Claude Code SDK

> 使用 Claude Code SDK 构建自定义 AI 代理

为什么使用 Claude Code SDK?

Claude Code SDK 提供了构建生产就绪代理所需的所有构建块：

- **优化的 Claude 集成**：自动提示缓存和性能优化
- **丰富的工具生态系统**：文件操作、代码执行、网络搜索和 MCP 扩展性
- **高级权限**：对代理功能的细粒度控制
- **生产必需品**：内置错误处理、会话管理和监控

您可以使用 SDK 构建什么?

以下是您可以创建的一些示例代理类型：

编码代理：

- 诊断和修复生产问题的 SRE 代理
- 审计代码漏洞的安全审查机器人
- 分类事件的值班工程助手
- 执行风格和最佳实践的代码审查代理

业务代理：

- 审查合同和合规性的法律助手
- 分析报告和预测的财务顾问

- 解决技术问题的客户支持代理
- 为营销团队提供的内容创建助手

SDK 目前在 TypeScript 和 Python 中可用，并提供命令行界面 (CLI) 用于快速原型设计。

快速开始

在 5 分钟内让您的第一个代理运行：

1. **安装 SDK**

   ### 命令行

   从 NPM 安装 `@anthropic-ai/claude-code`：

   ```bash
   npm install -g @anthropic-ai/claude-code
   ```

   ### TypeScript

   从 NPM 安装 `@anthropic-ai/claude-code`：

   ```bash
   npm install -g @anthropic-ai/claude-code
   ```

   ### Python

   从 PyPI 安装 `claude-code-sdk` 并从 NPM 安装 `@anthropic-ai/claude-code`：

   ```bash
   pip install claude-code-sdk
   npm install -g @anthropic-ai/claude-code  # 必需依赖项
   ```

   （可选）安装 IPython 用于交互式开发：

   ```bash
   pip install ipython
   ```

2. **设置您的 API 密钥**

   从 Anthropic Console 获取您的 API 密钥并设置 `ANTHROPIC_API_KEY` 环境变量：

```bash
export ANTHROPIC_API_KEY="your-api-key-here"
```

3. **创建您的第一个代理**

   创建以下示例代理之一：

命令行

```bash
# 创建一个简单的法律助手
claude -p "审查此合同条款的潜在问题：'当事方同意承担无限责任...'" \
  --append-system-prompt "您是一名法律助手。识别风险并提出改进建议。"
```

TypeScript

```ts
// legal-agent.ts
import { query } from "@anthropic-ai/claude-code";

// 创建一个简单的法律助手
for await (const message of query({
  prompt: "审查此合同条款的潜在问题：'当事方同意承担无限责任...'",
  options: {
    systemPrompt: "您是一名法律助手。识别风险并提出改进建议。",
    maxTurns: 2
  }
})) {
  if (message.type === "result") {
    console.log(message.result);
  }
}
```

Python

```python
# legal-agent.py
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def main():
    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是一名法律助手。识别风险并提出改进建议。",
            max_turns=2
```

```
            )
        ) as client:
            # 发送查询
            await client.query(
                "审查此合同条款的潜在问题：'当事方同意承担无限责任...'"
            )

            # 流式传输响应
            async for message in client.receive_response():
                if hasattr(message, 'content'):
                    # 在内容到达时打印流式内容
                    for block in message.content:
                        if hasattr(block, 'text'):
                            print(block.text, end='', flush=True)

if __name__ == "__main__":
    asyncio.run(main())
```

4. **运行代理**

命令行

将上面的命令直接复制并粘贴到您的终端中。

TypeScript

1. 设置项目：

```bash
npm init -y
npm install @anthropic-ai/claude-code tsx
```

2. 在您的 package.json 中添加 `"type": "module"`

3. 将上面的代码保存为 `legal-agent.ts`，然后运行：

```bash
npx tsx legal-agent.ts
```

Python

将上面的代码保存为 `legal-agent.py`，然后运行：

```bash
python legal-agent.py
```

对于 IPython/Jupyter notebooks，您可以直接在单元格中运行代码：

```python
await main()
```

上面的每个示例都创建了一个工作代理，它将：

- 使用 Claude 的推理能力分析提示
- 规划解决问题的多步骤方法
- 使用文件操作、bash 命令和网络搜索等工具执行操作
- 基于分析提供可操作的建议

核心用法

概述

Claude Code SDK 允许您从应用程序中以非交互模式与 Claude Code 进行接口。

命令行

先决条件

- Node.js 18+
- 来自 NPM 的 `@anthropic-ai/claude-code`

基本用法

Claude Code 的主要命令行界面是 `claude` 命令。使用 `--print`（或 `-p`）标志以非交互模式运行并打印最终结果：

```bash
claude -p "分析系统性能" \
  --append-system-prompt "您是一名性能工程师" \
  --allowedTools "Bash,Read,WebSearch" \
  --permission-mode acceptEdits \
  --cwd /path/to/project
```

配置

SDK 利用 Claude Code 中可用的所有 CLI 选项。以下是 SDK 使用的关键选项：

标志	描述	示例
`--print`，`-p`	以非交互模式运行	`claude -p "query"`
`--output-format`	指定输出格式（`text`、`json`、`stream-json`）	`claude -p --output-format json`
`--resume`，`-r`	通过会话 ID 恢复对话	`claude --resume abc123`
`--continue`，`-c`	继续最近的对话	`claude --continue`
`--verbose`	启用详细日志记录	`claude --verbose`
`--append-system-prompt`	附加到系统提示（仅与 `--print` 一起使用）	`claude --append-system-prompt "自定义指令"`

