Claude技能构建完全指南

Claude

目录

引言

技能是一组指令——打包成一个简单的文件夹——用于教导Claude如何处理特定任务或工作流程。技能是为满足您的特定需求定制Claude的最强大方式之一。无需在每次对话中重新解释您的偏好、流程和领域专业知识，技能让您一次性教导Claude，并每次都能受益。

当您拥有可重复的工作流程时，技能会变得非常强大：根据规格说明生成前端设计、采用一致的方法论进行研究、创建遵循团队风格指南的文档，或者编排多步骤流程。它们与Claude内置功能（如代码执行和文档创建）配合良好。对于那些构建MCP集成的人来说，技能增加了另一个强大的层面，有助于将原始工具访问转化为可靠、优化的工作流程。

本指南涵盖了构建有效技能所需了解的一切——从规划和结构到测试和分布。无论您是为自己、团队还是社区构建技能，您都会在整个过程中找到实用的模式和真实世界的示例。

您将学到：

- 技能结构的技术要求和最佳实践
- 独立技能和MCP增强工作流的模式
- 我们在不同用例中观察到的有效模式
- 如何测试、迭代和分发您的技能

适用对象：

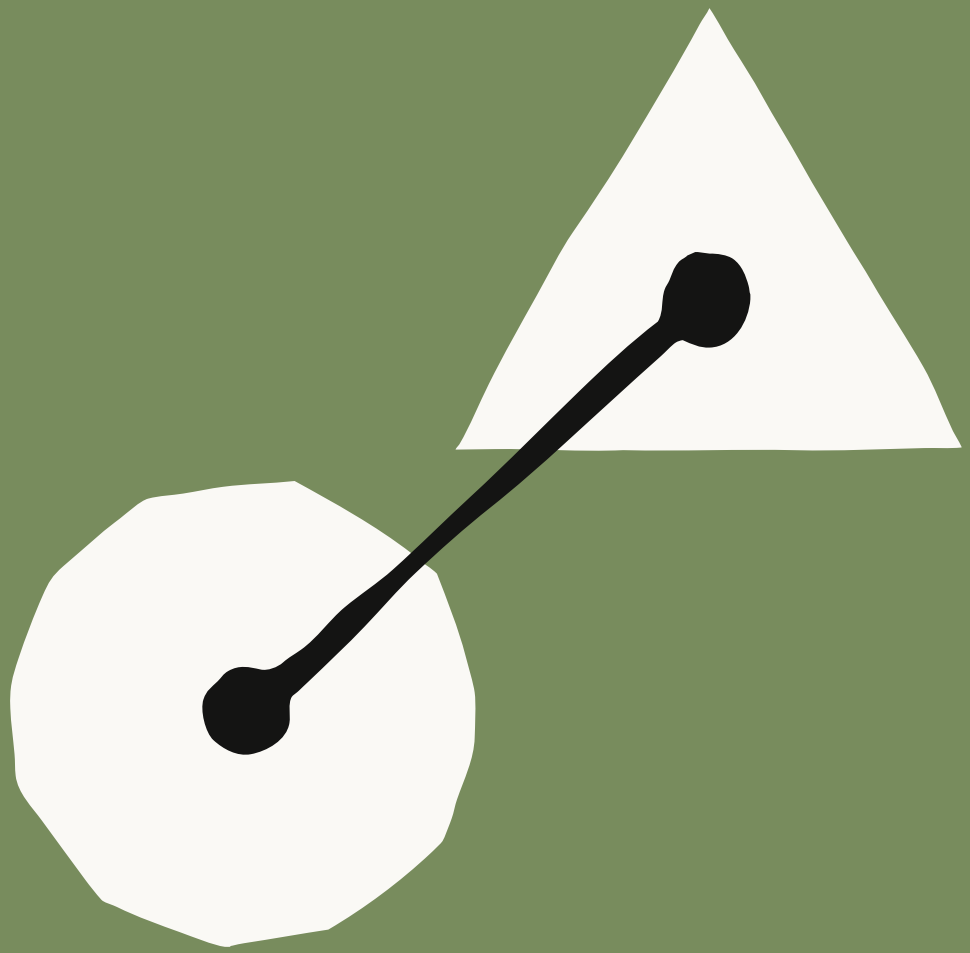
- 希望Claude持续遵循特定工作流程的开发者
- 希望Claude遵循特定工作流程的高级用户
- 希望在整个组织中标准化Claude工作方式的团队

本指南的两条路径

构建独立技能？请关注基础、规划与设计以及类别1-2。增强MCP集成？“技能 + MCP”部分和类别3适合您。两条路径共享相同的技术要求，但您可以选择与您的使用案例相关的内容。

通过本指南你将获得：到最后，你将能够一次性构建一个功能性技能。预计使用技能创建器构建和测试你的第一个工作技能大约需要15-30分钟。

让我们开始吧。



第1章

基础

第1章

基础

什么是技能?

