

2026.2.26 建造你的贾维斯 (claude code hook)

一、Hook是什么？

Hooks are user-defined shell commands or LLM prompts that execute automatically at specific points in Claude Code's lifecycle. Use this reference to look up event schemas, configuration options, JSON input/output formats, and advanced features like async hooks and MCP tool hooks. If you're setting up hooks for the first time, start with the [guide](#) instead.

钩子是用户自定义的 shell 命令或 LLM 提示符，会在 Claude Code 生命周期的特定节点自动执行。本参考文档可帮助您查找事件模式、配置选项、JSON 输入/输出格式以及异步钩子和 MCP 工具钩子等高级功能。

为什么叫hook？

hook来自英文钩子,可以想象一条正在运行的流水线（程序执行流程），在某个节点挂上一个钩子，当流程经过这里时，钩子就会钩住流程，先执行相关的代码，再放行。

有无hook的流程对比

正常流程： 用户发消息 ——→ Claude 处理 ——→ 回复

有Hook： 用户发消息 ——→ [钩子触发，跑设置的脚本] ——→ Claude 处理 ——→ 回复

关于Hook这个概念很古老，最早出现在操作系统里（Windows API就有hook 机制，可以拦截键盘鼠标事件）。后来 git hooks、React 的useEffect 这类 hooks、Webpack 的 plugin hooks，都是同一个思路：在别人的流程里插入自己的逻辑，而不需要修改原始代码。其实就是运行这些软件，我不改变使用的软件的运行逻辑，在运行节点加入一些自己的判断，相当于对软件个性化的过程（开放扩展，关闭修改：像浏览器插件、VS Code插件等）**所以叫hook，就是钩进去的意思。**

名词解释：

事件（Event）： 某件事发生了，比如用户提交了一条消息

匹配器（Matcher）： 过滤条件，比如只在使用 Bash 工具时触发

处理器（Handler）： 写的脚本，接收 JSON 数据并执行操作

JSON数据： 全称JavaScript Object Notation，用文本来描述数据结构（键: 值）

代码块

```
1 json数据示例
2 {
```

```
3     "name": "Minnn",
4     "age": 18,
5     "hobby": ["learning"]
6 }
```

Hook 里用JSON，就是Claude Code把当前状态（谁触发了、用了什么工具、内容是什么）打包信息塞给脚本，脚本读完再决定怎么处理。

Async: Asynchronous缩写，a前缀在英文里表示不、非，synchronous = 同步，同时发生，步调一致。**类比**发消息 vs 打电话：和导师打电话是**同步**——说一句，等对方回，步调一致。和导师发消息是**异步**——发完继续干别的，对方什么时候回都行。

MCP: Model Context Protocol, **Model** = AI 模型 (Claude) **Context** = 上下文，模型工作时所处的环境和信息 **Protocol** = 协议，双方约定好的通信规则。**让AI模型和外部工具之间，用一套统一的规则来通信**。相当于usb接口，不管是鼠标、键盘还是U盘，只要符合 USB 协议，插上就能用

二、Hook的生命周期

Hooks fire at specific points during a Claude Code session. When an event fires and a matcher matches, Claude Code passes JSON context about the event to your hook handler. For command hooks, this arrives on stdin. Your handler can then inspect the input, take action, and optionally return a decision. Some events fire once per session, while others fire repeatedly inside the agentic loop:

HookLifecycle：钩子会在Claude Code会话期间的特定点触发。当**事件触发**且**匹配器**匹配时，Claude Code 会将事件的 JSON 上下文传递给钩子处理程序。对于命令钩子，此上下文通过标准输入 (stdin) 传递。相关处理程序随后可以**检查输入**、**执行操作**，并可选择**返回结果**。某些事件在每个会话中触发一次，而其他事件则会在代理循环中重复触发：