`--allowedTools`	允许工具的空格分隔列表，或
	
	逗号分隔工具列表的字符串
		`claude --allowedTools mcp__slack mcp__filesystem`
		
		`claude --allowedTools "Bash(npm install),mcp__filesystem"`

`--disallowedTools`	拒绝工具的空格分隔列表，或
	
	逗号分隔工具列表的字符串
		`claude --disallowedTools mcp__splunk mcp__github`
		
		`claude --disallowedTools "Bash(git commit),mcp__github"`

`--mcp-config`	从 JSON 文件加载 MCP 服务器	`claude --mcp-config servers.json`
`--permission-prompt-tool`	用于处理权限提示的 MCP 工具（仅与 `--print` 一起使用）	`claude --permission-prompt-tool mcp__auth__prompt`

有关 CLI 选项和功能的完整列表，请参阅 CLI 参考 文档。

TypeScript

先决条件

- Node.js 18+
- 来自 NPM 的 `@anthropic-ai/claude-code`

> ℹ️ **注意**：要查看 TypeScript SDK 源代码，请访问 NPM 上的 `@anthropic-ai/claude-code` 页面。

基本用法

通过 TypeScript SDK 的主要接口是 `query` 函数，它返回一个异步迭代器，在消息到达时流式传输消息：

```ts
import { query } from "@anthropic-ai/claude-code";

for await (const message of query({
  prompt: "分析系统性能",
  abortController: new AbortController(),
  options: {
    maxTurns: 5,
    systemPrompt: "您是一名性能工程师",
    allowedTools: ["Bash", "Read", "WebSearch"]
  }
})) {
  if (message.type === "result") {
    console.log(message.result);
  }
}
```

配置

TypeScript SDK 接受命令行支持的所有参数，以及以下附加选项：

参数	描述	默认值
`abortController`	中止控制器	`new AbortController()`
`cwd`	当前工作目录	`process.cwd()`

参数	描述	默认值
`executable`	要使用的 JavaScript 运行时	在 Node.js 中运行时为 `node`，在 Bun 中运行
`executableArgs`	传递给可执行文件的参数	`[]`
`pathToClaudeCodeExecutable`	Claude Code 可执行文件的路径	与 `@anthropic-ai/claude-code` 一起提供的
`permissionMode`	会话的权限模式	`"default"`（选项：`"default"`、`"acceptEdits"`、`"plan`

Python

先决条件

- Python 3.10+
- 来自 PyPI 的 `claude-code-sdk`
- Node.js 18+
- 来自 NPM 的 `@anthropic-ai/claude-code`

> ℹ️ **注意**：要查看 Python SDK 源代码，请参阅 `claude-code-sdk` 仓库。

> 💡 **提示**：对于交互式开发，请使用 IPython：`pip install ipython`

基本用法

Python SDK 提供两个主要接口：

1. `ClaudeSDKClient` 类（推荐）

最适合流式响应、多轮对话和交互式应用程序：

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def main():
    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是一名性能工程师",
            allowed_tools=["Bash", "Read", "WebSearch"],
            max_turns=5
        )
    ) as client:
        await client.query("分析系统性能")

        # 流式响应
        async for message in client.receive_response():
            if hasattr(message, 'content'):
                for block in message.content:
                    if hasattr(block, 'text'):
                        print(block.text, end='', flush=True)

# 作为脚本运行
asyncio.run(main())

# 或在 IPython/Jupyter 中：await main()
```

SDK 还支持传递结构化消息和图像输入：

python

```python
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async with ClaudeSDKClient() as client:
    # 文本消息
    await client.query("分析此代码的安全问题")

    # 带有图像引用的消息（图像将由 Claude 的 Read 工具读取）
    await client.query("解释 screenshot.png 中显示的内容")

    # 按顺序发送多条消息
    messages = [
        "首先，分析 diagram.png 中的架构图",
        "现在根据图表建议改进",
        "最后，生成实现代码"
    ]

    for msg in messages:
```

```
            await client.query(msg)
            async for response in client.receive_response():
                # 处理每个响应
                pass

    # SDK 通过 Claude 的内置 Read 工具处理图像文件
    # 支持的格式：PNG、JPG、PDF 和其他常见格式
```

> ℹ️ **注意**：此页面上的 Python 示例使用 `asyncio`，但您也可以使用 `anyio`。

2. `query` 函数

用于简单的一次性查询：

python