技能是一个包含以下内容的文件夹:

- SKILL.md（必需）：包含YAML前置元数据的Markdown指令
- scripts/（可选）：可执行代码（Python、Bash等）
- references/（可选）：按需加载的文档
- assets/（可选）：输出中使用的模板、字体、图标

核心设计原则

渐进式披露

技能采用三级系统:

- 第一级（YAML前置元数据）：始终加载到Claude的系统提示中。仅提供足够信息，让Claude知道何时应使用每个技能，而无需将所有内容加载到上下文中。

- 第二级（SKILL.md正文）：当Claude认为该技能与当前任务相关时加载。包含完整的指令和指导。

- 第三级（链接文件）：技能目录内捆绑的附加文件，Claude可以根据需要选择性地导航和发现。

这种渐进式披露在保持专业专长的同时，最大限度地减少了令牌使用。

可组合性

Claude可以同时加载多个技能。您的技能应该与其他技能良好协作，而不是假设它是唯一可用的能力。

可移植性

技能在Claude.ai、Claude Code和API中完全一致地工作。创建一次技能，它就可以在所有界面上无需修改即可工作，前提是环境支持该技能所需的任何依赖项。

对于MCP构建者：技能 + 连接器

💡 构建独立技能而无需MCP？请跳转至规划与设计部分——您可以随时返回此处。

如果您已经拥有一个可运行的MCP服务器，那么您已经完成了最困难的部分。技能是位于其上的知识层——用于捕获您已知的工作流程和最佳实践，以便Claude能够一致地应用它们。

厨房类比

MCP提供了专业厨房：可访问工具、食材和设备。

技能提供食谱：关于如何创造有价值事物的分步指令。

它们共同使用户能够完成复杂任务，而无需自己弄清楚每一步。

它们如何协同工作：

MCP（连接性）	技能（知识）
将Claude连接到您的服务（Notion、Asana、Linear等）	教导Claude如何使用您的服务有效地
提供实时数据访问和工具调用	捕获工作流程和最佳实践
Claude 能做什么	Claude 应该如何做

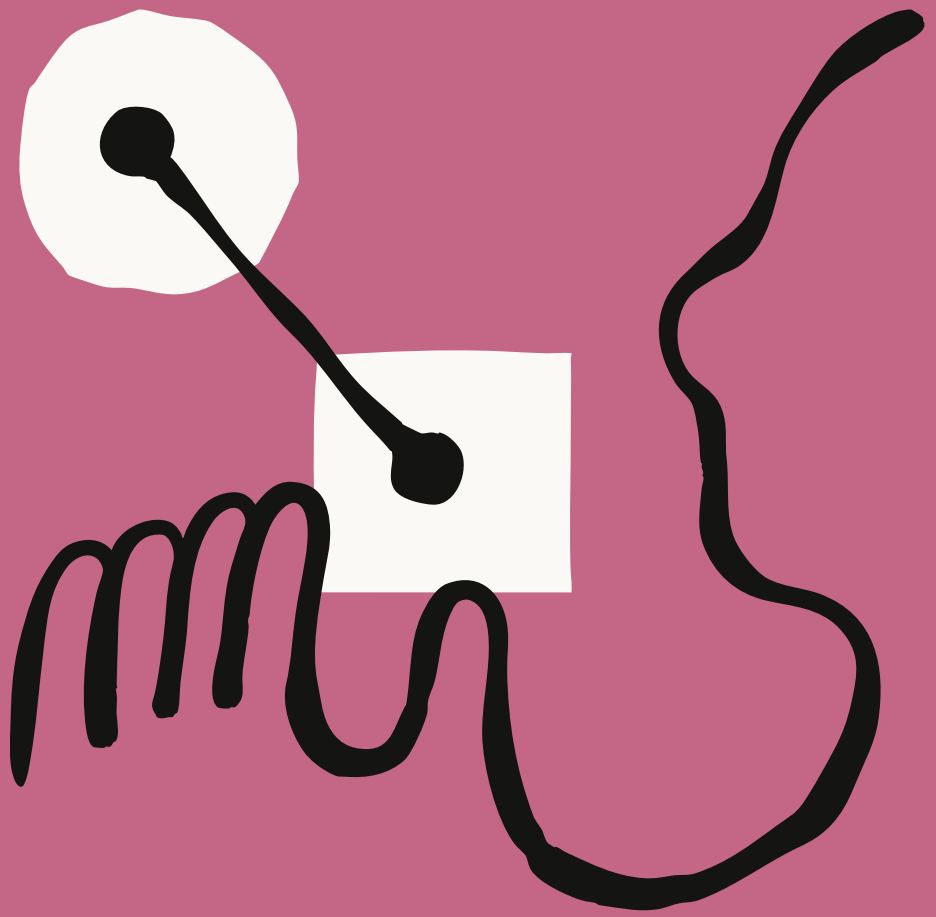
为何这对您的MCP用户至关重要

没有技能：

- 用户连接了您的MCP，但不知道下一步该做什么
- 支持工单询问“如何使用您的集成执行X操作”
- 每次对话都需从头开始
- 由于用户每次提示方式不同，导致不一致的结果
- 当真正的问题在于工作流指导时，用户却责怪你的连接器

借助技能：

- 预构建工作流在需要时自动激活
- 一致、可靠的工具使用
- 最佳实践融入每一次交互
- 降低你的集成学习曲线



第2章

规划与设计

第2章

规划与设计

从用例开始

在编写任何代码之前，先确定您的技能应支持的2-3个具体用例。

良好的用例定义：

```
Use Case: Project Sprint Planning
Trigger: User says "help me plan this sprint" or "create
sprint tasks"
Steps:
1. Fetch current project status from Linear (via MCP)
2. Analyze team velocity and capacity
3. Suggest task prioritization
4. Create tasks in Linear with proper labels and estimates
Result: Fully planned sprint with tasks created
```

问自己：

- 用户想要完成什么?
- 这需要哪些多步骤工作流?
- 需要哪些工具（内置还是MCP？）
- 应嵌入哪些领域知识或最佳实践?

