

背景

随着 AI 编程工具的快速发展，AI 辅助编程已经从“是否应该使用”的争议转变为“如何更好地使用”的探讨。AI 编程已成为现代软件开发流程中不可或缺的一环。无论是初级开发者还是经验丰富的工程师，及时拥抱 AI 技术都将成为提升工作效率的关键。

本教程将全面介绍 Cursor 编辑器，从基础功能到高级特性，结合实际案例深入探讨如何有效利用 AI 编程。我们的目标是帮助开发者摆脱重复性工作，将更多精力投入到富有创造性的任务中，真正实现开发效率的提升。

下载安装

在 <https://www.cursor.com/> 官网进行下载，下载后安装，目前最新的版本是 0.46.*，带来了许多新的功能，比如：

- UI 更新，将 Agent 作为默认界面
- MCP 工具改进

官网可能没有提供 0.46 版本的下载，需要通过其他渠道下载，比如：windows: <https://anysphere-binaries.s3.us-east-1.amazonaws.com/production/aff57e1d9a74ed627fb5bd393e347079514436a7/win32/x64/user-setup/CursorUserSetup-x64-0.46.0.exe> mac(intel): <https://anysphere-binaries.s3.us-east-1.amazonaws.com/production/aff57e1d9a74ed627fb5bd393e347079514436a7/darwin/x64/Cursor-darwin-x64.zip> mac(apple): <https://anysphere-binaries.s3.us-east-1.amazonaws.com/production/aff57e1d9a74ed627fb5bd393e347079514436a7/darwin/arm64/Cursor-darwin-arm64.zip>

从 VS Code 迁移

如果你是从 VS Code 迁移过来的用户，会发现迁移过程非常简单。Cursor 作为 VS Code 的一个分支，完全兼容 VS Code 的配置。你只需要进入 Cursor 设置 > 常规 > 账户，就可以一键导入 VS Code 的所有配置。

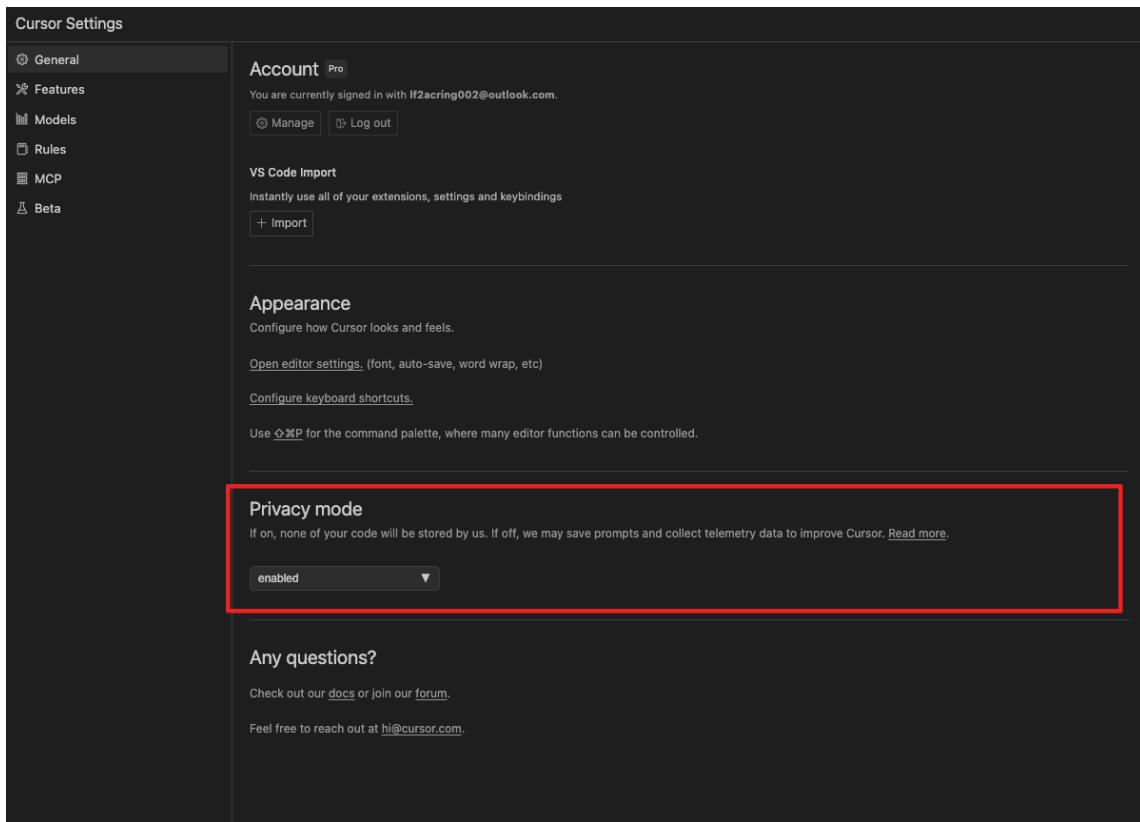
与 VS Code 不同的是，Cursor 没有采用插件机制来扩展功能，而是选择直接修改编辑器的核心代码。这种方式让 Cursor 能够更深度地集成 AI 功能，实现像 Tab 自动补全和 CMD-K 这样的创新特性，这些在传统的插件系统中是难以实现的。