```python
from claude_code_sdk import query, ClaudeCodeOptions

async for message in query(
    prompt="分析系统性能",
    options=ClaudeCodeOptions(system_prompt="您是一名性能工程师")
):
    if type(message).__name__ == "ResultMessage":
        print(message.result)
```

配置

由于 Python SDK 通过 `ClaudeCodeOptions` 类接受命令行支持的所有参数。

身份验证

Anthropic API 密钥

对于基本身份验证，从 Anthropic Console 检索 Anthropic API 密钥并设置 `ANTHROPIC_API_KEY` 环境变量，如快速开始中所示。

第三方 API 凭据

SDK 还支持通过第三方 API 提供商进行身份验证：

- **Amazon Bedrock**：设置 `CLAUDE_CODE_USE_BEDROCK=1` 环境变量并配置 AWS 凭据
- **Google Vertex AI**：设置 `CLAUDE_CODE_USE_VERTEX=1` 环境变量并配置 Google Cloud 凭据

有关第三方提供商的详细配置说明，请参阅 Amazon Bedrock 和 Google Vertex AI 文档。

多轮对话

对于多轮对话，您可以恢复对话或从最近的会话继续：

命令行

```bash
# 继续最近的对话
claude --continue "现在重构这个以获得更好的性能"

# 通过会话 ID 恢复特定对话
claude --resume 550e8400-e29b-41d4-a716-446655440000 "更新测试"

# 在非交互模式下恢复
claude --resume 550e8400-e29b-41d4-a716-446655440000 "修复所有 linting 问题" -
```

TypeScript

```ts
import { query } from "@anthropic-ai/claude-code";

// 继续最近的对话
for await (const message of query({
  prompt: "现在重构这个以获得更好的性能",
  options: { continueSession: true }
})) {
  if (message.type === "result") console.log(message.result);
}

// 恢复特定会话
for await (const message of query({
  prompt: "更新测试",
  options: {
    resumeSessionId: "550e8400-e29b-41d4-a716-446655440000",
    maxTurns: 3
  }
})) {
  if (message.type === "result") console.log(message.result);
}
```

Python

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions, query

# 方法 1：使用 ClaudeSDKClient 进行持久对话
async def multi_turn_conversation():
    async with ClaudeSDKClient() as client:
        # 第一个查询
        await client.query("让我们重构支付模块")
        async for msg in client.receive_response():
            # 处理第一个响应
            pass

        # 在同一会话中继续
        await client.query("现在添加全面的错误处理")
        async for msg in client.receive_response():
            # 处理继续
            pass

        # 对话上下文在整个过程中保持

# 方法 2：使用带有会话管理的 query 函数
async def resume_session():
    # 继续最近的对话
    async for message in query(
        prompt="现在重构这个以获得更好的性能",
        options=ClaudeCodeOptions(continue_conversation=True)
    ):
        if type(message).__name__ == "ResultMessage":
            print(message.result)

    # 恢复特定会话
    async for message in query(
        prompt="更新测试",
        options=ClaudeCodeOptions(
            resume="550e8400-e29b-41d4-a716-446655440000",
            max_turns=3
        )
    ):
        if type(message).__name__ == "ResultMessage":
            print(message.result)

# 运行示例
asyncio.run(multi_turn_conversation())
```

使用计划模式

计划模式允许 Claude 分析代码而不进行修改，对代码审查和规划更改很有用。

命令行

```bash
claude -p "审查此代码" --permission-mode plan
```

TypeScript

```ts
import { query } from "@anthropic-ai/claude-code";

for await (const message of query({
  prompt: "您的提示在这里",
  options: {
    permissionMode: 'plan'
  }
})) {
  if (message.type === "result") {
    console.log(message.result);
  }
}
```

Python

```python
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async with ClaudeSDKClient(
    options=ClaudeCodeOptions(permission_mode='plan')
) as client:
    await client.query("您的提示在这里")
```

> ℹ️ **注意**：计划模式限制编辑、文件创建和命令执行。有关详细信息，请参阅[权限模式]。

自定义系统提示

系统提示定义您的代理的角色、专业知识和行为。这是您指定要构建的代理类型的地方：

命令行

```bash
# SRE 事件响应代理
claude -p "API 宕机，调查" \
  --append-system-prompt "您是 SRE 专家。系统性地诊断问题并提供可操作的解决方案。"

# 法律文档审查代理
claude -p "审查此合同" \
  --append-system-prompt "您是企业律师。识别风险，建议改进，并确保合规性。"

# 附加到默认系统提示
claude -p "重构此函数" \
  --append-system-prompt "始终包含全面的错误处理和单元测试。"
```

TypeScript

```ts
import { query } from "@anthropic-ai/claude-code";

// S RE 事件响应代理
for await (const message of query({
  prompt: "API 宕机，调查",
  options: {
    systemPrompt: "您是 SRE 专家。系统性地诊断问题并提供可操作的解决方案。",
    maxTurns: 3
  }
})) {
  if (message.type === "result") console.log(message.result);
}

// 附加到默认系统提示
for await (const message of query({
  prompt: "重构此函数",
  options: {
    appendSystemPrompt: "始终包含全面的错误处理和单元测试。",
    maxTurns: 2
  }
})) {
  if (message.type === "result") console.log(message.result);
}
```

Python

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def specialized_agents():
    # 带有流式传输的 SRE 事件响应代理
    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是 SRE 专家。系统性地诊断问题并提供可操作的解决方案。",
            max_turns=3
        )
    ) as sre_agent:
        await sre_agent.query("API 宕机，调查")

        # 流式传输诊断过程
        async for message in sre_agent.receive_response():
            if hasattr(message, 'content'):
                for block in message.content:
                    if hasattr(block, 'text'):
                        print(block.text, end='', flush=True)