常见技能用例类别

在Anthropic，我们观察到三种常见用例：

类别1：文档与资产创建

用途：创建一致、高质量的输出，包括文档、演示文稿、应用程序、设计、代码等。

真实示例：前端设计技能（另请参阅 docx、pptx、xlsx 和 ppt 的技能）

—

“创建具有高设计质量的独特、生产级前端界面。适用于构建网页组件、页面、制品、海报或应用程序。”

关键技术：

- 嵌入式风格指南和品牌标准
- 用于确保输出一致性的模板结构
- 最终确定前的质量检查清单
- 无需外部工具 - 使用Claude内置功能

类别2：工作流自动化

用途：适用于从一致方法论中受益的多步骤流程，包括跨多个MCP服务器的协调。

真实示例：技能创建器技能

“创建新技能的交互式指南。引导用户完成用例定义、前置内容生成、指令编写和验证。”

关键技术：

- 带验证门的逐步工作流程
- 常见结构模板
- 内置审查和改进建议
- 迭代优化循环

类别3：MCP增强

用途：提供工作流指导，以增强MCP服务器提供的工具访问能力。

真实示例：哨兵代码审查技能（来自哨兵）

“通过其MCP服务器，利用哨兵的错误监控数据，自动分析并修复GitHub拉取请求中检测到的错误。”

关键技术：

- 按顺序协调多个MCP调用
- 嵌入领域专业知识
- 提供用户原本需要指定的上下文
- 常见MCP问题的错误处理

定义成功标准

如何知道你的技能正在发挥作用?

这些是目标设定——粗略的基准而非精确的阈值。力求严谨性，但要接受其中会包含基于直觉的评估元素。我们正在积极开发更稳健的测量指导和工具。

量化指标：

- 技能在90%的相关查询上触发 – 测量方法：运行10-20个应触发您技能的测试查询。追踪其自动加载的次数与需要显式调用的次数。

- 在X次工具调用内完成工作流程 – 测量方法：比较启用技能与未启用技能时执行相同任务的情况。统计工具调用次数和总消耗令牌数。

- 0 每个工作流程中失败的API调用 – 测量方法：在测试运行期间监控MCP服务器日志。追踪重试率和错误代码。

定性指标：

- 用户无需提示Claude下一步操作 – 评估方法：在测试期间，记录您需要重定向或澄清的频率。向测试用户征求反馈。

- 工作流程无需用户纠正即可完成 – 如何评估：运行相同请求3-5次。比较输出的结构一致性和质量。

- 跨会话的一致的结果 - 如何评估：新用户能否在最小指导下首次尝试就完成任务?

技术要求

文件结构

```
your-skill-name/
├── SKILL.md              # Required - main skill file
├── scripts/              # Optional - executable code
│   ├── process_data.py # Example
│   └── validate.sh # Example
├── references/           # Optional - documentation
│   ├── api-guide.md # Example
│   └── examples/ # Example
└── assets/              # Optional - templates, etc.
    └── report-template.md # Example
```

关键规则

SKILL.md 命名:

- 必须完全匹配`SKILL.md`（区分大小写）
- 不接受任何变体（SKILL.MD、skill.md 等）

技能文件夹命名:

- 使用短横线命名法：`notion-project-setup` ✅
- 无空格：`Notion Project Setup` ❌
- 无下划线：`notion_project_setup` ❌
- 无大写字母：`NotionProjectSetup` ❌

无README.md:

- 不要在你的技能文件夹中包含README.md
- 所有文档都应放在SKILL.md或references/中
- 注意：通过GitHub分发时，您仍需要为人类用户准备一个仓库级README——请参阅分发与共享。

YAML前置元数据：最重要的部分

YAML前置元数据是Claude决定是否加载您技能的方式。请确保正确设置。

最小必需格式

```
---
name: your-skill-name
description: What it does. Use when user asks to [specific
phrases].
---
```

这就是开始所需的全部内容。

字段要求

名称（必填）：

- 仅限短横线命名法
- 不能包含空格或大写字母
- 应与文件夹名称匹配

描述（必填）：

- 必须包含以下两项：– 该技能的功能 –
使用时机（触发条件）

- 少于1024个字符
- 无XML标签（< 或>）
- 包含用户可能提到的具体任务
- 如果相关，请提及文件类型

许可证（可选）：

- 如果要将技能开源，请使用
- 常见选项：MIT、Apache-2.0

兼容性（可选）

- 1-500个字符
- 表示环境要求：例如目标产品、必需的系统软件包、网络访问需求等。

元数据（可选）：

- 任何自定义键值对
- 建议：作者、版本、MCP服务器
- 示例：、、、yaml`metadata:author: ProjectHub version: 1.0.0 mcp-server: projecthub`、、、

安全限制

前置内容中禁止包含：

- XML尖括号（< >）
- 名称中包含“Claude”或“Anthropic”的技能（保留）

原因：前置内容会出现在Claude的系统提示中。恶意内容可能注入指令。

编写有效的技能

描述字段

根据Anthropic的工程博客所述：“此元数据……为Claude提供了足够的信息，使其知道何时应使用每个技能，而无需将所有内容加载到上下文中。”这是渐进式披露的第一层级。

结构：

```
[What it does] + [When to use it] + [Key capabilities]
```

良好描述的示例：

```
# Good - specific and actionable
description: Analyzes Figma design files and generates
developer handoff documentation. Use when user uploads .figfiles,
 asks for "design specs", "component documentation", or
"design-to-code handoff".

# Good - includes trigger phrases
description: Manages Linear project workflows including sprint
planning, task creation, and status tracking. Use when user
mentions "sprint", "Linear tasks", "project planning", or asks
to "create tickets".

# Good - clear value proposition
description: End-to-end customer onboarding workflow for
PayFlow. Handles account creation, payment setup, and
subscription management. Use when user says "onboard newcustomer",
 "set up subscription", or "create PayFlow account".
```

