Claude Code 介绍以及开源生态

2025年7月20日　10 分钟阅读

#Agent #Claude #Code

> *Claude Code 是 Claude 的命令行工具，用于代理编码，提供灵活的、可定制的、可脚本化的和安全的编程方式。*

摘要

Claude Code 是一种命令行工具，用于代理编码，提供灵活的、可定制的、可脚本化的和安全的编程方式。 本文介绍Claude Code的快速上手，有用的功能，以及一些最佳实践。以及Claude Code的开源生态。 考虑到Claude账号对中国用户不友好，本文介绍的Claude Code 使用的是第三方和中国大陆的模型提供商。

关键点

- Claude Code 是一个用于代理编码的命令行工具，提供灵活和低级别的模型访问。
- CLAUDE.md 文件可以用于自动化上下文提取，是记录常用命令、代码风格和开发环境设置的理想场所。
- Claude Code 可以通过多种方式管理允许的工具，确保安全性。
- 可以通过自定义斜杠命令和使用 MCP 服务器扩展 Claude 的功能。
- 常见工作流程包括探索-计划-编码-提交，以及测试驱动开发等。
- Claude Code 支持无头模式，可用于自动化和持续集成。
- 多 Claude 工作流可以提高效率，例如一个 Claude 编写代码，另一个进行验证。
- 使用 git worktrees 可以在同一代码库的不同分支上同时运行多个 Claude 会话。

安装

方式一： 直接使用Claude Code

下面的步骤运行在Windows 11 + Powershell （比如 VS code + powershell terminal）

目录

文章信息

字数

阅读时间

发布时间

更新时间

标签

#Agent #Claude #C

Step 1:在VS code或Cursor里面的，打开一个powershell的terminal，输入以下命令：

```
# Install Claude Code
npm install -g @anthropic-ai/claude-code

# Navigate to your project
cd your-awesome-project
```

Step 2: 在项目中进行配置 建立.claude文件夹，里面建立一个settings.json文件，内容如下： 更多具体配置参考：https://docs.anthropic.com/en/docs/claude-code/settings

```
{
    "installMethod": "unknown",
    "autoUpdates": true,
    "hasCompletedOnboarding": true,
    "env": {
      "CLAUDE_CODE_MAX_OUTPUT_TOKENS": 1M,
      "ANTHROPIC_BASE_URL":
"https://api.moonshot.cn/anthropic/",
      "ANTHROPIC_API_KEY": "xxx"
    },
    "enabledMcpjsonServers": ["memory", "github"]  #
optional
}
```

Step 3: 配置MCP服务器 在.claude文件夹里面建立一个mcp.json文件，内容如下：

```
{
 "mcpServers": {
  "task-master-ai": {
   "command": "npx",
   "args": ["-y", "--package=task-master-ai", "task-master-ai"],
   "env": {
    "ANTHROPIC_API_KEY": "ANTHROPIC_API_KEY_HERE",
    "PERPLEXITY_API_KEY": "PERPLEXITY_API_KEY_HERE",
    "OPENAI_API_KEY": "sk-xxx",
    "OPENAI_BASE_URL": "https://api.openai.com/v1",
    "GOOGLE_API_KEY": "GOOGLE_API_KEY_HERE",
    "XAI_API_KEY": "XAI_API_KEY_HERE",
    "OPENROUTER_API_KEY": "xxx",
    "MISTRAL_API_KEY": "MISTRAL_API_KEY_HERE",
    "AZURE_OPENAI_API_KEY": "AZURE_OPENAI_API_KEY_HERE",
    "OLLAMA_API_KEY": "OLLAMA_API_KEY_HERE"
   }
  },
  "context7": {
   "command": "npx",
   "args": ["-y", "@upstash/context7-mcp"]
  },
  "sequential-thinking": {
   "command": "npx",
   "args": ["-y", "@modelcontextprotocol/server-sequential-thinking"]
  }
 }
}
```

方式二： 使用Claude-Code-Router

环境： Windows 11 + Git bash + Claude Code + Claude Code Router

Step 1： 安装Claude Code 在visual code或cursor的git bash 终端里面运行下面命令：

```
# Install Claude Code
npm install -g @anthropic-ai/claude-code

# Navigate to your project
cd your-awesome-project

# Start coding with Claude
claude
```

Step 2: 安装Claude Code Router 参考https://github.com/musistudio/claude-code-router/blob/main/README_zh.md

```
npm install -g @musistudio/claude-code-router
```

Step 3: 配置 Claude Code Router 创建并配置你的 ~/.claude-code-router/config.json 文件。具体参考配置示例 如果router需要通过一个HTTP(s)_PROXY来链接外部大模型Provider，那么PROXY_URL是必须的。HOST 和APIKEY对本地部署router不是必须的。关键配置是Providers和Router。

> Providers: 用于配置不同的模型提供商（比如Openrouter, OpenAI, Gemini, DeepSeek，Kimi等等）。
>
> Router: 用于设置路由规则。default 指定默认模型，如果未配置其他路由，则该模型将用于所有请求。

使用 router 启动 Claude Code：

```
ccr code
```

注意: 每次修改完配置文件，要使用"ccr restart"来重启router服务使配置生效。

Claude的能力

Claude Code 会根据需要读取你的文件——你无需手动添加上下文。Claude 还可以访问其自身的文档，并能回答有关其功能和能力的问题。

Claude Code 在修改文件前总是会请求许可。你可以批准单独的更改，或者在会话期间启用“全部接受”模式。

和Git交互

```
what files have I changed?
commit my changes with a descriptive message
create a new branch called feature/quickstart
show me the last 5 commits
help me resolve merge conflicts
```