    # 带有自定义提示的法律审查代理
    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            append_system_prompt="始终包含全面的错误处理和单元测试。",
            max_turns=2
        )
    ) as dev_agent:
        await dev_agent.query("重构此函数")

        # 收集完整响应
        full_response = []
        async for message in dev_agent.receive_response():
            if type(message).__name__ == "ResultMessage":
                print(message.result)

asyncio.run(specialized_agents())
```

高级用法

通过 MCP 的自定义工具

模型上下文协议 (MCP) 让您为代理提供自定义工具和功能。这对于构建需要特定领域集成的专门代理至关重要。

示例代理工具配置：

```json
{
  "mcpServers": {
    "slack": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-slack"],
      "env": {"SLACK_TOKEN": "your-slack-token"}
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-jira"],
      "env": {"JIRA_TOKEN": "your-jira-token"}
    },
    "database": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {"DB_CONNECTION_STRING": "your-db-url"}
    }
  }
}
```

用法示例：

命令行

```bash
# 带有监控工具的 SRE 代理
claude -p "调查支付服务中断" \
  --mcp-config sre-tools.json \
  --allowedTools "mcp__datadog,mcp__pagerduty,mcp__kubernetes" \
  --append-system-prompt "您是 SRE。使用监控数据诊断问题。"

# 带有 CRM 访问的客户支持代理
claude -p "帮助解决客户工单 #12345" \
  --mcp-config support-tools.json \
  --allowedTools "mcp__zendesk,mcp__stripe,mcp__user_db" \
  --append-system-prompt "您是技术支持专家。"
```

TypeScript

ts

```ts
import { query } from "@anthropic-ai/claude-code";

// 带有监控工具的 SRE 代理
for await (const message of query({
  prompt: "调查支付服务中断",
  options: {
    mcpConfig: "sre-tools.json",
    allowedTools: ["mcp__datadog", "mcp__pagerduty", "mcp__kubernetes"],
    systemPrompt: "您是 SRE。使用监控数据诊断问题。",
    maxTurns: 4
  }
})) {
  if (message.type === "result") console.log(message.result);
}
```

Python

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def mcp_enabled_agent():
    # 带有文档访问和流式传输的法律代理
    # 注意：根据需要配置您的 MCP 服务器
    mcp_servers = {
        # 示例配置 - 根据需要取消注释并配置：
        # "docusign": {
        #     "command": "npx",
        #     "args": ["-y", "@modelcontextprotocol/server-docusign"],
        #     "env": {"API_KEY": "your-key"}
        # }
    }

    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            mcp_servers=mcp_servers,
            allowed_tools=["mcp__docusign", "mcp__compliance_db"],
            system_prompt="您是专门从事合同审查的企业律师。",
            max_turns=4
        )
    ) as client:
        await client.query("审查此合同的合规风险")

        # 监控工具使用和响应
```

```
            async for message in client.receive_response():
                if hasattr(message, 'content'):
                    for block in message.content:
                        if hasattr(block, 'type'):
                            if block.type == 'tool_use':
                                print(f"\n[使用工具：{block.name}]\n")
                            elif hasattr(block, 'text'):
                                print(block.text, end='', flush=True)
                        elif hasattr(block, 'text'):
                            print(block.text, end='', flush=True)

                if type(message).__name__ == "ResultMessage":
                    print(f"\n\n审查完成。总成本：${message.total_cost_usd:.4f}")

    asyncio.run(mcp_enabled_agent())
```

> ℹ️ **注意**：使用 MCP 工具时，您必须使用 `--allowedTools` 标志明确允许它们。MCP 工具名称遵循模式 `mcp__<serverName>__<toolName>`，其中：
>
> - `serverName` 是您的 MCP 配置文件中的键
> - `toolName` 是该服务器提供的特定工具
>
> 这种安全措施确保 MCP 工具仅在明确允许时使用。
> 如果您只指定服务器名称（即 `mcp__<serverName>`），将允许该服务器的所有工具。
> 不支持通配符模式（例如 `mcp__go*`）。

自定义权限提示工具

可选地，使用 `--permission-prompt-tool` 传入一个 MCP 工具，我们将使用它来检查用户是否授予模型调用给定工具的权限。当模型调用工具时，会发生以下情况：

1. 我们首先检查权限设置：所有 settings.json 文件，以及传递给 SDK 的 `--allowedTools` 和 `--disallowedTools`；如果其中一个允许或拒绝工具调用，我们继续进行工具调用
2. 否则，我们调用您在 `--permission-prompt-tool` 中提供的 MCP 工具

`--permission-prompt-tool` MCP 工具传递工具名称和输入，并且必须返回带有结果的 JSON 字符串化负载。负载必须是以下之一：

ts

```
// 工具调用被允许
{
```

```
  "behavior": "allow",
  "updatedInput": {...}, // 更新的输入，或只是返回原始输入
}

// 工具调用被拒绝
{
  "behavior": "deny",
  "message": "..." // 解释为什么权限被拒绝的人类可读字符串
}
```

实现示例：

命令行

bash

```
# 与您的 MCP 服务器配置一起使用
claude -p "分析并修复安全问题" \
  --permission-prompt-tool mcp__security__approval_prompt \
  --mcp-config security-tools.json \
  --allowedTools "Read,Grep" \
  --disallowedTools "Bash(rm*),Write"

# 使用自定义权限规则
claude -p "重构代码库" \
  --permission-prompt-tool mcp__custom__permission_check \
  --mcp-config custom-config.json \
  --output-format json
```