名词解释：

stdin: standard input 的缩写，拆开：标准的、默认的输出。计算机里有三条标准管道 (pipeline) 。

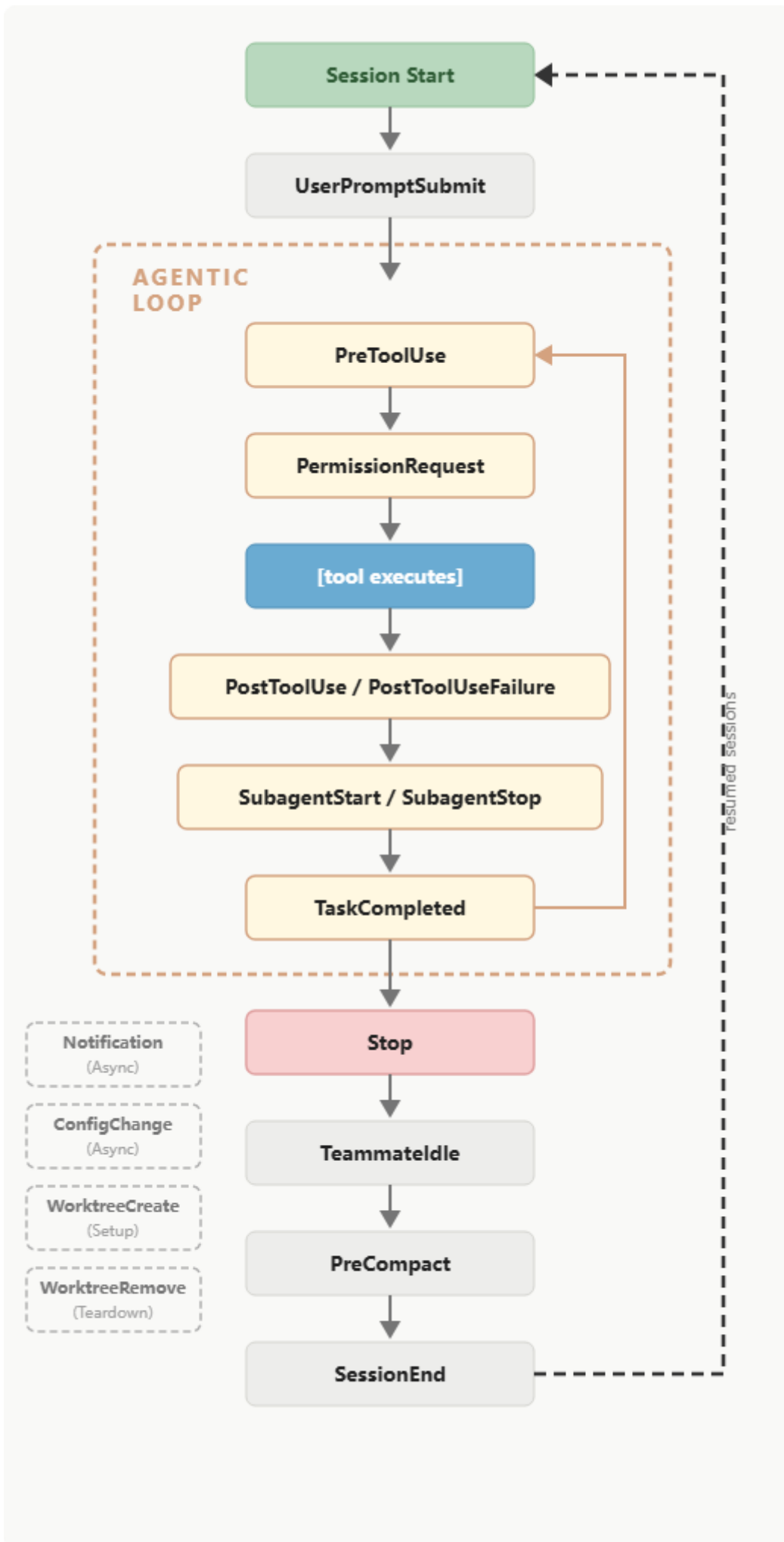
- stdin → 数据流进程序（标准输入）
- stdout → 数据流出程序（标准输出）
- stderr → 错误信息流出程序（标准错误）