记忆能力

CLAUDE.md 文件是Claude 在开始对话时会自动将其纳入上下文。可以理解为claude code的记忆文件或rules文件。该文件没有固定的格式要求。我们建议保持其简洁易懂、便于人类阅读。 主要用于：

> 开发环境设置（例如，使用 pyenv、conda，uv，哪些编译器可用）
>
> 核心文件和实用函数
>
> 代码风格指南
>
> 测试说明
>
> 该项目特有的任何意外行为或警告

你希望Claude Code记住的其他信息

CLAUDE.md最佳实践

分层定义内容：全局（**根目录**）放通用规范，模块目录放特定约定，**子目录**放特殊说明，既保证继承又避免冗余。

定期审查聚合效果：多层级聚合可能导致冲突或信息冗余，建议定期用Claude的"上下文预览"功能检查实际生效的内容。

善用自动生成与编辑命令：通过/init快速生成初始CLAUDE.md，用#指令实时补充，保持内容鲜活。让其自动整合到CLAUDE.md中。许多工程师在编码时经常使用#来记录命令、文件和风格指南，然后在提交时包括CLAUDE.md的更改，以便团队成员受益。

版本化管理：将主CLAUDE.md纳入git版本管理，辅助用CLAUDE.local.md满足个性化需求。

自己独有的claude code用法习惯可以配置在"**~/.claude/CLAUDE.md**"

应该优化CLAUDE.md文件，因为它们是Claude提示的一部分。常见错误是添加大量内容而不评估其效果。需要花时间实验，以找到最佳的指令执行方式。

在Anthropic，他们偶尔会通过"**提示改进器**"运行CLAUDE.md文件，并经常调整指令（例如用"IMPORTANT"或"YOU MUST"加以强调）以提高遵循度。

例子

```
# Bash commands
- npm run build: Build the project
- npm run typecheck: Run the typechecker

# Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from
'bar')

# Workflow
- Be sure to typecheck when you're done making a series of code
changes
- Prefer running single tests, and not the whole test suite, for
performance
```

用法

用法1: 集成到github action workflow Claude code action 允许你在GitHub Actions工作流中运行Claude Code。你可以使用此功能在Claude Code基础上构建任何自定义工作流。

参考Claude Code Github Action 文档 参考claude-code-action 参考claude-code-action 使用示例

case 1: 介绍代码库

```
> I'm new to this codebase. Can you explain it to me?
我对这个代码库还不熟悉。你能给我讲讲吗？
```

case 2: 解决issue

```
> Can you look at the open Github issues for the Financial Data Analyst project and fix ones that are relevant?
你能查看一下“金融数据分析师”项目在 GitHub 上的未解决的问题，并解决那些与之相关的问题吗？
```

Tips:

> *告诉克劳德重现问题的指令，并获取一个堆栈跟踪。*
>
> *请提及重现错误的任何步骤。*
>
> *如果错误是间歇性的还是持续性的，请告知克劳德。*

case 3: 重构代码 更多具体可参考：重构代码

```
> Refactor the permission request components to share common UI elements and behavior.
重构权限请求组件，以共享通用的用户界面元素和行为。
```

case 4: 编写测试

```
> write unit tests for the calculator functions
```

case 5: 更新文档

```
 update the README with installation instructions
```

case 6: 审查代码并提出改进建议

```
review my changes and suggest improvements
```

case 7: 生成技术文档 比如，生成一个技术文档，方便新人学习已有代码库实现，或为后期AI Coder提供上下文。

```
trace the login process from front-end to database
```

更多例子： 先从宽泛的问题入手，然后再逐步聚焦到具体领域。 询问项目中使用的编码规范和模式 请求一份项目专用术语表

```
give me an overview of this codebase
explain the main architecture patterns used here
what are the key data models?
how is authentication handled?
```

case 8: 编写代码

使用工具

Git

Claude可以使用gh CLI与GitHub进行交互，如创建问题、打开拉取请求、读取评论等。如果没有安装gh，Claude仍可以使用GitHub API或MCP服务器（如果已安装）。Claude Code 使用 Git 的场景包括：

查询历史记录：通过搜索 Git 历史来回答诸如“v1.2.3 版本包含了哪些更改？”、“谁负责这个特定功能？”或“为什么这个 API 被设计成这样？”等问题。明确提示 Claude 检查 Git 历史可以帮助回答这些问题。

编写提交信息：Claude 会自动查看你的更改和最近的历史记录，综合相关上下文生成合适的提交信息。

处理复杂操作：帮助进行复杂的 Git 操作，如回滚文件、解决 rebase 冲突以及比较和应用补丁。

MCP

在.mcp.json 文件中配置MCP Server

在使用 MCP 时，也可以通过用 –mcp-debug 标志启动 Claude 来帮助识别配置问题。

Command

可以将重复的工作流调试循环和日志分析等提示模板存储在 .claude/commands 文件夹的 Markdown 文件中，以便通过斜杠命令菜单使用，并可以提交到 git 供团队其他成员使用。

