATH 里。如果不想每次都激活环境，做个软链接：

```
mkdir -p ~/.local/bin
ln -sf "$PWD/.venv/bin/tokenscope" ~/.local/bin/tokenscope
```

注意

软链接指向项目目录里的虚拟环境，所以别把项目目录挪走或删除，否则命令会失效。另外确认 `~/.local/bin` 在你的 PATH 里（`echo $PATH` 看一下）。

语言

所有输出支持中英文，默认按 `TOKENSCOPE_LANG` 或系统 locale 自动判断：

```
tokenscope report --lang zh      # 单次指定
export TOKENSCOPE_LANG=zh        # 对所有命令生效
```

3 60 秒体验

在接入自己的真实用量之前，先用内置样例数据看一遍报表。**不联网、不需要 API key、不花钱**，跑完自动清理。

```
tokenscope demo
```

它会依次展示六张表，每张前面配一句大白话说明它在回答什么问题：

第 1 张 · 总览	这段时间总共花了多少，分别花在哪个工具上
第 2 张 · 成分拆分	钱到底被什么占掉了。这是整个工具的核心
第 3 张 · 工具开销	哪个工具动作最费钱，变动开销与固定开销分开
第 4 张 · 重发放大	最反直觉的一张：同一份内容被反复收费
第 5 张 · 会话列表	这笔钱花在哪件具体的事上
第 6 张 · 逐轮明细	单件事的每一轮请求，含缓存生效的说明

想留着样例数据自己翻，或者用它试试浏览器看板：

```
tokenscope demo --keep ~/ts-demo.db
tokenscope report --db ~/ts-demo.db --view amplify
tokenscope web --db ~/ts-demo.db
```

4 接入你的 agent

原理是把 agent 的 base URL 指向本机代理，代理再转发到原来的上游。链路变成：

```
agent → tokenscope (本机 :8787) → 原来的模型服务
```

场景 A：Claude Code 直连官方 API

1 改配置，让它走代理

```
tokenscope hook claude --agent claude-code
```

这会改 `~/.claude.json` 里的 `env.ANTHROPIC_BASE_URL`，并把原上游记在 `~/.tokenscope/hooks.json`。凭证和模型名一个字不动。

2 启动记录（这个终端要一直开着）

```
tokenscope serve
```

不用传 `--upstream`：hook 已经记住了原上游，serve 会自动读，面板上会标明「来自 hook 记录的原上游」。

3 另开一个终端，照常干活

```
claude
```

不需要设任何环境变量。回看 serve 那个终端，每次请求都会打一行日志。

4 用完还原

```
tokenscope unhook claude
```

挂载期间代理必须开着

配置指向 `127.0.0.1:8787`，代理没启动的话 Claude Code 会连不上。不确定当前状态就跑 `tokenscope hook --status`。

场景 B：Claude Code 走公司自建网关

和场景 A 完全一样。`hook` 会自动读出你配置里当前的网关地址并记下来，`serve` 自动指向它。带路径前缀的网关（例如 `http://gw.corp/anthropic`）也支持。

为什么必须改配置文件

Claude Code 配置文件里 `env` 段的优先级高于 shell 环境变量。直接 `export ANTHROPIC_BASE_URL=...` 会被无声忽略——工具照常工作，但流量绕过代理，看起来就像「监控不生效」。这是最容易踩的坑。

场景 C：本地 Ollama

```
# 终端一
tokenscope serve --preset ollama

# 终端二
export ANTHROPIC_BASE_URL=http://127.0.0.1:8787/a/claude-code
export ANTHROPIC_AUTH_TOKEN=ollama
claude --model qwen3-coder
```

Ollama v0.14.0 起原生支持 Anthropic Messages API，所以 Claude Code 可以直接对话，不需要额外的翻译代理。

场景 D：Aider 及其他 OpenAI 兼容工具

```
# 终端一
tokenscope serve --preset ollama

# 终端二
export OPENAI_API_BASE=http://127.0.0.1:8787/a/aider/v1
# 或者 Aider 直连 Ollama 时
export OLLAMA_API_BASE=http://127.0.0.1:8787/a/aider
aider --model ollama_chat/qwen3-coder
```

所有框架的接入命令都可以直接打印出来：

```
tokenscope env
```

关于 /a/<名字> 前缀

URL 里的 `/a/<名字>` 决定这批流量在报表里归到哪个 agent 名下，转发给上游时会被剥掉。不带前缀的归到 `default`，也可以用请求头 `X-Tokenscope-Agent` 指定。

多个上游混用

默认走云端、某个 agent 走本地：

```
tokenscope serve --preset anthropic --route aider=http://127.0.0.1:11434
```

5 看结果

终端

```
tokenscope report                # 全部四个视角
tokenscope report --view components # 只看钱被什么占掉了
tokenscope report --view amplify   # 只看重发放大
tokenscope report --agent claude-code # 只看某个 agent
tokenscope report --days 1         # 只看最近一天
tokenscope report --json            # 机器可读

tokenscope sessions              # 你干过哪些事，各花多少
tokenscope session a3c530d6      # 单件事的逐轮明细（前缀即可）
```