TypeScript

ts

```
const server = new McpServer({
  name: "Test permission prompt MCP Server",
  version: "0.0.1",
});

server.tool(
  "approval_prompt",
  '模拟权限检查 - 如果输入包含"allow"则批准，否则拒绝',
  {
    tool_name: z.string().describe("请求权限的工具名称"),
    input: z.object({}).passthrough().describe("工具的输入"),
    tool_use_id: z.string().optional().describe("唯一的工具使用请求 ID"),
  },
  async ({ tool_name, input }) => {
```

```
    return {
      content: [
        {
          type: "text",
          text: JSON.stringify(
            JSON.stringify(input).includes("allow")
              ? {
                  behavior: "allow",
                  updatedInput: input,
                }
              : {
                  behavior: "deny",
                  message: "测试 approval_prompt 工具拒绝权限",
                }
          ),
        },
      ],
    };
  }
);

// 在 SDK 中使用
import { query } from "@anthropic-ai/claude-code";

for await (const message of query({
  prompt: "分析代码库",
  options: {
    permissionPromptTool: "mcp__test-server__approval_prompt",
    mcpConfig: "my-config.json",
    allowedTools: ["Read", "Grep"]
  }
})) {
  if (message.type === "result") console.log(message.result);
}
```

Python

python

```
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def use_permission_prompt():
    """使用自定义权限提示工具的示例"""

    # MCP 服务器配置
    mcp_servers = {
```

```
        # 示例配置 - 根据需要取消注释并配置:
        # "security": {
        #     "command": "npx",
        #     "args": ["-y", "@modelcontextprotocol/server-security"],
        #     "env": {"API_KEY": "your-key"}
        # }
    }

    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            permission_prompt_tool_name="mcp__security__approval_prompt",  #
            mcp_servers=mcp_servers,
            allowed_tools=["Read", "Grep"],
            disallowed_tools=["Bash(rm*)", "Write"],
            system_prompt="您是安全审计员"
        )
    ) as client:
        await client.query("分析并修复安全问题")

        # 监控工具使用和权限
        async for message in client.receive_response():
            if hasattr(message, 'content'):
                for block in message.content:
                    if hasattr(block, 'type'):  # 添加对 'type' 属性的检查
                        if block.type == 'tool_use':
                            print(f"[工具: {block.name}] ", end='')
                    if hasattr(block, 'text'):
                        print(block.text, end='', flush=True)

            # 检查错误消息中的权限拒绝
            if type(message).__name__ == "ErrorMessage":
                if hasattr(message, 'error') and "Permission denied" in str(
                    print(f"\n⚠️ 权限被拒绝: {message.error}")

# 示例 MCP 服务器实现 (Python)
# 这将在您的 MCP 服务器代码中
async def approval_prompt(tool_name: str, input: dict, tool_use_id: str = No
    """自定义权限提示处理程序"""
    # 您的自定义逻辑在这里
    if "allow" in str(input):
        return json.dumps({
            "behavior": "allow",
            "updatedInput": input
        })
    else:
        return json.dumps({
            "behavior": "deny",
```

```
            "message": f"拒绝 {tool_name} 的权限"
        })

asyncio.run(use_permission_prompt())
```

使用说明：

- 使用 `updatedInput` 告诉模型权限提示改变了其输入；否则，将 `updatedInput` 设置为原始输入，如上面的示例所示。例如，如果工具向用户显示文件编辑差异并让他们手动编辑差异，权限提示工具应该返回该更新的编辑。
- 负载必须是 JSON 字符串化的

输出格式

SDK 支持多种输出格式：

文本输出（默认）

命令行

```bash
claude -p "解释文件 src/components/Header.tsx"
# 输出：这是一个显示...的 React 组件
```

TypeScript

```ts
// 默认文本输出
for await (const message of query({
  prompt: "解释文件 src/components/Header.tsx"
})) {
  if (message.type === "result") {
    console.log(message.result);
    // 输出：这是一个显示...的 React 组件
  }
}
```

Python

```python
# 带有流式传输的默认文本输出
async with ClaudeSDKClient() as client:
    await client.query("解释文件 src/components/Header.tsx")

    # 在文本到达时流式传输
    async for message in client.receive_response():
        if hasattr(message, 'content'):
            for block in message.content:
                if hasattr(block, 'text'):
                    print(block.text, end='', flush=True)
                    # 输出实时流式传输：这是一个显示...的 React 组件
```

JSON 输出

返回包括元数据的结构化数据：

命令行

```bash
claude -p "数据层如何工作？" --output-format json
```

TypeScript

```ts
// 收集所有消息以进行类似 JSON 的访问
const messages = [];
for await (const message of query({
  prompt: "数据层如何工作？"
})) {
  messages.push(message);
}

// 访问带有元数据的结果消息
const result = messages.find(m => m.type === "result");
console.log({
  result: result.result,
  cost: result.total_cost_usd,
  duration: result.duration_ms
});
```