例子： 用来自动拉取并修复 Github 问题的斜杠命令：

该命令内容存在文件“.claude/commands/fix-github-issue.md”， 在Claude code里面使用“/project:fix-github-issue 1234”来修复issue 1234。

```
Please analyze and fix the GitHub issue: $ARGUMENTS.

Follow these steps:

1. Use `gh issue view` to get the issue details
2. Understand the problem described in the issue
3. Search the codebase for relevant files
4. Implement the necessary changes to fix the issue
5. Write and run tests to verify the fix
6. Ensure code passes linting and type checking
7. Create a descriptive commit message
8. Push and create a PR

Remember to use the GitHub CLI (`gh`) for all GitHub-related tasks.
```

使用Workflow

WF 1: 探索、规划、编码、提交 (解决复杂问题)

探索 (Explore)：让 Claude 阅读 相关文件/图片/URL（给出大致指向或具体名称）。明确禁止写代码。仅作探索和信息收集。

规划 (Plan)：

使用子代理 (Subagents), 复杂问题应在探索/规划阶段使用子代理验证细节和疑问，能更好地保持上下文。

深度思考 (Think Mode)：要求 Claude 制定具体解决方案计划时，用 think， think hard， think harder， ultrathink 等词触发逐级增强的思考预算。

产出计划文档： 若方案可行，生成文档（如README、GitHub Issue）作为回溯点。

编码 (Code)：让 Claude 编写代码实现解决方案。要求其在编码过程中实时验证解决方案的合理性。

提交 (Commit)：让 Claude 提交代码（Commit）。创建拉取请求 (Create PR)。 （可选）更新文档： 如修改 README 或变更日志，说明变更内容。

核心强调：

步骤1（探索）和2（规划）至关重要。 缺少它们会导致 Claude 过早开始编码，不利于需要深度思考的问题。

规划阶段使用**子代理**和**深度思考模式**能显著提升复杂问题的处理效果。

规划文档化提供了重要的回溯点。

WF 2: 测试驱动开发 （TDD开发流程）

工作流“Write tests, commit; code, iterate, commit” 的要点总结：

编写测试 (Write Tests)：◦明确要求 Claude 基于期望输入/输出来编写测试（单元、集成、E2E）。◦关键指令：声明使用测试驱动开发 (TDD)，防止它过早创建模拟实现（即使功能还不存在）。◦确保测试覆盖预期行为。

运行测试并验证失败 (Run & Verify Failure)：◦要求 Claude 运行刚写的测试。◦关键指令：明确要求此时不写实现代码。◦确认测试在实现缺失或不正确时确实失败。

提交测试用例 (Commit Tests)：◦对测试满意后，让 Claude 提交这些测试用例到版本控制（如 Git）。形成明确的起点。

编写实现代码并迭代验证 (Code & Iterate)：◦要求 Claude 编写代码以使测试通过。◦关键指令：禁止修改测试本身（除非测试本身有误）。◦让 Claude 持续迭代：编写代码 -> 运行测试 -> 调整代码 -> 运行测试，直至所有测试通过。◦建议：在迭代中可要求 Claude 使用独立子代理验证实现是否过度拟合现有测试（即是否能泛化）。

提交实现代码 (Commit Code)：◦对实现满意后，让 Claude 提交通过的代码。

核心优势与原理:

清晰目标驱动： 测试提供了明确的、可验证的目标（“靶子”），让 Claude 能聚焦迭代。

迭代优化： 通过“写-测-调-测”循环实现渐进式改进。

TDD强化： AI 代理执行 TDD 流程（先写测试->失败->实现->通过）效果显著。

防过拟合（可选）： 子代理验证有助于提高解决方案的健壮性。

WF 3: 编写代码 → 截图对比 → 迭代优化 （适用于视觉驱动的开发流程）

提供视觉目标

让 Claude 能获取屏幕截图（方式可选）：
• 使用 Puppeteer MCP 服务（浏览器自动化）
• 使用 iOS 模拟器 MCP 服务
• 手动复制/粘贴截图 到对话
• 直接提供 图片文件路径

设定视觉参考（Mock）

通过 拖放图片、复制粘贴或文件路径 提供设计稿/目标效果图。

编写代码 & 视觉对比迭代

让 Claude 根据设计稿编写代码，并 截图当前实现。

对比截图与目标设计，调整代码，循环优化，直到匹配。

关键点：

通常 2-3 轮迭代 后效果显著提升。

第一版可能接近，但多次优化后会更精准。

提交最终版本

满意后，让 Claude 提交代码（Commit）。

核心优势

视觉驱动开发：像人类开发者一样，Claude 通过 截图对比 能更精准调整 UI/视觉效果。

迭代提升质量：多次优化比单次输出效果更好。

适用于：前端开发、UI 调整、设计还原等场景。

!!!TBD

Hooks

https://www.youtube.com/watch?v=J5B9UGTuNoM

Subagent

https://www.youtube.com/watch?v=7B2HJr0Y68g

注意：Claude官方给出的Subagent 最佳实践:

设计专注的子代理：建议先用 Claude 生成初始子代理，再进行迭代和定制，以获得最佳效果。

编写详细的提示：在你的系统提示中包含具体的说明、示例和限制条件。你提供的指导越多，子代理的表现就会越好。

限制工具访问权限：仅授予子代理执行其任务所必需的工具。这有助于提高安全性，并使子代理能够专注于相关操作。

版本控制：将项目子代理程序签入版本控制系统，以便你的团队能够共同受益并改进它们。

链接子代理：对于复杂的流程，你可以串联多个子代理。e.g.