类比生活中的收快递、寄快递、投诉快递。

在Hook中：Claude Code把JSON 数据 → 塞进脚本的stdin（收件窗口）→ 脚本读取 stdin → 拿到数据 → 处理 → 结果写到 stdout

两个程序之间用这套标准管道传递数据，不需要写文件、不需要网络，直接管道对管道。

cladue一次完整会话的生命周期流程



会话开始（SessionStart），用户发出一条指令（UserPromptSubmit），相当于下达一个任务，接下来进入任务循环（Agentic Loop），该任务要调用工具进行工具使用前检查（PreToolUse），如果工具太危险（例如对电脑中的文件进行删除：`rm -rf`）需要确认请求（PermissionRequest），工具开始执行，执行完成后汇报结果（PostToolUse），如果执行失败则记录错误（PostToolUseFailure）。如果这个任务太复杂，一个人搞不定，就临时调来专家小队（SubagentStart），小队完成后解散汇报（SubagentStop）。一个小目标完成，打勾（TaskCompleted）——然后循环回去，继续下一个工具调用，直到任务全部完成。跳出循环后：Claude 完成本次回复（Stop），队员进入待命状态（TeammateIdle），对话历史太长时自动整理压缩（PreCompact），最后会话结束（SessionEnd）。

Notification、ConfigChange、WorktreeCreate、WorktreeRemove都在主流程外因为它们触发的条件和Claude调用工具没有因果关系：

Notification → Claude通知时，随时触发

ConfigChange → 改配置时，随时触发

WorktreeCreate → 做git操作时，才触发

WorktreeRemove → 做git操作时，才触发。（Worktree = 工作树，同一份源代码，新建了三个文件夹，每个文件夹放一个版本，互不干扰）

不像PreToolUse / PostToolUse，必须跟着工具调用走，这四个是各自独立的触发条件，所以画在主流程外面。

The table below summarizes when each event fires. The [Hook events](#) section documents the full input schema and decision control options for each one.

| Event | When it fires |
|--------------------|--|
| SessionStart | When a session begins or resumes |
| UserPromptSubmit | When you submit a prompt, before Claude processes it |
| PreToolUse | Before a tool call executes. Can block it |
| PermissionRequest | When a permission dialog appears |
| PostToolUse | After a tool call succeeds |
| PostToolUseFailure | After a tool call fails |
| Notification | When Claude Code sends a notification |
| SubagentStart | When a subagent is spawned |
| SubagentStop | When a subagent finishes |
| Stop | When Claude finishes responding |
| TeammateIdle | When an <u>agent team</u> teammate is about to go idle |
| TaskCompleted | When a task is being marked as completed |
| ConfigChange | When a configuration file changes during a session |
| WorktreeCreate | When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior |
| WorktreeRemove | When a worktree is being removed, either at session exit or when a subagent finishes |
| PreCompact | Before context compaction |
| SessionEnd | When a session terminates |