Python

python

```python
# 收集所有带有元数据的消息
async with ClaudeSDKClient() as client:
    await client.query("数据层如何工作？")

    messages = []
    result_data = None

    async for message in client.receive_messages():
        messages.append(message)

        # 捕获带有元数据的结果消息
        if type(message).__name__ == "ResultMessage":
            result_data = {
                "result": message.result,
                "cost": message.total_cost_usd,
                "duration": message.duration_ms,
                "num_turns": message.num_turns,
                "session_id": message.session_id
            }
            break

    print(result_data)
```

响应格式：

json

```json
{
  "type": "result",
  "subtype": "success",
  "total_cost_usd": 0.003,
  "is_error": false,
  "duration_ms": 1234,
  "duration_api_ms": 800,
  "num_turns": 6,
  "result": "响应文本在这里...",
  "session_id": "abc123"
}
```

流式 JSON 输出

在接收到每条消息时流式传输：

```bash
$ claude -p "构建应用程序" --output-format stream-json
```

每个对话都以初始的 `init` 系统消息开始，然后是用户和助手消息列表，最后是带有统计信息的最终 `result` 系统消息。每条消息都作为单独的 JSON 对象发出。

消息模式

从 JSON API 返回的消息根据以下模式严格类型化：

```ts
type SDKMessage =
  // 助手消息
  | {
      type: "assistant";
      message: Message; // 来自 Anthropic SDK
      session_id: string;
    }

  // 用户消息
  | {
      type: "user";
      message: MessageParam; // 来自 Anthropic SDK
      session_id: string;
    }

  // 作为最后一条消息发出
  | {
      type: "result";
      subtype: "success";
      duration_ms: float;
      duration_api_ms: float;
      is_error: boolean;
      num_turns: int;
      result: string;
      session_id: string;
      total_cost_usd: float;
    }

  // 作为最后一条消息发出，当我们达到最大轮数时
  | {
      type: "result";
      subtype: "error_max_turns" | "error_during_execution";
      duration_ms: float;
```

```
    duration_api_ms: float;
    is_error: boolean;
    num_turns: int;
    session_id: string;
    total_cost_usd: float;
  }

// 在对话开始时作为第一条消息发出
| {
    type: "system";
    subtype: "init";
    apiKeySource: string;
    cwd: string;
    session_id: string;
    tools: string[];
    mcp_servers: {
      name: string;
      status: string;
    }[];
    model: string;
    permissionMode: "default" | "acceptEdits" | "bypassPermissions" | "pla
  };
```

我们将很快以 JSONSchema 兼容格式发布这些类型。我们对主要 Claude Code 包使用语义版本控制来传达此格式的重大更改。

`Message` 和 `MessageParam` 类型在 Anthropic SDK 中可用。例如，请参阅 Anthropic TypeScript 和 Python SDK。

输入格式

SDK 支持多种输入格式：

文本输入（默认）

命令行

```bash
# 直接参数
claude -p "解释此代码"
```

```
# 从 stdin
echo "解释此代码" | claude -p
```

TypeScript

ts

```ts
// 直接提示
for await (const message of query({
  prompt: "解释此代码"
})) {
  if (message.type === "result") console.log(message.result);
}

// 从变量
const userInput = "解释此代码";
for await (const message of query({ prompt: userInput })) {
  if (message.type === "result") console.log(message.result);
}
```

Python

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient

async def process_inputs():
    async with ClaudeSDKClient() as client:
        # 文本输入
        await client.query("解释此代码")
        async for message in client.receive_response():
            # 处理流式响应
            pass

        # 图像输入（Claude 将自动使用 Read 工具）
        await client.query("这个图表中有什么？screenshot.png")
        async for message in client.receive_response():
            # 处理图像分析
            pass

        # 混合内容的多个输入
        inputs = [
            "分析 diagram.png 中的架构",
            "将其与最佳实践进行比较",
            "生成改进版本"
        ]
```

```
        for prompt in inputs:
            await client.query(prompt)
            async for message in client.receive_response():
                # 处理每个响应
                pass

asyncio.run(process_inputs())
```

流式 JSON 输入

通过 `stdin` 提供的消息流，其中每条消息代表用户轮次。这允许对话的多个轮次而无需重新启动 `claude` 二进制文件，并允许在模型处理请求时向模型提供指导。

每条消息都是一个 JSON '用户消息' 对象，遵循与输出消息模式相同的格式。消息使用 jsonl 格式格式化，其中输入的每一行都是一个完整的 JSON 对象。流式 JSON 输入需要 `-p` 和 `--output-format stream-json` 。

目前这仅限于纯文本用户消息。

bash

```
$ echo '{"type":"user","message":{"role":"user","content":[{"type":"text","t
```

代理集成示例

SRE 事件响应机器人

命令行

bash

```
#!/bin/bash

# 自动化事件响应代理
investigate_incident() {
    local incident_description="$1"
    local severity="${2:-medium}"

    claude -p "事件：$incident_description（严重性：$severity）" \
      --append-system-prompt "您是 SRE 专家。诊断问题，评估影响，并提供即时行动项目。
      --output-format json \
      --allowedTools "Bash,Read,WebSearch,mcp__datadog" \
```

```
      --mcp-config monitoring-tools.json
}

# 用法
investigate_incident "支付 API 返回 500 错误" "high"
```

TypeScript

ts

```ts
import { query } from "@anthropic-ai/claude-code";