方法一：直接描述

```
First use the code-analyzer subagent to find performance
issues, then use the optimizer subagent to fix them
```

方法二： 在claude code的一个command里面把不同的子代理链接起来，比如：

```
Here are your instructions for the provided URL ($ARGUMENTS)
:

第一步:使用post-creator生成帖子。
第二步:使用post-reviewer对帖子进行审核,并打分、给出解释。
第三步:将post-reviewer的反馈给到post-creator,由post-creator对帖子
进行修改。
第四步:重复第一到第三步,最多三次。直到post-reviewer给出的分数超过90
分。
第五步:将最终版的帖子存进根目录下,文件为Markdownif格式。
Don't run any other additional commands or instructions.
```

动态subagent选择：

Claude Code 会根据上下文智能地选择子代理。为了获得最佳效果，请将你的**description**字段设置得具体且具有行动导向。

注意：

关键概念：代理文件（.claude/agents/*.md）中的内容是配置子代理行为的系统提示，而非用户提示。这是创建代理时最常见的误解。

强烈推荐的关于Subagent的Youtube视频：My Claude Code Sub Agents BUILD THEMSELVES。

例子

官方给了3个Subagent的例子

Code Reviewer

Debugger

Data-Scientist

Github里面Subagent的示例 在Github里面使用关键字“claude code subagent”可以找到更多的Subagent的示例。目测写的Prompt格式都不一样，不确定哪些效果更好。先暂时按照star数目排序(截止至2025/08/13)：(大多数都按类别分，方便查找)

https://github.com/wshobson/agents （8.1k）

https://github.com/iannuttall/claude-agents （1.6k）

https://github.com/davepoon/claude-code-subagents-collection （1.5k）

https://github.com/VoltAgent/awesome-claude-code-subagents (988)

https://github.com/lst97/claude-code-sub-agents (782)

https://github.com/0xfurai/claude-code-subagents (158) 这个只关注不同语言。

有趣的用法

可以使用“claude –verbose”这个详细模式来测试你的子代理。你将看到你的主代理发送给子代理的确切提示、详细步骤、工具执行情况以及子代理获取并返回给主代理的输出。这有助于你更好地优化子代理。一如既往，你的视频太棒了。我从你的视频中学到了很多。

使用 –system-prompt 来突破主要代理瓶颈，直接查看代理情况

基于meta-agent代码。可以建立一个很棒的循环，plan代理、build代理和review代理依次为用户执行任务（没有具体信息）。

用 Puppeteer 做了一个“UI 摄影记者代理”，它会拍摄 UI 修复前后的画面，让主代理“看到”自己的工作成果。用的“Puppeteer”库，它有一个屏幕截图工具。Claude Code 可以帮忙设置其余部分…

有趣的观点

？？ Claude 不仅可以使用Subagent（子代理），而且可以设置主代理（meta-agent）。 主代理是其他代理集合的容器。子代理是元代理的一部分（例如，构建子代理、调试子代理等）。这种机制对团队非常有用，在这里你可以安排代理人来执行不同的任务。

Anthropic 团队正在发布这些极其宝贵、强大的基础模块，还不清楚他们接下来会怎么做，但一旦完成这些，我认为他们会开始构建顶层/抽象层，以清晰的方式整合所有这些模块，从而击败从高级（如 Loveable、Replit）到低级（如 Cursor、GitHub）的所有工具。

我们需要类似代理协调器的东西，并验证一个代理给出的答案是否与另一个代理所期待的内容相兼容……考虑一下变量和代理合同（$variables and agent contracts）。或许可以有一个合同协调代理来处理对齐问题。

有人对meta-agent做了微调：

description: Builds new Claude Code sub agents from descriptions. Use PROACTIVELY if the user says 'create agent', 'build agent', 'new sub agent', 'meta-agent', or 'meta agent'. When prompting this agent, describe the specific role, capabilities, and tools the new agent should have.

在 Claude Code 1.0.64 中，你可以指定子代理将使用哪个模型。

你如何看待 BMAD 和 superclaude 这两个框架？你觉得它们有价值吗？或者它们与你正在使用的概念混合起来是否有价值？

我很想看看它与 Claude-Flow 相比如何，Claude-Flow 是通过创建群集来实现目标的。

你竟然没有使用 Wispr Flow 与终端进行交流，这让我感到很意外……

但在设置好代理后，我看到响应中报告的代币使用量非常惊人。像只是返回它能做什么的简单描述这样的基本功能，就显示用了 2 万代币。但这一切都在几秒钟内完成，所以我不明白为什么代币使用量会这么高。每次基本功能都要用掉 2 万代币，这似乎还不太实用。丹，你那边情况如何？代币使用量是正常的，还是代理调用报告有误？

今天我看到克劳德给一个二级代理打电话，那个二级代理又给另一个二级代理打了电话，我想知道这是不是新功能？还有其他人注意到吗？

我在 claude-code 项目中发现了很多关于子代理无法生成其他代理的问题，存在一些技巧，比如生成无头的 Claude 实例，或者使用像 claude-flow 这样复杂的工具。对于简单的流程，尽量使用命令作为主要的“代理”来生成子代理。