Cursor 与 VS Code 对比

特性	Cursor	VS Code
基础架构	基于VS Code开源版本分支	微软开发的开源编辑器
AI集成	深度集成AI功能	通过插件方式集成AI
Tab自动补全	原生支持	需要插件支持
配置迁移	支持一键导入VS Code配置	原始配置源
插件同步	不支持自动同步插件	支持账户同步插件
隐私模式	支持	有限支持
AI辅助编辑	原生支持(CMD+K)	需要插件支持

⚠ 值得注意的是，Cursor 目前不支持配置的自动同步。如果你需要在多台设备上使用 Cursor，建议先在 VS Code 中安装和同步插件，然后再通过 Cursor 导入这些配置。这样可以确保在所有设备上获得一致的开发体验。

private mode



cursor 支持开启隐私模式,启用** 隐私模式 **后, Cursor 或任何第三方将永远不会存储您的任何代码。否则, 可能会收集提示、代码片段和遥测数据以改进 Cursor。作为企业用户, 我们需要开启隐私模式, 来保护我们的代码不被泄露。

可以在 **Cursor Settings > General > Privacy Mode** (常规) 下启用 隐私模式 **。

Tab 自动补全

我们可以创建一个 `index.js` 文件, 然后输入注释

```
// 快速排序算法
```

然后回车, cursor 就会在下一行自动用灰色补齐出函数名称, 此时按下 tab

cursor 就会自动补齐出完整的快速排序算法代码。这个功能不仅支持算法, 还支持各种常见的代码片段和函数实现。你可以通过继续输入注释来让 cursor 生成更多的代码。

```
1 // 快速排序算法
2 function quickSort(arr) {
```

按下 tab 之后, cursor 则会继续展示剩余可以补全的部分

```
index.js > ⚡ quickSort
1 // 快速排序算法
2 function quickSort(arr) {
    if (arr.length <= 1) return arr;
    const pivot = arr[Math.floor(arr.length / 2)];
    const left = [];
    const right = [];
    for (let i = 0; i < arr.length; i++) {
        if (i === Math.floor(arr.length / 2)) continue;
        if (arr[i] < pivot) left.push(arr[i]);
        else right.push(arr[i]);
    }
    return [...quickSort(left), pivot, ...quickSort(right)];
}
```

只需要再次按下 tab 就可以写完整个函数了

Tab 补全功能是平时使用的最多的功能，可以大大提高编码效率。需要注意的是 Tab 补全使用的是 cursor 自研的模型，而非其他大语言模型，所以容易出现错误。但是相对来说，响应的效率更高，所以如果补全出现错误，不妨调用其他大语言模型来修复。

Tab补全 vs 大语言模型对比

特性	Tab补全(Cursor自研模型)	大语言模型(如Claude/GPT)
响应速度	极快(毫秒级)	相对较慢(秒级)
上下文理解	有限	强大
代码准确性	简单代码准确，复杂代码可能有错	各类复杂代码的准确性更高
资源消耗	低	高
适用场景	快速补全简单代码片段	复杂算法、逻辑、多文件代码生成
错误修复能力	有限，需人工介入	可自我纠错，分析并修复错误
学习能力	固定模型，更新慢	持续学习，模型定期更新
用户交互	简单直接，按Tab即可	需要明确提示，交互更复杂