// 自动化事件响应代理
async function investigateIncident(
  incidentDescription: string,
  severity = "medium"
) {
  const messages = [];

  for await (const message of query({
    prompt: `事件: ${incidentDescription} (严重性: ${severity}) `,
    options: {
      systemPrompt: "您是 SRE 专家。诊断问题，评估影响，并提供即时行动项目。",
      maxTurns: 6,
      allowedTools: ["Bash", "Read", "WebSearch", "mcp__datadog"],
      mcpConfig: "monitoring-tools.json"
    }
  })) {
    messages.push(message);
  }

  return messages.find(m => m.type === "result");
}

// 用法
const result = await investigateIncident("支付 API 返回 500 错误", "high");
console.log(result.result);
```

Python

python

```python
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions
```

```python
async def investigate_incident(incident_description: str, severity: str = "m
    """带有实时流式传输的自动化事件响应代理"""

    # 监控工具的 MCP 服务器配置
    mcp_servers = {
        # 示例配置 - 根据需要取消注释并配置:
        # "datadog": {
        #     "command": "npx",
        #     "args": ["-y", "@modelcontextprotocol/server-datadog"],
        #     "env": {"API_KEY": "your-datadog-key", "APP_KEY": "your-app-ke
        # }
    }

    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是 SRE 专家。诊断问题，评估影响，并提供即时行动项目。",
            max_turns=6,
            allowed_tools=["Bash", "Read", "WebSearch", "mcp__datadog"],
            mcp_servers=mcp_servers
        )
    ) as client:
        # 发送事件详细信息
        prompt = f"事件: {incident_description} (严重性: {severity}) "
        print(f"🚨 调查中: {prompt}\n")
        await client.query(prompt)

        # 流式传输调查过程
        investigation_log = []
        async for message in client.receive_response():
            if hasattr(message, 'content'):
                for block in message.content:
                    if hasattr(block, 'type'):
                        if block.type == 'tool_use':
                            print(f"[{block.name}] ", end='')
                    if hasattr(block, 'text'):
                        text = block.text
                        print(text, end='', flush=True)
                        investigation_log.append(text)

            # 捕获最终结果
            if type(message).__name__ == "ResultMessage":
                return {
                    'analysis': ''.join(investigation_log),
                    'cost': message.total_cost_usd,
                    'duration_ms': message.duration_ms
                }
```

```
# 用法
result = await investigate_incident("支付 API 返回 500 错误", "high")
print(f"\n\n调查完成。成本: ${result['cost']:.4f}")
```

自动化安全审查

命令行

bash

```bash
# 拉取请求的安全审计代理
audit_pr() {
    local pr_number="$1"

    gh pr diff "$pr_number" | claude -p \
      --append-system-prompt "您是安全工程师。审查此 PR 的漏洞、不安全模式和合规问题。
      --output-format json \
      --allowedTools "Read,Grep,WebSearch"
}

# 用法并保存到文件
audit_pr 123 > security-report.json
```

TypeScript

ts

```ts
import { query } from "@anthropic-ai/claude-code";
import { execSync } from "child_process";

async function auditPR(prNumber: number) {
  // 获取 PR 差异
  const prDiff = execSync(`gh pr diff ${prNumber}`, { encoding: 'utf8' });

  const messages = [];
  for await (const message of query({
    prompt: prDiff,
    options: {
      systemPrompt: "您是安全工程师。审查此 PR 的漏洞、不安全模式和合规问题。",
      maxTurns: 3,
      allowedTools: ["Read", "Grep", "WebSearch"]
    }
  })) {
    messages.push(message);
```

```
  }

  return messages.find(m => m.type === "result");
}

// 用法
const report = await auditPR(123);
console.log(JSON.stringify(report, null, 2));
```

Python

python

```python
import subprocess
import asyncio
import json
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def audit_pr(pr_number: int):
    """带有流式反馈的拉取请求安全审计代理"""
    # 获取 PR 差异
    pr_diff = subprocess.check_output(
        ["gh", "pr", "diff", str(pr_number)],
        text=True
    )

    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是安全工程师。审查此 PR 的漏洞、不安全模式和合规问题。",
            max_turns=3,
            allowed_tools=["Read", "Grep", "WebSearch"]
        )
    ) as client:
        print(f"🔍 审计 PR #{pr_number}\n")
        await client.query(pr_diff)

        findings = []
        async for message in client.receive_response():
            if hasattr(message, 'content'):
                for block in message.content:
                    if hasattr(block, 'text'):
                        # 在发现时流式传输发现
                        print(block.text, end='', flush=True)
                        findings.append(block.text)

            if type(message).__name__ == "ResultMessage":
```

```
                return {
                    'pr_number': pr_number,
                    'findings': ''.join(findings),
                    'metadata': {
                        'cost': message.total_cost_usd,
                        'duration': message.duration_ms,
                        'severity': 'high' if 'vulnerability' in ''.join(fin
                    }
                }

# 用法
report = await audit_pr(123)
print(f"\n\n审计完成。严重性: {report['metadata']['severity']}")
print(json.dumps(report, indent=2))
```

多轮法律助手

命令行

bash

```
# 带有会话持久性的法律文档审查
session_id=$(claude -p "开始法律审查会话" --output-format json | jq -r '.sessic