Plugin

TBD

节约成本的最佳实践

/clear /compress 指定范围和文件。

最佳实践

A. 使用 claude.md 文件

作用： 作为跨会话或团队共享状态的主要方式，弥补 Claude Code 没有内存的不足。

工作原理： 在启动时，如果工作目录中存在 claude.md 文件，其内容会被加载到上下文（prompt）中，作为重要的开发人员指令。

放置位置：项目目录： 与团队共享项目特定说明。

用户主目录： 存储始终希望 Claude 知道的通用指令。

内容示例： 运行单元测试的方法、项目布局概述、测试位置、模块信息、风格指南等。

引用其他文件： 可以在 claude.md 中引用其他 claude.md 文件或相关文件（例如使用 @ 符号）。

多 claude.md 文件： 默认只读取当前工作目录的 claude.md 文件；子目录中的 claude.md 文件需要 Claude 在搜索时主动发现并读取。 B. 权限管理

默认行为： 读取操作自动放行，写入或执行 Bash 命令等可能改变机器状态的操作需要用户确认（“yes”、“always allow”、“no”）。

加速工作流：自动接受模式（Autoaccept mode）： 按 Shift+Tab 可让 Claude 自动执行。

配置特定命令： 在设置中配置特定 Bash 命令始终被批准（例如 npm run test）。 C. 集成设置（Integration Setup）

CLI 工具： Claude 擅长使用终端，通过安装更多的 CLI 工具（如 GitHub 的 GH 工具）可以赋予其更多能力。推荐优先使用文档完善的 CLI 工具而非 MCP 服务器。

内部工具： 可以将内部专用工具（如 Anthropic 的 Koup）告知 Claude（通常通过 claude.md）。 D. 上下文管理（Context Management）

上下文窗口限制： Anthropic 的模型有 200,000 个 tokens 的上下文窗口限制，长时间会话可能填满。

处理方法：/cle： 清除所有上下文，除 claude.md 外，重新开始新会话。

/compact： 插入用户消息，让 Claude 总结当前会话，然后用此总结作为新会话的种子继续。 E. 高效工作流

规划与待办事项：问题分析与计划： 不直接要求修复 bug，而是让 Claude 搜索原因并提供修复计划，由用户验证。

待办事项列表： Claude 在执行大型任务时会创建待办事项列表，用户可以监控并及时按 escape 介入纠正错误方向。

智能副驾驶编码（Smart Vibe Coding）：测试驱动开发（TDD）： 鼓励 Claude 小步修改、运行测试、确保通过。

持续检查： 让 Claude 检查 TypeScript 和 Linting。

定期提交： 确保可以回溯，避免大型错误。

使用截图引导与调试： Claude 基于多模态模型，可以直接粘贴截图或引用图片文件（如 mock.png）作为输入，引导其构建网站等。 F. 高级技巧

同时运行多个 Claude 实例： 挑战用户尝试同时运行多个 Claude 实例（例如在 Tmux 或不同标签页中），进行编排。

使用 Escape 键：中断与干预： 在 Claude 工作时按 escape 中断并引导其更改方向。

双击 Escape： 隐藏功能，允许回溯对话，重置工具扩展。

MCP 中的工具扩展： 如果 Bash 和内置工具无法满足需求，考虑使用 MCP 服务器。

无头自动化： 通过 GitHub Actions 等方式程序化地使用 Claude，探索更多创意集成点。 IV. 最新功能与更新

A. 模型选择与配置

/model 命令： 查看当前运行的模型（例如 Sonnet）。

/config 命令： 切换到不同的模型（例如 Opus）。

B. 工具调用间的“思考”（Think Hard）

功能增强： Claude 4 模型现在可以在工具调用之间进行“思考”，这在旧模型中是不允许的，而这正是思考最关键的时候。

触发方式： 在提示中加入“think hard”或“extended thinking”。

视觉指示： 思考过程会显示为浅灰色文本。

C. IDE 集成

VS Code 与 JetBrains： Claude 现在与主流 IDE 有更好的集成，例如能感知用户正在编辑的文件。

D. 保持更新

变更日志： 访问 Anthropic 在 GitHub 上的 Claude Code 公开项目，查看变更日志以获取最新功能和更新信息。

集成不同工具使用

Super Claude Framework (BMAD Style)

Super Claude Framework github

SuperClaude Framework 是一个增强 Claude Code 的配置框架，通过注入行为指令和组件编排，将其转变为结构化开发平台，支持系统化的工作流自动化。它提供 25 条命令、15 个专门智能代理、7 种行为模式以及 8 个 MCP 服务器集成，实现从头脑风暴到部署的完整生命周期管理。使用 `/sc:help` 查看所有命令列表。

最新版本 v4 引入以下核心功能：

> **智能代理系统**：15 个领域专用代理，包括深度研究、安全分析、前端架构等自动协调工具。
>
> **行为模式**：7 种适应性模式，如头脑风暴、任务管理、深度研究等。
>
> **MCP 服务器集成**：支持复杂分析、UI 组件生成、性能测试等强大功能。
>
> **深度研究体系**：适用于自主 Web 研究，支持多跳推理、质量评分和跨会话学习。
>
> **性能优化**：框架体积减小但支持更复杂任务。

支持多种安装方式（`pipx`、`pip` 和 `npm`），提供详细文档覆盖快速入门、命令参考、代理指南等领域，欢迎社区贡献代码、文档改进以及测试工作。

框架采用 MIT 开源许可，项目受到 Anthropic 公司的 Claude Code 启发，但非官方关联或认可。

Zen MCP Server

Graphiti MCP Server

Claude Code Templates (BMAD Style & SDD Style & Claude Code Style(hook, command, mcp, setting, template))