上下文感知能力

Cursor 的 Tab 补全功能拥有出色的上下文理解能力。它不仅能智能分析当前文件，还会自动将最近打开和访问过的相关文件纳入上下文范围。这意味着当你在编写代码时，如果需要引用其他文件中的类、函数或变量，只要这些文件最近被打开过，Cursor 就能准确理解它们之间的关联，从而提供更精准的补全建议。

这种跨文件的智能上下文分析让开发者在处理复杂项目时能够保持连贯的思路，大大提升了编码效率。无论是调用其他模块的API，还是复用已有的代码逻辑，Cursor 都能基于完整的上下文信息给出恰当的建议。

使用 **ctrl/command + →** 逐字接受补全

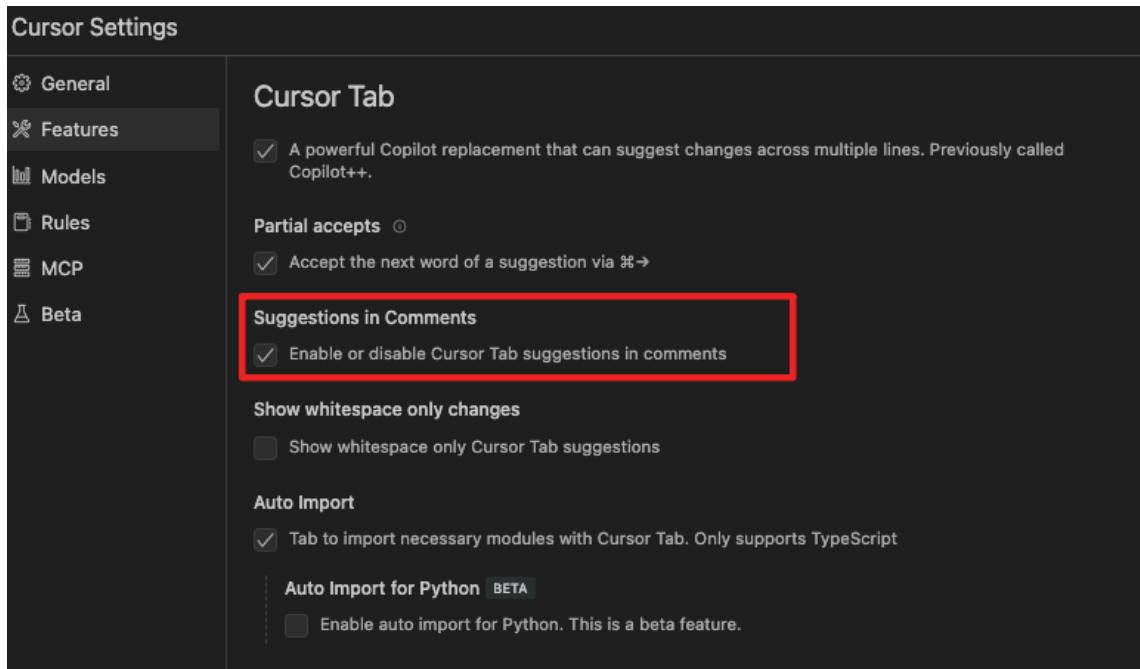
在使用自动补全时，有时我们可能只需要部分建议内容，而不是全部接受。此时，可以使用 **ctrl/command + →** 快捷键来逐字接受补全内容。这个功能允许你一个字符一个字符地确认补全建议，从而精确控制要采纳的内容范围。

这种方法特别适合以下情况：

- 当补全建议部分正确时
- 需要对生成内容进行微调时
- 想要在接受过程中思考代码逻辑时

关闭注释补全

默认情况下，Cursor 在你编写注释时也会提供自动补全功能，这有时可能会干扰你的注释编写流程。如果你希望关闭这一功能，可以通过以下步骤操作：进入 **Cursor Settings > Features > Cursor Tab**，然后取消勾选“Suggestions in comments”选项。这样就可以在保留代码补全的同时，避免在编写注释时出现不必要的补全建议。