表格按终端宽度自适应：窄屏下自动收起解释性列，不折行。

浏览器看板

```
tokenscope web
```

默认 `http://127.0.0.1:8788`，自动打开浏览器。看板把「token 占比 / 成本占比」两条并排画出来，点任意会话展开逐轮明细和 prompt 增长曲线。右上角可以按 agent 和时间筛选、切换中英文、导出 JSON。会话支持深链接（`#session=<id>`），链接可以分享。

看板不引任何外部资源

没有 CDN、没有 web 字体、没有分析脚本，图表是手写 SVG，整个看板是一个自包含的 HTML 文件。断网可用，也不会为了看自己的账单而向第三方发请求。

看板没有鉴权

默认只监听 `127.0.0.1`。谁能访问端口谁就能看到你的任务摘要和成本数据。绑到 `0.0.0.0` 需要显式加 `-host`，届时会打印安全警告。

数据存在哪

`~/.tokenscope/usage.db` 主库 (SQLite)。`-wal` / `-shm` 是配套文件，拷走要一起拷

`~/.tokenscope/hooks.json` hook 状态：配置路径、原 base URL

`~/.tokenscope/pricing.json` 你自己填的模型单价

改位置：`--db /path/to.db` 单次指定，或 `export TOKENSCOPE_HOME=~/.somewhere` 改整个目录。清空：`tokenscope reset --yes`。

5b 还能用多久

前面几章回答「钱花在哪」，这一章回答「还能干多久」。数据来源分三层，你属于哪一层取决于你的上游给不给你限流信息。

三层来源

- | | |
|------------|---|
| ① 上游回报（权威） | 订阅制 Pro/Max 直连官方时，响应头里有 5 小时窗口和 7 天窗口的使用率与重置时间；按量计费则有每分钟的剩余请求数和 token 数 |
| ② 本地声明额度 | 自建网关通常会把限流头剥掉。这时你自己声明额度，tokenscope 用它的记录算滚动窗口内的消耗 |
| ③ 仅趋势 | 连额度都不知道时，至少能给消耗速率和「再聊 5 轮 prompt 会涨到多少」 |

订阅制有个容易搞错的地方

5 小时窗口和 7 天窗口必须都没满才能继续，但哪个说话算数由响应头里的 `representative-claim` 字段指定。tokenscope 按它判断，不自己拍——忽略这个字段会导致 5 小时窗口还剩 98% 的人被误挡。

终端实时状态条

```
tokenscope live          # 常驻刷新
tokenscope live --once    # 只打印一次
tokenscope live --interval 5 # 5 秒刷新一次
```

它会显示每个窗口的已用比例、剩余、重置倒计时，以及当前消耗速率、按此速率撞上上限的预计时间、当前任务的轮次和累计成本。

声明自己的额度

格式是 `<窗口>=<额度>`，serve 和 live 都接受：

```
tokenscope serve --limit 5h=200k # 5 小时 20 万 token
tokenscope serve --limit 1d=$5    # 一天 5 美元
tokenscope live  --limit 7d=10M   # 7 天 1000 万 token
```

滚动窗口的起点是窗口内第一次请求的时间，不是整点——这和 Anthropic 5 小时窗口的行为一致。

主动预警

serve 运行时会在三种情况下主动打一行提示，不需要你去查：

- 权威窗口剩余低于 20%（低于 8% 升级为严重）
- 按当前速率 20 分钟内会撞上上限（6 分钟内升级为严重）
- 会话的 prompt 膨胀到再 5 轮就会涨 1.5 倍以上

同一条提醒 5 分钟内只说一次。提醒太密等于没有提醒。不想要就加 `--no-alerts`。

速率需要足够的样本

agent 常在一两秒内连发好几个请求。用 1.5 秒的样本外推「每分钟」会得到 几百万 token 这种荒谬数字，所以样本跨度不足 20 秒时不显示速率 —— 宁可不给，也不给一个会让你做错决定的数。

看板上的实时面板

浏览器看板顶部有同一份数据的图形版，每 3 秒自动刷新（页面切到后台时暂停）。额度条按剩余比例变色，权威窗口有标记。

6 配置单价

内置单价表只是**参考值**，随时可能过期。报表会明确标出哪些模型用的是内置默认值、哪些完全没有单价（按 0 计），不会假装免费。

```
tokenscope pricing          # 生成 ~/.tokenscope/pricing.json
tokenscope pricing --path    # 只打印路径
```

在文件里按模型名前缀加条目，单位是**美元 / 每百万 token**：

```
{
  "your-org/your-model": {
    "input": 3.0,
    "output": 15.0,
    "cache_read": 0.3,
    "cache_write": 3.75
  }
}
```

匹配时长前缀优先，所以 `claude-3-5-sonnet` 不会被 `claude` 抢先。

本地模型单价为 0，但统计依然有意义

token 数直接决定显存占用、上下文是否溢出、以及推理耗时。报表里「成本」一列 只反映云端调用，本地部分请看 token 和延迟。

预算护栏

```
tokenscope serve --budget-usd 2.0          # 超了告警
tokenscope serve --budget-usd 2.0 --block-over-budget  # 超了拦截
```