Claude Code Templates, 主页位于https://www.aitmpl.com/ 是一个开源项目，提供针对 Anthropic 的 Claude Code 的配置工具集与开发模板。主要功能包括：

模块分类：

AI代理（Agents）：领域专家智能代理，如安全审计员、性能优化器等。

命令（Commands）：自定义命令，如 /generate-tests, /optimize-bundle。

外部服务集成（MCPs）：支持 GitHub、PostgreSQL、Stripe 等服务。

配置（Settings）：相关参数设定，如超时设置、输出样式等。

自动触发（Hooks）：开发流程中的自动化操作。

项目模板（Templates）：框架定制化项目配置和最佳实践集合。

核心工具：

Claude Code Analytics：实时监控开发过程中的性能和状态。

对话监控（Conversation Monitor）：支持本地与远程访问 Claude 回复。

健康检查（Health Check）：优化系统运行状况。

安装方式：

使用 `npx` 快速安装整个开发栈或单个组件。

提供交互式浏览和模块安装方式。

贡献与文档：

欢迎用户按照详细指南贡献新模块与功能。

文档与教程可在项目官网或文档中心（docs.aitmpl.com）查看。

项目采用 MIT 许可，支持多种编程语言（如 JavaScript, Python 等），拥有 6600+ 星标、586 次分叉，并持续部署更新。

Claude-Flow (SDD Style)

Claude-Flow 是一个企业级 AI 协作平台，支持多智能体编排和自动化工作流，专注于利用 Claude Code 构建高效 AI 系统。主要特性包括：

性能提升：通过 Claude Code SDK，实现了 100-600x 的性能加速，支持实时查询控制、动态权限调整和快速智能体生成。

高级功能：

会话分叉：支持实时控制、多智能体并行任务。

钩子匹配：优化触发速度，支持四级权限体系和智能缓存。

内置 MCP 服务器：通过内存节省和工具调用加速，提升执行效率。

云平台集成：

Flow Nexus 提供沙盒环境、分布式学习和模板市场。

支持自组织智能体架构、关键任务持久化存储以及 GitHub 自动化功能。

高效工作流：通过 swarm 和 hive-mind 命令简化任务和项目管理，适用于从单一任务到复杂项目的各种场景。

文档与支持：包含详细教程、API 参考和高效配置，社区支持活跃。

Claude-Flow 是多智能体框架领域的领导者，拥有 87 工具和 64 专业智能体，优化了 AI 编排流程，是构建高效对话系统和分布式智能体的理想选择。

Context Engineering Template （SDD Style)

该GitHub项目context-engineering-intro 提供了一个**全面的模板**(没有代码），用于指导如何进行上下文工程（Context Engineering），以优化AI编码助手的工作效率。上下文工程是一种比传统提示工程（Prompt Engineering）更高效的方式，它通过提供全面的上下文（包括文档、示例、规则等），使AI能够完成复杂任务并减少失败率。

项目内容包括：

快速开始指南：用户可通过克隆模板、设置规则（CLAUDE.md）、添加代码示例（examples/文件夹）、创建功能需求（INITIAL.md），生成并执行产品需求提示（PRP）。

优势分析：上下文工程比仅提供任务提示的提示工程更强大，能够减少AI失误、确保一致性、支持多步实施并提供自校正功能。

模板结构：文件夹包含CLAUDE.md（全局规则）、INITIAL.md（功能需求模板）、PRPs（产品需求提示模板）、examples/（代码示例）和README.md等。

分步指南：从定义项目规则、创建初始需求到生成和执行PRP，强调利用示例代码、文档和验证门确保实施质量。

最佳实践：注重详细需求描述、丰富代码示例、使用验证机制、引用文档以及定制项目规则。

语言主要为Python，支持TypeScript和PLpgSQL。项目强调使用Claude Code作为主要AI助手工具，但方法适用于所有AI助手。

此项目旨在推动上下文工程替代传统开发方式，使AI更高效可靠。

Demo可以参考AI 超元域

Claude Code Spec Workflow (SDD Style)

Claude Code Spec Workflow主要是为了Claude Code完美复现Kiro的Spec-Driven规范驱动开发。claude-code-spec-workflow 这个项目将不再支持，而是转到spec-workflow-mcp.

> *需要注意的是，在Github Spec-Kit开源项目日益成熟后，这个项目可能会被spec-kit取代。*

Spec Workflow MCP 是一个基于模型上下文协议（MCP）的服务器，提供结构化的规范驱动开发工作流程工具，专注于 AI 辅助的软件开发。其主要功能包括实时 Web 仪表盘和 VSCode 插件，帮助开发者直接在开发环境中监控和管理项目进度。

主要特点

结构化开发工作流程 - 包括需求、设计、任务的顺序规范创建。

实时仪表盘 - 提供实时更新的进度监控和任务管理。

VSCode 插件 - 集成侧边栏仪表盘，方便 VSCode 用户使用。

审批工作流程 - 完整审批流程支持修订反馈。

任务进度跟踪 - 可视化进度条及详细状态。

多语言支持 - 提供 11 种语言支持。

快速开始

添加至 MCP 配置。

可选择通过 Web 仪表盘或 VSCode 插件使用。

用法示例

“创建用户认证的规范”：自动生成完整开发规范。

“列出我的规范”：查看所有规范及状态。

“执行规范 user-auth 中的任务 1.2”：运行具体任务。

项目结构与开发

项目提供开发指南，支持依赖安装、构建，以及开发模式运行。开发者可通过 NPM 命令轻松进行设置。

许可协议

GPL-3.0

此项目以 TypeScript 为主开发

Claude Code PM

Claude Code PM 是一个利用GitHub Issues和Git工作树的智能项目管理系统，旨在通过多AI代理并行执行加速软件开发。核心功能包括：上下文保留、并行任务执行、GitHub原生集成，提供完整的追溯能力。主要特点：