不良描述的示例：

```
# Too vague
description: Helps with projects.

# Missing triggers
description: Creates sophisticated multi-page documentation
systems.

# Too technical, no user triggers
description: Implements the Project entity model with
hierarchical relationships.
```

编写主要指令

在前置内容之后，用Markdown编写实际的指令。

推荐的结构：

请根据您的技能调整此模板。将括号内的部分替换为您的具体内容。

```
---
name: your-skill
description: [...]
---

# Your Skill Name

## Instructions

### Step 1: [First Major Step]
Clear explanation of what happens.
Example:
```bash
python scripts/fetch_data.py --project-id PROJECT_ID
Expected output: [describe what success looks like]
```

（根据需要添加更多步骤）

示例

示例 1：[常见场景]

用户说：“设置一个新的营销活动”

操作：

1. 通过 MCP 获取现有活动 2. 使用提供的参数创建
新活动

结果：活动已创建，包含确认链接

（根据需要添加更多示例）

故障排除

错误：[常见错误消息]

原因：[为何发生]

解决方案：[如何修复]

（根据需要添加更多错误案例）
```

指令最佳实践

具体且可操作

✅ 良好示例：

```
Run `python scripts/validate.py --input {filename}` to check
data format.
If validation fails, common issues include:
- Missing required fields (add them to the CSV)
- Invalid date formats (use YYYY-MM-DD)
```

❌ **Bad:**

```
Validate the data before proceeding.
```

包含错误处理

```
## Common Issues

### MCP Connection Failed
If you see "Connection refused":
1. Verify MCP server is running: Check Settings > Extensions
2. Confirm API key is valid
3. Try reconnecting: Settings > Extensions > [Your Service] >
Reconnect
```

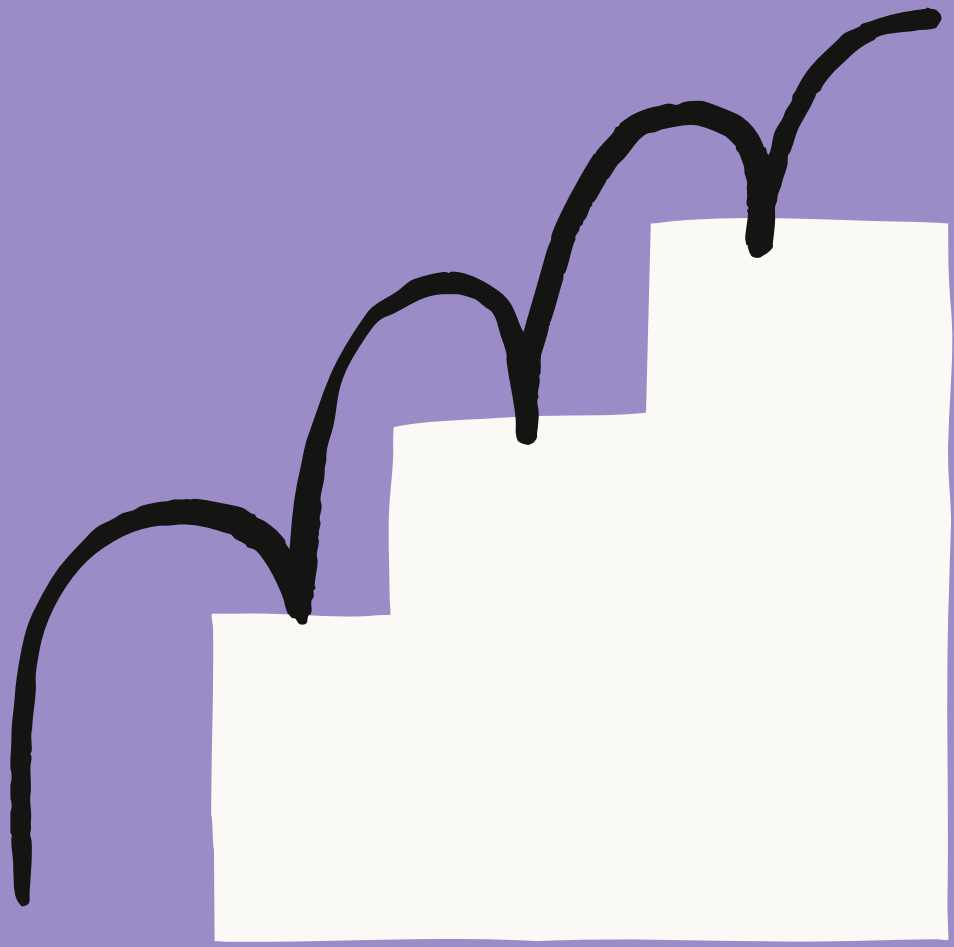
清晰引用捆绑资源

```
Before writing queries, consult `references/api-patterns.md`for:

- Rate limiting guidance
- Pagination patterns
- Error codes and handling
```