默认只告警。 `--block-over-budget` 会在单会话花超之后直接 返回 429，agent 侧会看到一个错误 —— 确认你能接受这种打断方式再开。

7 隐私与凭证

代理在链路上必然看得见你的 token，这是代理的本质。但它不解析、不记录、不落盘，也不发往任何第三方。下面是可以自己验证的方式。

一条命令自查

```
tokenscope audit --self-test
```

它往代理里注入一个假 token 走完整流程，再回头逐项检查这个 token 有没有出现在 数据库、日志、报表输出、错误响应里，并确认只有一个出站目标。全程本地不联网，每一项都会说明「怎么查的」。

看磁盘上写了什么

```
tokenscope audit
```

列出工具写过的每个文件、权限、以及是否含凭证或 prompt 片段。权限有问题的行会标红。同时列出**代理连接过的全部地址**——每个出站请求都会登记，包括失败的，所以这是完整清单。

关键事实

不复制你的凭证	hook 默认不备份配置文件。写回前会校验「改动恰好只有 base URL 一处」，不符就拒绝写入
不放松文件权限	改写配置时显式继承原权限；库文件和状态文件都是 0600
不存消息正文	只存 token 计数和成分。唯一的例外是任务摘要（首条消息前 70 字），用 <code>--no-titles</code> 可关
只有一个出口	唯一的对外连接是你配置的上游

自检的局限

自检是本程序自己跑的，一个恶意版本可以让它说谎。它能证明「这份代码在这条路径上 没有泄漏」，也能检测出安装被篡改，但不能替代读代码或抓包。要更强的保证，读一遍 `tokenscope/proxy.py`（约 350 行），或者抓包对比进出流量。

不想让 prompt 片段落盘

```
tokenscope serve --no-titles
```

关掉之后报表只显示 session hash，其余统计不受影响。

8 命令速查

命令	作用
tokenscope demo	零配置体验，用样例数据看一遍报表
tokenscope serve	启动统计代理
tokenscope web	打开浏览器看板
tokenscope live	终端实时状态条：剩多少额度、还能撑多久
tokenscope hook <目标>	改写 agent 配置使其走代理（可逆）
tokenscope hook --status	查看当前挂载状态
tokenscope unhook [目标]	还原配置，省略目标则全部还原
tokenscope report	综合报表
tokenscope sessions	会话列表
tokenscope session <前缀>	单会话逐轮明细
tokenscope audit	列出落盘文件与出站地址
tokenscope audit --self-test	凭证泄漏自查
tokenscope env	打印各框架的接入命令
tokenscope pricing	生成可编辑的单价文件
tokenscope reset --yes	清空统计数据

serve 常用参数

参数	说明
--preset	上游预设：anthropic / ollama / openai / lmstudio
--upstream URL	直接指定上游地址
--route AGENT=URL	为特定 agent 指定上游，可重复
--port N	监听端口，默认 8787
--budget-usd N	单会话预算上限
--block-over-budget	超预算时拦截而非仅告警
--limit SPEC	本地声明额度，例 5h=200k、1d=\$5

--no-alerts	不主动打印额度与增长预警
--no-titles	不记录任务摘要
--quiet	不打印逐请求日志
--db PATH	数据库路径

全局参数

参数	说明
--lang zh en	输出语言，放子命令前后都可以
--version	版本号

环境变量

变量	说明
TOKENSCOPE_LANG	输出语言，zh 或 en
TOKENSCOPE_HOME	数据与配置目录，默认 ~/.tokenscope

9 排错

命令不存在: tokenscope: command not found

命令装在虚拟环境里，没进系统 PATH。激活环境（`. .venv/bin/activate`）或者按第 2 章做个软链接。

agent 能用，但报表是空的

说明请求没走代理，这是最常见的问题。按顺序查：

- 代理还开着吗：`curl -s http://127.0.0.1:8787/__tokenscope/health` 应返回 `{"ok":true,...}`
- 挂载生效了吗：`tokenscope hook --status`
- 如果你是手动设环境变量：确认是在**同一个终端**里 export 的，并且该 agent 的配置文件没有覆盖它（Claude Code 就会覆盖）
- 看的是同一个库吗：serve 和 report 如果一个带了 `--db` 一个没带，数据就在两个地方

agent 连不上 / 报连接错误

大概率是配置还挂着但代理没开。要么启动 `tokenscope serve`，要么 `tokenscope unhook claude` 还原。

成本一列全是 \$0

模型不在内置单价表里。报表会标注「未知单价，成本按 0 计」。按第 6 章填单价即可，token 和缓存统计不受影响。

改了代码但行为没变

serve 是常驻进程，改完代码要重启它才会生效。

自己验证整条链路

不烧真实额度就能确认装好了：

```
python scripts/simulate_agent.py # 假上游 + 代理 + 两个 agent 的多轮会话
python scripts/showcase.py      # 直接看所有报表视角
pytest -q                       # 全部测试
```