转换PRD为任务：通过结构化流程从产品需求文档（PRD）到技术实施、任务分解和代码交付。

多代理并行工作：实现多个AI代理同时执行分解任务，提升开发速度。

GitHub协作：实时同步任务状态，促进团队协作与透明管理。

严格规范设计：提供五阶段工作流程，确保每行代码都可以追溯到规范。

支持情况：优先本地运行，显式与GitHub同步，避免外部工具复杂性。

安装和配置过程简单，可在几分钟内启动，并提供针对团队透明度、开发高效性及错误减少的显著优化解决方案。项目托管于MIT协议，适合需要提升交付效率的开发团队。

项目主页链接：automaze.io/ccpm。

Claude Context

Claude Context Github 是一个为 Claude Code 和其他 AI 编程助手添加语义代码搜索功能的 MCP 插件，能够从整个代码库中提供深度上下文。其核心功能包括：

- 🧠 **语义搜索**：从数百万行代码中快速找到相关代码，无需多轮交互。
- 💰 **成本优化**：通过将代码库存储在向量数据库中，仅使用相关代码上下文，降低大规模代码库的请求成本。
- 🚀 **支持多种技术**：兼容多语言代码搜索，包括 TypeScript、Python、Java、C++ 等；支持多种嵌入模型和向量数据库（例如 Milvus 和 Zilliz Cloud）。
- ⚡ **增量索引与智能分块**：使用 Merkle 树进行增量索引，仅重新索引更改的文件；应用 AST（抽象语法树）分块分析代码。
- 🛠️ **可定制**：支持文件扩展名配置、忽略模式和嵌入模型设置。

快速开始

配置 Vector 数据库（Zilliz Cloud）。

获取 OpenAI API 密钥用于嵌入模型。

在 Claude Code 中添加 MCP 服务器，通过 CLI 命令完成。

支持的用法

VSCode 插件：直接从 IDE 中实现语义代码搜索。

核心包 API：用于自定义项目中代码索引和搜索的功能。

其他 MCP 配置客户端：包括 Gemini CLI、Cursor、Zencoder 等。

架构与未来计划

- 🔍 支持更多嵌入模型
- 🧩 改进代码分块和搜索结果排名
- 💡 开发交互式智能搜索模式
- 🌐 增强的 Chrome 插件

项目采用 MIT 协议；贡献者可参考详尽的开发指南参与。

BMAD-METHOD

BMAD-METHOD 是一个用于敏捷 AI 驱动开发的框架，旨在通过先进的人工智能工具优化软件开发和跨领域工作。

项目核心创新：

- **代理规划**：提供专门的 AI 代理（如分析师、项目经理、架构师），结合高级提示工程和用户参与，生成详细的产品需求文档（PRD）和架构文件。
- **上下文工程开发**：Scrum Master 代理将规划转换为超详细的开发任务，确保开发代理完全理解实现所需的背景、细节和架构指导。

主要功能：

- **代码库展平工具**：将项目文件压缩为便于 AI 消费的结构化 XML，以支持分析、调试或开发帮助。
- **扩展包**：用于创意写作、商业策略、健康与教育等不同领域的专用 AI 代理。
- **快速安装与更新**：一键命令即可完成框架安装或升级，支持定制化文件的保留。

资源与支持：

- 用户指南与技术深潜文档
- 社区支持（Discord）
- 开源贡献指南

此项目基于 MIT 许可协议，支持 Node.js v20+，并提供详细的快速启动和安装指导。

Claudia

Claudia github

Claude Code Router

Claude Code Router github是一个强大的工具，用于将 Claude Code 请求路由到不同的模型并自定义任何请求。本项目主要特点包括模型路由、多供应商支持、请求/响应转换、动态模型切换、GitHub Actions 集成以及插件扩展功能。

核心功能：

- 路由：根据任务需求将请求分配到不同模型。
- 多供应商支持：兼容 OpenRouter、DeepSeek、Ollama、Gemini 等多种模型服务。
- 自定义转换：通过 Transformer 修改请求和响应格式。
- 动态切换：使用 /model 命令动态选择模型。
- UI 模式：提供可视化配置界面（Beta）。
- 环境变量支持：安全管理 API 密钥，通过环境变量插值配置。

安装与使用：

- 安装：支持 npm 包安装。
- 配置：通过创建 `config.json` 文件设置模型供应商、路由规则及其他选项。
- 启动并运行：使用命令 `ccr code` 启动Claude Code（需要先安装Claude Code） 或 `ccr restart` 启动router服务。
- 集成 GitHub Actions：自动化 Claude Code 工作流。

高级功能：

- 支持自定义路由规则，指定模型和供应商。
- 内置与第三方模型交互的转换器，并支持插件加载扩展功能。

适用场景包括代码解释、复杂推理任务、大上下文处理及实时搜索。用户可动态调整模型和路由，系统高度定制化。项目开源，提供中文 README 文档，采用 MIT 许可，鼓励社区支持与赞助。