使用渐进式披露

保持SKILL.md专注于核心指令。将详细文档移至`references/`并链接到它。（关于三级系统如何运作，请参阅核心设计原则。）



第三章

测试与迭代

第三章

测试与迭代

根据您的需求，技能可以在不同严谨性级别下进行测试：

- 在Claude.ai中进行手动测试 - 直接运行查询并观察行为。快速 t 迭代，无需设置。
- 在Claude Code中进行脚本化测试 - 自动化测试用例，以便在变更中进行可重复的验证。
- 通过技能API进行编程测试 - 构建评估套件，针对定义的测试集进行系统化运行。

选择符合您质量要求和技能可见性的方法。一个由小团队内部使用的技能，其测试需求与部署给数千名企业用户的技能不同。

专业提示：在扩展之前，先在单个任务上进行迭代

我们发现，最高效的技能创建者会针对单个具有挑战性的任务进行迭代，直到Claude成功为止，然后将成功的方法提取为一项技能。这利用了Claude的上下文学习能力，并且比广泛的测试能提供更快的反馈信号。一旦你有了一个有效的基础，就可以扩展到多个测试用例以实现覆盖。

推荐的测试方法

根据早期经验，有效的技能测试通常涵盖三个领域：

1. 触发测试

目标：确保您的技能在正确的时机加载。

测试用例：

- ✅ 在明确任务上触发
- ✅ 在改写请求上触发
- ❌ 在无关主题上不触发

示例测试套件：

```
Should trigger:
- "Help me set up a new ProjectHub workspace"
- "I need to create a project in ProjectHub"
- "Initialize a ProjectHub project for Q4 planning"

Should NOT trigger:
- "What's the weather in San Francisco?"
- "Help me write Python code"
- "Create a spreadsheet" (unless ProjectHub skill handles
sheets)
```

2. 功能测试

目标：验证该技能能产生正确的输出。

测试用例：

- 生成有效输出
- API调用成功
- 错误处理有效
- 覆盖边缘情况

示例：

```
Test: Create project with 5 tasks
Given: Project name "Q4 Planning", 5 task descriptions
When: Skill executes workflow
Then:
  - Project created in ProjectHub
  - 5 tasks created with correct properties
  - All tasks linked to project
  - No API errors
```

3. 性能比较

目标：证明该技能相比基准能改善结果。

使用定义成功标准中的指标。以下是可能的一个比较示例。

基准比较：

```
Without skill:
- User provides instructions each time
- 15 back-and-forth messages
- 3 failed API calls requiring retry- 12,
000 tokens consumed
```

```
With skill:
- Automatic workflow execution
- 2 clarifying questions only
- 0 failed API calls- 6,
000 tokens consumed
```

使用技能创建器技能

该`skill-creator`技能——可通过插件目录在Claude.ai中使用或为Claude Code下载——能帮助您构建和迭代技能。如果您拥有MCP服务器并了解您最重要的2-3个工作流程，您可以在一次会话中构建并测试一个功能性技能——通常只需15-30分钟。

创建技能：

- 根据自然语言描述生成技能
- 生成格式正确的SKILL.md文件（包含前置内容）
- 建议触发短语和结构

审查技能：

- 标记常见问题（模糊描述、缺失触发、结构问题）
- 识别潜在的过度/不足触发风险
- 根据技能所述目的建议测试用例

迭代改进：

- 在使用你的技能并遇到边缘情况或失败后，将这些示例带回给技能创建器
- 示例："利用本次聊天中识别出的问题与解决方案，改进该技能处理[特定边缘情况]的方式"

使用方法：

```
"Use the skill-creator skill to help me build a skill for
[your use case]"
```

注意：技能创建器可帮助您设计和优化技能，但不会执行自动化测试套件或生成定量评估结果。

基于反馈的迭代

技能是活文档。计划基于以下内容进行迭代：

触发不足信号：

- 技能未加载（当它应该加载时）
- 用户手动启用它
- 关于何时使用它的支持问题

> 解决方案：在描述中添加更多细节和细微差别 - 这可能包括特别是针对技术术语的关键词

过度触发信号：

- 技能加载用于无关查询
- 用户禁用它
- 对目的的困惑

> 解决方案：添加负面触发器，更加具体

执行问题：

- 不一致的结果
- API调用失败
- 需要用户纠正

> 解决方案：改进指令，添加错误处理

第4章

分发与共享

第4章

分发与共享

技能使您的MCP集成更加完善。当用户比较连接器时，那些具备技能的连接器提供了更快的价值实现路径，使您在仅MCP替代方案中占据优势。

当前分发模型（2026年1月）

个人用户如何获取技能：

1. 下载技能文件夹 2. 压缩文件夹（如需） 3. 通过设置 > 能力 > 技能上传至Claude.ai 4. 或放置于Claude Code技能目录中

组织级技能：

- 管理员可以部署工作区范围技能（发布于2025年12月18日）
- 自动更新
- 集中管理

一个开放标准

我们已发布智能体技能作为一项开放标准。与MCP一样，我们相信技能应能在不同工具和平台间移植——同一技能应能正常工作无论您使用的是Claude还是其他AI平台。话虽如此，有些技能旨在充分利用特定平台的能力；作者可以在技能的`compatibility`字段中注明这一点。我们一直在与生态系统成员就该标准进行合作，并对早期采用感到兴奋。

通过API使用技能

对于编程用例——例如构建利用技能的应用程序、智能体或自动化工作流程——API提供了对技能管理和执行的直接控制。

关键能力：

- `/v1/skills`用于列出和管理技能的端点
- 通过`container.skills`参数将技能添加到消息API请求中
- 通过Claude控制台进行版本控制和管理
- 适用于Claude代理SDK，用于构建自定义代理