# 分多个步骤审查合同
claude -p --resume "$session_id" "审查 contract.pdf 的责任条款"
claude -p --resume "$session_id" "检查 GDPR 要求的合规性"
claude -p --resume "$session_id" "生成风险的执行摘要"
```

TypeScript

ts

```
import { query } from "@anthropic-ai/claude-code";

async function legalReview() {
  // 开始法律审查会话
  let sessionId: string;

  for await (const message of query({
    prompt: "开始法律审查会话",
    options: { maxTurns: 1 }
  })) {
    if (message.type === "system" && message.subtype === "init") {
```

```
      sessionId = message.session_id;
    }
  }

  // 使用同一会话的多步骤审查
  const steps = [
    "审查 contract.pdf 的责任条款",
    "检查 GDPR 要求的合规性",
    "生成风险的执行摘要"
  ];

  for (const step of steps) {
    for await (const message of query({
      prompt: step,
      options: { resumeSessionId: sessionId, maxTurns: 2 }
    })) {
      if (message.type === "result") {
        console.log(`步骤: ${step}`);
        console.log(message.result);
      }
    }
  }
}
```

Python

python

```
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

async def legal_review():
    """带有持久会话和流式传输的法律文档审查"""

    async with ClaudeSDKClient(
        options=ClaudeCodeOptions(
            system_prompt="您是企业律师。提供详细的法律分析。",
            max_turns=2
        )
    ) as client:
        # 同一会话中的多步骤审查
        steps = [
            "审查 contract.pdf 的责任条款",
            "检查 GDPR 要求的合规性",
            "生成风险的执行摘要"
        ]
```

```
        review_results = []

        for step in steps:
            print(f"\n📋 {step}\n")
            await client.query(step)

            step_result = []
            async for message in client.receive_response():
                if hasattr(message, 'content'):
                    for block in message.content:
                        if hasattr(block, 'text'):
                            text = block.text
                            print(text, end='', flush=True)
                            step_result.append(text)

                if type(message).__name__ == "ResultMessage":
                    review_results.append({
                        'step': step,
                        'analysis': ''.join(step_result),
                        'cost': message.total_cost_usd
                    })

        # 摘要
        total_cost = sum(r['cost'] for r in review_results)
        print(f"\n\n✅ 法律审查完成。总成本: ${total_cost:.4f}")
        return review_results

# 用法
results = await legal_review()
```

Python 特定最佳实践

关键模式

python

```
import asyncio
from claude_code_sdk import ClaudeSDKClient, ClaudeCodeOptions

# 始终使用上下文管理器
async with ClaudeSDKClient() as client:
    await client.query("分析此代码")
    async for msg in client.receive_response():
        # 处理流式消息
        pass
```

```python
# 并发运行多个代理
async with ClaudeSDKClient() as reviewer, ClaudeSDKClient() as tester:
    await asyncio.gather(
        reviewer.query("审查 main.py"),
        tester.query("为 main.py 编写测试")
    )

# 错误处理
from claude_code_sdk import CLINotFoundError, ProcessError

try:
    async with ClaudeSDKClient() as client:
        # 您的代码在这里
        pass
except CLINotFoundError:
    print("安装 CLI：npm install -g @anthropic-ai/claude-code")
except ProcessError as e:
    print(f"进程错误：{e}")

# 收集带有元数据的完整响应
async def get_response(client, prompt):
    await client.query(prompt)
    text = []
    async for msg in client.receive_response():
        if hasattr(msg, 'content'):
            for block in msg.content:
                if hasattr(block, 'text'):
                    text.append(block.text)
        if type(msg).__name__ == "ResultMessage":
            return {'text': ''.join(text), 'cost': msg.total_cost_usd}
```

IPython/Jupyter 提示

python

```python
# 在 Jupyter 中，直接在单元格中使用 await
client = ClaudeSDKClient()
await client.connect()
await client.query("分析 data.csv")
async for msg in client.receive_response():
    print(msg)
await client.disconnect()

# 创建可重用的辅助函数
async def stream_print(client, prompt):
    await client.query(prompt)
```

```
async for msg in client.receive_response():
    if hasattr(msg, 'content'):
        for block in msg.content:
            if hasattr(block, 'text'):
                print(block.text, end='', flush=True)
```

最佳实践

- **使用 JSON 输出格式**进行响应的程序化解析：

```bash
# 使用 jq 解析 JSON 响应
result=$(claude -p "生成代码" --output-format json)
code=$(echo "$result" | jq -r '.result')
cost=$(echo "$result" | jq -r '.cost_usd')
```

- **优雅地处理错误** - 检查退出代码和 stderr：

```bash
if ! claude -p "$prompt" 2>error.log; then
    echo "发生错误：" >&2
    cat error.log >&2
    exit 1
fi
```

- **使用会话管理**在多轮对话中维护上下文
- **考虑超时**用于长时间运行的操作：

```bash
timeout 300 claude -p "$complex_prompt" || echo "5 分钟后超时"
```

- **尊重速率限制**通过在多个请求之间添加延迟

相关资源

- CLI 用法和控制 - 完整的 CLI 文档
- GitHub Actions 集成 - 使用 Claude 自动化您的 GitHub 工作流程
- 常见工作流程 - 常见用例的分步指南

最后更新于: Thursday, August 21, 2025 at 4:21:20 PM

Previous page
GitHub Actions

Next page
MCP 协议