【事件一览】

SessionStart - 会话开始或恢复时

UserPromptSubmit - 用户提交提示词，Claude处理之前

PreToolUse - 工具调用执行之前，可以阻止

PermissionRequest - 出现权限对话框时

PostToolUse - 工具调用成功之后

PostToolUseFailure - 工具调用失败之后

Notification - Claude Code 发送通知时

SubagentStart - 子代理被创建时

SubagentStop - 子代理完成时

Stop - Claude完成回复时

TeammateIdle - 团队成员即将进入空闲时

TaskCompleted - 任务被标记为完成时

ConfigChange - 配置文件在会话中变更时

WorktreeCreate - 创建工作树时，替代默认git行为

WorktreeRemove - 工作树被移除时

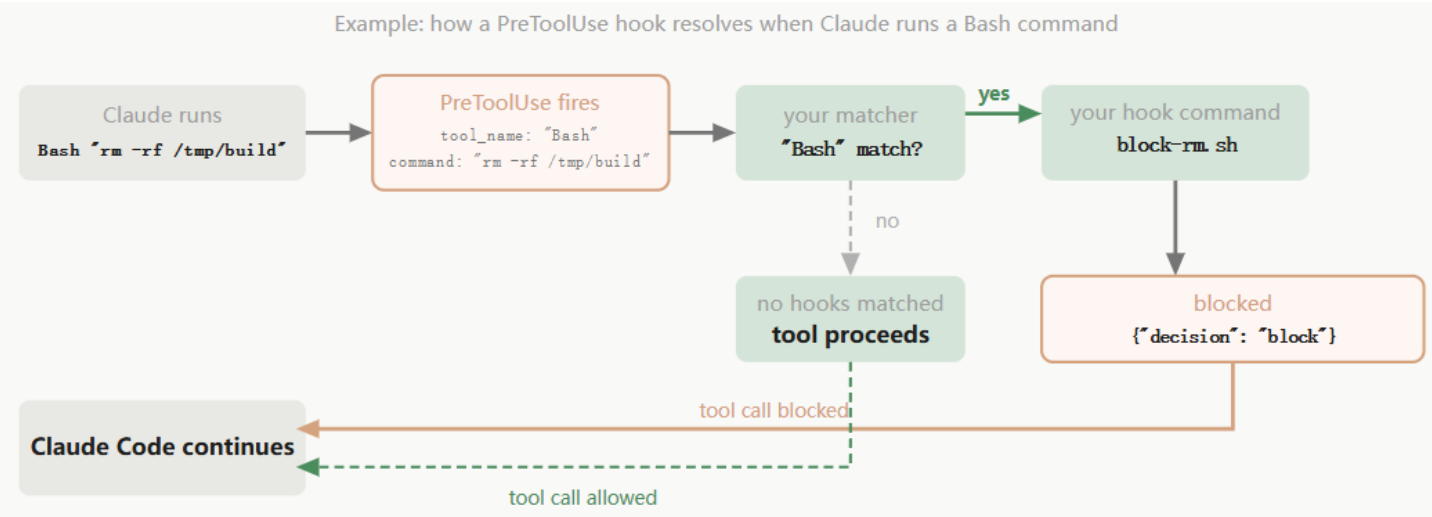
三、怎么用hook

为了了解这些组件是如何协同工作的，可以看这个 PreToolUse 钩子，它可以阻止破坏性的 shell 命令。该钩子会在每次调用Bash工具之前运行 block-rm.sh：

代码块

```
1  {
2    "hooks": {
3      "PreToolUse": [
4        {
5          "matcher": "Bash",
6          "hooks": [
7            {
8              "type": "command",
9              "command": ".claude/hooks/block-rm.sh"
10           }
11         ]
12       }
13     ]
14   }
15 }
```

该脚本从标准输入读取 JSON 输入，提取命令，如果命令包含 rm -rf，则返回 permissionDecision 为 “deny”：



有Hook且匹配 → 脚本决定放行还是拦截；没有Hook匹配 → 直接放行。两条路最终都回到 Claude Code continues，只是一个带着被拦截的结果，一个带着执行完毕的结果。

```

1  #!/bin/bash
2  #告诉系统用bash来运行这个文件
3  #.claude/hooks/block-rm.sh
4  #stdin传来的 JSON → jq 解析 → 取出tool_input.command 的值 → 存进变量 COMMAND
5  COMMAND=$(jq -r '.tool_input.command')
6
7  if echo "$COMMAND" | grep -q 'rm -rf'; then
8      jq -n '{
9          hookSpecificOutput: {
10             hookEventName: "PreToolUse",
11             permissionDecision: "deny",
12             permissionDecisionReason: "Destructive command blocked by hook"
13         }
14     }'
15 else
16     exit 0 # allow the command
17 fi