何时通过API与Claude.ai使用技能：

使用场景	最佳界面
最终用户直接与技能交互	Claude.ai / Claude Code
开发过程中的手动测试与迭代	Claude.ai / Claude Code
个体化、临时工作流	Claude.ai / Claude Code
以编程方式使用技能的应用程序	API
大规模生产部署	API
自动化流水线和代理系统	API

注意：API中的技能需要代码执行工具测试版，该工具提供了技能运行所需的安全环境。

有关实现细节，请参阅：

- 技能API快速入门
- 创建自定义技能
- 代理SDK中的技能

推荐方法今日

首先，在GitHub上托管您的技能，使用公共仓库、清晰的自述文件（面向人类访客——这与您的技能文件夹是分开的，技能文件夹不应包含README.md），并提供带有截图的示例用法。然后，在您的MCP文档中添加一个部分，链接到该技能，解释为何同时使用两者具有价值，并提供快速入门指南。

1. 在GitHub上托管

– 开源技能的公共仓库 – 包含安装指令的清晰自述文件 – 示例用法和截图

2. 在您的MCP仓库中记录文档

– 从MCP文档链接到技能 – 解释两者结合使用的价值 – 提供快速入门指南

3. 创建安装指南

```
## Installing the [Your Service] skill

1. Download the skill:
   - Clone repo: `git clone https://github.com/yourcompany/
     skills`
   - Or download ZIP from Releases

2. Install in Claude:
   - Open Claude.ai > Settings > skills
   - Click "Upload skill"
   - Select the skill folder (zipped)

3. Enable the skill:
   - Toggle on the [Your Service] skill
   - Ensure your MCP server is connected

4. Test:
   - Ask Claude: "Set up a new project in [Your Service]"
```

定位您的技能

您如何描述您的技能决定了用户是否理解其价值并真正尝试使用它。在撰写关于您技能的描述时——无论是在您的自述文件、文档还是营销材料中——请牢记以下原则。

关注成果，而非功能：

✅ 良好示例：

```
"The ProjectHub skill enables teams to set up complete project
workspaces in seconds — including pages, databases, and
templates — instead of spending 30 minutes on manual setup."
```

❌ **Bad:**

```
"The ProjectHub skill is a folder containing YAML frontmatter
and Markdown instructions that calls our MCP server tools."
```

突出MCP + 技能故事：

```
"Our MCP server gives Claude access to your Linear projects.
Our skills teach Claude your team's sprint planning workflow.
Together, they enable AI-powered project management."
```

第5章

模式与故障排除

第5章

模式与故障排除

这些模式源自早期采用者和内部团队创建的技能。它们代表了我们观察到的行之有效的常见方法，而非规定性的模板。

选择你的方法：问题优先 vs 工具优先

这就像家得宝一样。你可能带着一个问题走进去——"我需要修理一个厨房橱柜"——然后店员会指引你找到合适的工具。或者，你可能挑选了一个新钻头，然后询问如何将其用于你的具体工作。

技能的工作方式也是如此：

- 问题优先："我需要设置一个项目工作区" → 您的技能以正确的顺序编排恰当的MCP调用。用户描述结果；技能处理工具。

- 工具优先："我已连接Notion MCP" → 您的技能向Claude传授最优工作流程和最佳实践。用户拥有访问权限；技能提供专业知识。

大多数技能倾向于一个方向。了解哪种框架适合您的使用场景，有助于您选择下面的正确模式。

模式1：顺序工作流程编排

使用场景：当您的用户需要按特定顺序执行多步骤流程时。

示例结构：

```
## Workflow: Onboard New Customer

### Step 1: Create Account
Call MCP tool: `create_customer`
Parameters: name, email, company

### Step 2: Setup Payment
Call MCP tool: `setup_payment_method`
Wait for: payment method verification

### Step 3: Create Subscription
Call MCP tool: `create_subscription`
Parameters: plan_id, customer_id (from Step 1)

### Step 4: Send Welcome Email
Call MCP tool: `send_email`
Template: welcome_email_template
```

关键技术：

- 显式步骤排序
- 步骤间依赖关系
- 各阶段验证
- 失败回滚指令

模式2：多MCP协调

使用场景：工作流程跨越多个服务。

示例：设计到开发交接

```
### Phase 1: Design Export (Figma MCP)
1. Export design assets from Figma
2. Generate design specifications
3. Create asset manifest

### Phase 2: Asset Storage (Drive MCP)
1. Create project folder in Drive
2. Upload all assets
3. Generate shareable links

### Phase 3: Task Creation (Linear MCP)
1. Create development tasks
2. Attach asset links to tasks
3. Assign to engineering team

### Phase 4: Notification (Slack MCP)
1. Post handoff summary to #engineering
2. Include asset links and task references
```

关键技术：

- 清晰的阶段分离
- MCP之间的数据传递
- 进入下一阶段前的验证
- 集中式错误处理

模式3：迭代优化

使用场景：输出质量通过迭代得到提升。

示例：报告生成

```
## Iterative Report Creation

### Initial Draft
1. Fetch data via MCP
2. Generate first draft report
3. Save to temporary file

### Quality Check
1. Run validation script: `scripts/check_report.py`
2. Identify issues:
   - Missing sections
   - Inconsistent formatting
   - Data validation errors

### Refinement Loop
1. Address each identified issue
2. Regenerate affected sections
3. Re-validate
4. Repeat until quality threshold met

### Finalization
1. Apply final formatting
2. Generate summary
3. Save final version
```