这是为构建基于 Claude Code 的编码基础设施而开发的项目，支持持续更新以及多种交互自定义选项。支持多种大模型提供商。功能成熟。

KIMICC

一步命令 npx kimicc 使用 Kimi K2 运行 Claude Code。参考开源项目kimicc

集成GLM4.6

集成GLM4.6到Claude Code CLI

GLM4.6采取KIMI类似的方案，既可以直接兼容Claude API，也可以通过使用claude-code-router进行路由。下面是直接配置claude code配置文件或环境变量的方式来直接在Claude Code中使用GLM4.6。

STEP 1: 配置URL和KEY

```
#国内使用
ANTHROPIC_BASE_URL=https://open.bigmodel.cn/anthropic/

#国外使用
# ANTHROPIC_BASE_URL=https://api.z.ai/api/anthropic

# API Key
ANTHROPIC_AUTH_TOKEN=<you glm api key>
```

STEP 2：配置模型名

手动修改配置文件 ~/.claude/settings.json：

```
{
  "alwaysThinkingEnabled": true
  "env": {
    "ANTHROPIC_DEFAULT_HAIKU_MODEL": "glm-4.6",
    "ANTHROPIC_DEFAULT_SONNET_MODEL": "glm-4.6",
    "ANTHROPIC_DEFAULT_OPUS_MODEL": "glm-4.6"
  }
}
```

集成GLM4.6到VS Code插件Claude Code

安装Claude Code插件

进入VS Code设置，找到**Claude Code”

找到“在settings.json中编辑”，贴入下面配置信息

```
{
    "claudeCode.environmentVariables": [
        {
            "name": "ANTHROPIC_BASE_URL",
            "value":
"https://open.bigmodel.cn/api/anthropic"
        },
        {
            "name": "ANTHROPIC_AUTH_TOKEN",
            "value": "you glm api key"
        },
        {
            "name": "ANTHROPIC_MODEL",
            "value": "glm-4.6"
        },
        {
            "name": "API_TIMEOUT_MS",
            "value": "3000000"
        },
        {
            "name":
"CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC",
            "value": "1"
        }
    ],
    "claudeCode.selectedModel": "default"
}
```

在VS Code界面的中上部分偏右找到Claude的图标，点击打开即可。 打开Claude Code界面后，输入“/select mode”选择“Default”即可。

更多GLM4.6套餐信息

使用GLM4.6套餐的方式大约是直接使用API的10%。 具体套餐用量额度如下：

Lite 套餐：每 5 小时最多约 120 次 prompts，相当于 Claude Pro 套餐用量的 3 倍

Pro 套餐：每 5 小时最多约 600 次 prompts，相当于 Claude Max(5x) 套餐用量的 3 倍

Max 套餐：每 5 小时最多约 2400 次 prompts，相当于 Claude Max(20x) 套餐用量的 3 倍

从可消耗 tokens 量来看，每次 prompt 预计可调用模型 15-20 次，每月总计可用总量高达几十亿到数百亿tokens，折算下来仅为 API 价格的 0.1 折，极具性价比。

已支持 Claude Code、Roo Code、Kilo Code、Cline、OpenCode、Crush、Goose 等 10+ 编程工具，

仅 Pro、Max 套餐支持视觉理解、联网搜索MCP工具，Lite 套餐调用视觉理解 MCP 需要单独收费，且暂未支持调用搜索工具。

AI 超元域的demo

集成不同的MCP Server

```
# claude mcp add --transport http context7 https://mcp.context7.com/mcp
claude mcp add context7 npx @upstash/context7-mcp
claude mcp add sequential-thinking npx @modelcontextprotocol/server-sequential-thinking

claude mcp add puppeteer npx @modelcontextprotocol/server-puppeteer

claude mcp add magic npx @21st-dev/magic@latest --env API_KEY=<你的api key>
```

我使用Claude Code 的心得体会

全新设计一个新项目时，开始花更多的时间和Claude Code讨论PRD.

功能点，业务流程，架构的讨论 架构的讨论

技术栈的采用，前后端是否分离，是否使用微服务，是否使用容器化，是否使用云原生，是否使用AI辅助开发 使用MCP task-master制定计划 使用MCP task-master制定计划 使用MCP task-master制定计划

参考

来自官网的最佳实践介绍了使用 Claude Code 的最佳实践，包括如A何设置、优化以及常见的工作流程。 Anthropic团队如何使用Claude Code工作介绍了Anthropic不同团队（开发，安全工程师，产品设计师，项目经理，律师，数据科学家,数据基础设施）如何使用Claude Code工作，包括如何设置、优化以及常见的工作流程。 Claude Code 自己的主页： 详尽的文档，涵盖了所有功能，并提供了更多示例、实现细节和高级技术。 Claude Code的Overview

> **还未翻译好的篇章**

(https://docs.anthropic.com/en/docs/claude-code/changelog)
https://www.anthropic.com/claude-code

分享这篇文章

相关文章推荐

> **Agent经济：红杉资本2025 AI峰会释放...**

Agent经济：红杉资本2025 AI峰会释放的超级信号

> **模型上下文协议（MCP）深度解析...**

本文介绍了模型上下文协议（MCP），并对其技术...

> **模型上下文协议（MCP）深度解析...**

本文介绍了模型上下文协议（MCP），并对其技术...