```

整体流程：

Claude想执行 `rm -rf /folder`



PreToolUse 触发，JSON 传入脚本



jq 取出 command 字段



grep 检测到 `rm -rf`



输出 deny JSON → Claude Code阻止执行

四、hook声音提示实践

在用Claude Code 处理任务时，总得守在电脑旁边——不知道它什么时候跑完，不知道什么时候需要确认操作，只能时不时切回窗口看一眼，打断自己手头的事。

JARVIS Hook就是为了解决这个问题。在几个关键节点加入语音提示：发出指令时有回应，需要确认时会叫你，任务完成时语音提醒。不用盯着屏幕，该干嘛干嘛，声音会在对的时候提醒。目前只实现了最基础的几个节点，后续在实际使用中遇到新的需求，再逐步把对应的Hook加进来。

配置文件

~/.claude/settings.json

```
1 代码块 "hooks": {
2      "UserPromptSubmit": [
3          {
4              "hooks": [
5                  {
6                      "type": "command",
7                      "command": "py -3 C:/Users/18089/.claude/hooks/jarvis_hook.py",
8                      "timeout": 10
9                  }
10             ]
11         }
12     ],
13     "Notification": [
14         {
15             "hooks": [
16                 {
17                     "type": "command",
18                     "command": "py -3 C:/Users/18089/.claude/hooks/jarvis_hook.py",
19                     "timeout": 10
20                 }
21             ]
22         }
23     ],
24     "Stop": [
25         {
26             "hooks": [
27                 {
28                     "type": "command",
29                     "command": "py -3 C:/Users/18089/.claude/hooks/jarvis_hook.py",
30                     "timeout": 10
31                 }
32             ]
33         }
34     ],
35     "SubagentStop": [
36         {
37             "hooks": [
38                 {
39                     "type": "command",
40                     "command": "py -3 C:/Users/18089/.claude/hooks/jarvis_hook.py",
41                     "timeout": 10
42                 }
43             ]
44         }
45     ],
46     "PreCompact": [
47         {
```

```

48         "hooks": [
49             {
50                 "type": "command",
51                 "command": "py -3 C:/Users/18089/.claude/hooks/jarvis_hook.py",
52                 "timeout": 10
53             }
54         ]
55     }
56 ]
57 }

```

没有设置matcher，匹配所有情况，每次事件触发都会执行脚本。

主脚本处理事件 jarvis_hook.py

用于读取事件名，播放对应语音

代码块

```

1  data = json.loads(sys.stdin.read())    #从 stdin 读取 Claude Code传来的 JSON
2  event = data.get('hook_event_name')    #取出事件名
3  #简单的判断逻辑
4  if event == 'UserPromptSubmit':
5      f = 'prompt_submit.mp3'
6  elif event == 'Stop':
7      f = 'task_done.mp3'
8  # ...
9  play(os.path.join(sound_dir, f))      # 用 winmm.dll 播放 MP3
10
11  播放用的是 Windows 底层 winmm.dll，直接调用系统多媒体接口，不依赖任何第三方库：
12
13  winmm.mciSendStringW(f'open "{path}" type mpegvideo alias jv', None, 0, 0)
14  winmm.mciSendStringW('play jv wait', None, 0, 0)
15  winmm.mciSendStringW('close jv', None, 0, 0)