关键技术：

- 明确的质量标准
- 迭代改进
- 验证脚本
- 知道何时停止迭代

模式4：上下文感知工具选择

使用时机：相同结果，根据上下文使用不同工具。

示例：文件存储

```
## Smart File Storage

### Decision Tree
1. Check file type and size
2. Determine best storage location:
   - Large files (>10MB): Use cloud storage MCP
   - Collaborative docs: Use Notion/Docs MCP
   - Code files: Use GitHub MCP
   - Temporary files: Use local storage

### Execute Storage
Based on decision:
- Call appropriate MCP tool
- Apply service-specific metadata
- Generate access link

### Provide Context to User
Explain why that storage was chosen
```

关键技术：

- 清晰的决策标准
- 备用选项
- 选择透明度

模式5：领域特定智能

使用场景：当您的技能提供超出工具访问范围的专门知识时。

示例：金融合规

```
## Payment Processing with Compliance

### Before Processing (Compliance Check)
1. Fetch transaction details via MCP
2. Apply compliance rules:
   - Check sanctions lists
   - Verify jurisdiction allowances
   - Assess risk level
3. Document compliance decision

### Processing
IF compliance passed:
  - Call payment processing MCP tool
  - Apply appropriate fraud checks
  - Process transaction
ELSE:
  - Flag for review
  - Create compliance case

### Audit Trail
- Log all compliance checks
- Record processing decisions
- Generate audit report
```

关键技术：

- 逻辑中嵌入领域专业知识
- 先合规后行动
- 全面文档
- 清晰的治理

故障排除

技能无法上传

错误："无法在已上传的文件夹中找到SKILL.md"

原因：文件未准确命名为SKILL.md

解决方案：

- 重命名为SKILL.md（区分大小写）
- 使用以下命令验证：ls -la应显示SKILL.md

错误："无效的前置内容"

原因：YAML 格式问题

常见错误：

```
# Wrong - missing delimiters
name: my-skill
description: Does things

# Wrong - unclosed quotes
name: my-skill
description: "Does things

# Correct
---
name: my-skill
description: Does things
---
```

错误："无效的技能名称"

原因：名称包含空格或大写字母

```
# Wrong
name: My Cool Skill

# Correct
name: my-cool-skill
```

技能未触发

症状：技能从未自动加载

Fix:

修改您的描述字段。请参阅描述字段了解好/坏示例。

快速检查清单：

- 它是否过于笼统？（“帮助处理项目”无效）
- 它是否包含用户实际会说的触发短语？
- 如果适用，它是否提及相关的文件类型？

调试方法：

询问Claude：“你什么时候会使用[技能名称] 技能？”Claude会引用描述来回答。根据缺失的内容进行调整。

技能触发过于频繁

症状：技能为不相关的查询加载

解决方案：

1. 添加负面触发器

```
description: Advanced data analysis for CSV files. Use for
statistical modeling, regression, clustering. Do NOT use for
simple data exploration (use data-viz skill instead).
```

2. 更具体一些

```
# Too broad
description: Processes documents

# More specific
description: Processes PDF legal documents for contract review
```

3. 明确范围

```
description: PayFlow payment processing for e-commerce. Use
specifically for online payment workflows, not for general
financial queries.
```

MCP连接问题

症状：技能已加载但MCP调用失败

检查清单：

1. 验证MCP服务器已连接
   – Claude.ai：设置 > 扩展 > [您的服务]– 应显示“已连接”状态

2. 检查身份验证
   – API密钥有效且未过期 – 已授予适当的权限/范围 – OAuth令牌已刷新

3. 独立测试MCP
   – 要求Claude直接调用MCP（不使用技能） – “使用 [服务] MCP来获取我的项目” – 如果失败，问题在于MCP而非技能

4. 验证工具名称
   – 技能参考资料中的MCP工具名称正确 – 检查MCP服务器文档 – 工具名称区分大小写

指令未遵循

症状：技能已加载但Claude不遵循指令

常见原因：

1. 指令过于冗长
   – 保持指令简洁 – 使用项目符号和编号列表
   – 将详细参考移至单独文件

2. 指令被埋没
   – 将关键指令置于顶部 – 使用 ## 重要或 # # 关键标题 – 必要时重复关键点

3. 模糊语言

```
# Bad
Make sure to validate things properly

# Good
CRITICAL: Before calling create_project, verify:
- Project name is non-empty
- At least one team member assigned
- Start date is not in the past
```

高级技巧：对于关键验证，考虑捆绑一个脚本以编程方式执行检查，而不是依赖语言指令。代码是确定性的；语言解释则不然。有关此模式的示例，请参阅Office技能。

4. 模型“惰性” 添加明确鼓励：

```
## Performance Notes
- Take your time to do this thoroughly
- Quality is more important than speed
- Do not skip validation steps
```

注意：将此添加到用户提示中比在SKILL.md中更有效。

大上下文问题

症状：技能似乎变慢或响应质量下降

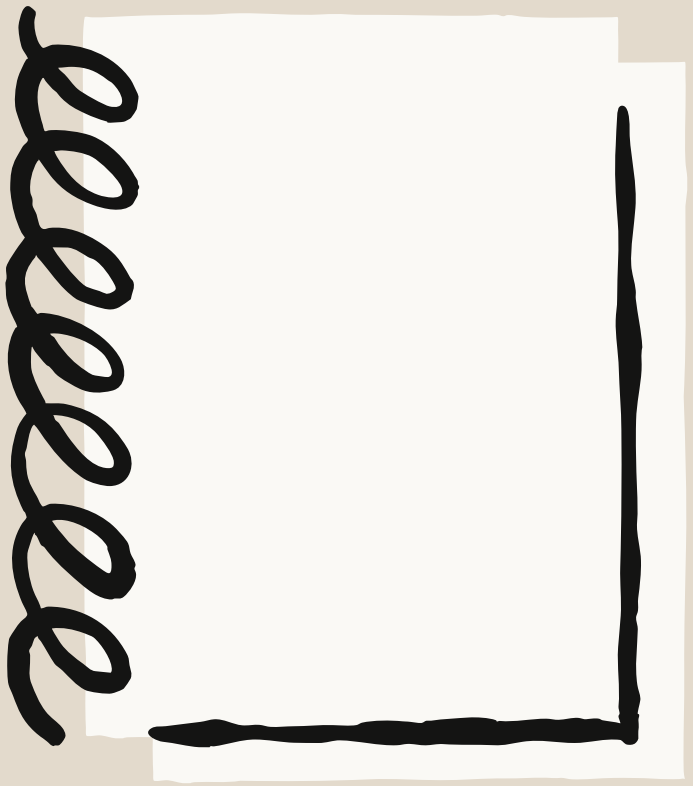
原因：

- 技能内容过大
- 同时启用的技能过多
- 所有内容一次性加载，而非采用渐进式披露

解决方案：

1. 优化SKILL.md文件大小 – 将详细文
档移至references/ – 链接至参考资料而
非内联 – 保持SKILL.md文件在5000字
以内

2. 减少启用的技能
   – 评估是否同时启用了超过20至50个技能 – 建议选择性启用 – 考虑为相
   关功能使用技能“包”



第六章

资源与参考文献

第六章

资源与参考文献

如果您正在构建您的第一个技能，请从最佳实践指南开始，然后根据需要参考API文档。

官方文档

Anthropic资源：

- 最佳实践指南
- 技能文档
- API参考
- MCP文档

博客文章：

- 介绍代理技能
- 工程博客：为现实世界装备代理
- 技能详解
- 如何为Claude创建技能
- 为Claude Code构建技能
- 通过技能改进前端设计

示例技能

公共技能库：

- GitHub：anthropics/skills
- 包含Anthropic创建的技能，您可以进行自定义

工具与实用程序

技能创建器技能：

- 内置于Claude.ai中，并适用于Claude Code
- 可根据描述生成技能
- 提供评审与建议
- 用法：“请使用技能创建器帮我构建一个技能”

验证：

- 技能创建器可以评估您的技能
- 提问：“请审阅此技能并提出改进建议”

获取支持

对于技术问题：

- 一般问题：请访问 Claude 开发者 Discord 社区论坛

对于错误报告：

- GitHub Issues：anthropics/skills/issues
- 包括：技能名称、错误消息、重现步骤

参考A：快速检查清单

使用此清单在技能上传前后验证您的技能。如果您想更快开始，可使用技能创建器技能生成您的初稿，然后运行此清单以确保您没有遗漏任何内容。

开始之前

- [ ] 已确定2-3个具体用例 已确定工具（内置或MCP） 已审阅本指南和示例技能 已规划文件夹结构
- [ ]
- [ ]

开发期间

- [ ] 文件夹名称采用短横线命名法 SKILL.md 文件存在（拼写准确） YAML前置元数据包含 --- 分隔符 名称字段：短横线命名法，无空格，无大写字母 描述包含内容和时机 无XML标签（< >） 指令清晰且可操作 包含错误处理 提供示例 参考资料链接清晰
- [ ]
- [ ]
- [ ]
- [ ]
- [ ]
- [ ]

上传前

- [ ] 测试在明确任务上的触发 测试在改写请求上的触发 验证在无关主题上不触发 功能测试通过 工具集成有效（如适用） 压缩为.zip文件
- [ ]
- [ ]
- [ ]

上传后

- [ ] 在真实对话中测试 监控触发不足/过度触发 收集用户反馈 迭代描述和指令 在元数据中更新版本
- [ ]
- [ ]
- [ ]

参考B：YAML前置元数据

必填字段

```
---
name: skill-name-in-kebab-case
description: What it does and when to use it. Include specific
trigger phrases.
---
```

所有可选字段

```
name: skill-name
description: [required description]
license: MIT # Optional: License for open-source
allowed-tools: "Bash(python:*) Bash(npm:*) WebFetch" # Optional:
Restrict tool access
metadata: # Optional: Custom fields
  author: Company Name
  version: 1.0.0
  mcp-server: server-name
  category: productivity
  tags: [project-management, automation]
  documentation: https://example.com/docs
  support: support@example.com
```

安全说明

允许：

- 任何标准YAML类型（字符串、数字、布尔值、列表、对象）
- 自定义元数据字段
- 长描述（最多1024个字符）

禁止：

- XML尖括号（< >）- 安全限制
- YAML中的代码执行（使用安全YAML解析）
- 以“Claude”或“Anthropic”为前缀命名的技能（保留）

参考C：完整技能示例

如需查看完整、可用于生产环境的技能，以演示本指南中的模式：

- 文档技能 - PDF、DOCX、PPTX、XLSX 创建
- 示例技能 - 各种工作流程模式
- 合作伙伴技能目录 - 查看来自各种合作伙伴的技能，例如 Asana、Atlassian、Canva、Figma、哨兵、Zapier 等

这些仓库保持最新状态，并包含超出此处所涵盖内容的额外示例。克隆它们，根据您的使用场景进行修改，并将其用作模板。

Claude

Claude.ai