```

关于设置的五个事件的实际触发场景

①发一条消息 → UserPromptSubmit

输入："帮我写一个排序函数"→ 回车

→ jarvis_hook.py 收到事件

→ 播放 prompt_submit.mp3: **"System online. At your service, minnn."**

→ Claude开始处理

② Claude 需要确认 → Notification

Claude 想执行危险操作，弹出权限确认框

→ `notification_type == "permission_prompt"`

→ 播放 `need_confirm.mp3`: **"Awaiting confirmation, minnn."**

→ 提醒去看对话，点确认或拒绝

③ 任务完成 → Stop

Claude完成本次回复，停止输出

→ 检查 `stop_hook_active`（防止无限循环）

→ 播放 `task_done.mp3`: **"Mission accomplished,minnn."**

④ 子代理完成 → SubagentStop

Claude 调用了Task 工具，子代理跑完返回

→ 播放 `subagent_done.mp3`: **"Sub-agent protocol complete,minnn."**

⑤ 上下文压缩前 → PreCompact

对话历史太长，Claude Code 准备自动压缩

→ 播放 `pre_compact.mp3`: **"Initiating context compression,minnn."**

→ 提醒：历史记录即将被整理

完整的工作流work flow

代码块

```
1      发消息
2          ↓
3      Claude Code 触发 UserPromptSubmit
4          ↓
5      JSON 通过 stdin 传入 jarvis_hook.py:
6      {
7          "hook_event_name": "UserPromptSubmit",
8          "session_id": "xxx",
9          "prompt": "帮我写一个排序函数"
10     }
11         ↓
12     脚本读取 event → 匹配 → 找到 prompt_submit.mp3
13         ↓
14     winmm.dll 播放语音
15         ↓
16     print('{}') → exit0 → Claude 继续处理你的消息
```

五、可以复用的hook的skill

是什么？

一个让 Claude Code 在关键节点自动播报语音的系统，像钢铁侠的 JARVIS 一样提示你当前状态。

代码块

1

jarvis-hook/

2

├── SKILL.md

← 完整文档：架构、安装步骤、技术说明

3

└── scripts/

4

├── jarvis_hook.py

← 主Hook脚本，部署到 ~/.claude/hooks/

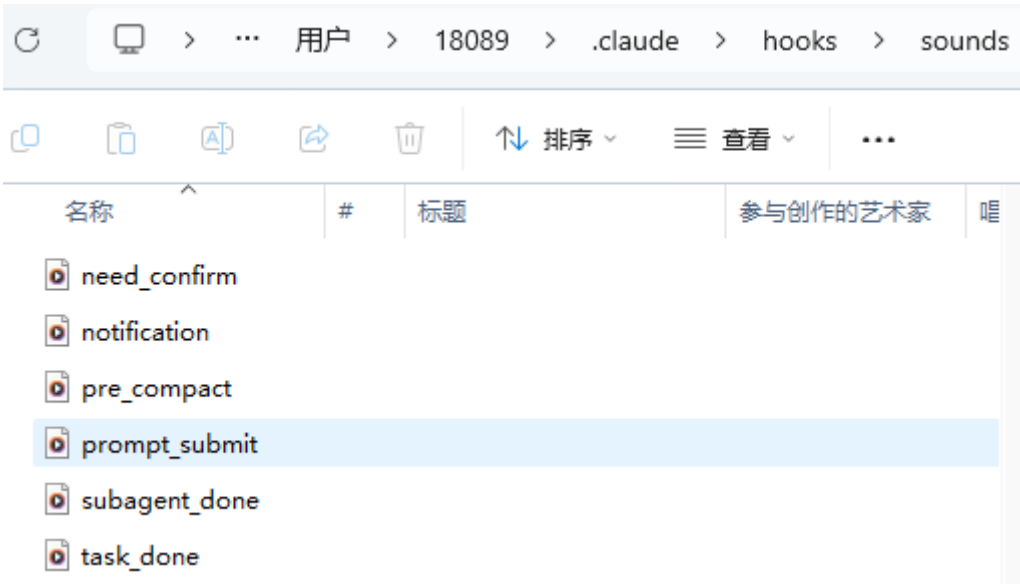
5

└── generate_sounds.py

← 语音生成工具，用来制作 MP3 文件

语音生成：`generate_sounds.py`

用微软edge-tts神经网络语音引擎生成 MP3，可以自定义：



六条台词对应六个事件，每条都带上了我的名字 "Minnn"：

prompt_submit → "System online. At your service, Minnn."

need_confirm → "Awaiting confirmation, Minnn."

task_done → "Mission accomplished."

subagent_done → "Sub-agent protocol complete."

pre_compact → "Initiating context compression."

notification → "Incoming notification, Minnn."

运行一次生成，MP3 文件存到 sounds/ 目录，之后每次触发直接播放，不重新合成。

扩展能力

SKILL.md中预留了飞书通知的扩展方案——任务完成时除了播语音，还能往飞书发一条消息，在手机上也能收到通知。