在速览中使用 Tab

在速览中使用 Tab 补全是非常方便且强大的功能，当我们在函数名上按下 F12 时，可以快速跳转到所有使用这个函数的地方，我们称之为「速览」，在速览中，Tab 功能同样是生效的，我们可以利用这个特性，快速修改函数及使用位置的参数。

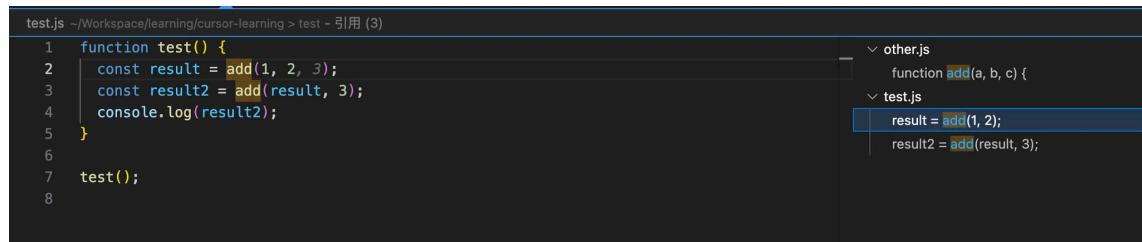
例如，我们有一个 add 函数，将两个参数相加，我们想改成三个参数相加

```
// 加法
function add(a, b) {
  return a + b;
}
```

改为

```
// 加法
function add(a, b, c) {
  return a + b + c;
}
```

此时按下 F12，并跳转到使用的位置



The screenshot shows two files: 'test.js' and 'other.js'. In 'test.js', there is a function definition:

```
test.js ~/Workspace/learning/cursor-learning > test - 引用 (3)
1  function test() {
2    const result = add(1, 2, 3);
3    const result2 = add(result, 3);
4    console.log(result2);
5  }
6
7  test();
8
```

In 'other.js', there is a function definition:

```
other.js
1  function add(a, b, c) {
2    ...
3  }
```

The code completion feature is active in 'other.js', showing suggestions for the 'add' function. The suggestion 'result = add(1, 2);' is highlighted with a blue selection bar.

可以看到，已经自动帮我们补全了第三个参数

这里的基本原理在于 cursor tab 的上下文包含了我们最近的修改和访问的函数，所以如果我们刚修改完函数，此时到任何使用该函数的地方都可以快速补全对应的修改。

快速编辑功能 (Command + K)

通过选中代码块（如整个函数），然后按下 Command + K 快捷键，你可以激活 Cursor 的快速编辑弹窗。在弹窗中输入你的编辑需求，AI 将智能地修改你选中的代码部分。

注意：Command + K 只会处理被选中的代码部分，因此确保完整选中需要修改的函数或代码块非常重要。当你指针位于函数块中时，按下 Command + K 会自动选中整个块。

The screenshot shows a code editor with a dark theme. A cursor suggestion box is open over the code. The box has a title '简化函数' (Simplify Function) and contains three buttons: 'Accept' (highlighted in blue), 'Reject', and 'Follow-up instructions... ⌘K'. The code being edited is a quicksort algorithm:

```
1 // 快速排序算法
2 function quickSort(arr) {
3     if (arr.length <= 1) return arr;
4     const pivot = arr[Math.floor(arr.length / 2)];
5     const left = [];
6     const right = [];
7     for (let i = 0; i < arr.length; i++) {
8         if (arr[i] < pivot) {
9             left.push(arr[i]);
10        } else {
11            right.push(arr[i]);
12        }
13    }
14    const left = arr.filter(x => x < pivot);
15    const right = arr.filter(x => x >= pivot);
16    return [...quickSort(left), pivot, ...quickSort(right)];
17 }
18 }
```

如图所示，Cursor 帮助我们优化了函数写法。你可以通过以下方式应用或取消修改：

- 点击 Accept 按钮或使用 Command + Enter 快捷键接受修改
- 使用 Command + Backspace 快捷键取消修改

如果没有选中任何代码，则默认会新增代码

Command+K 快速提问

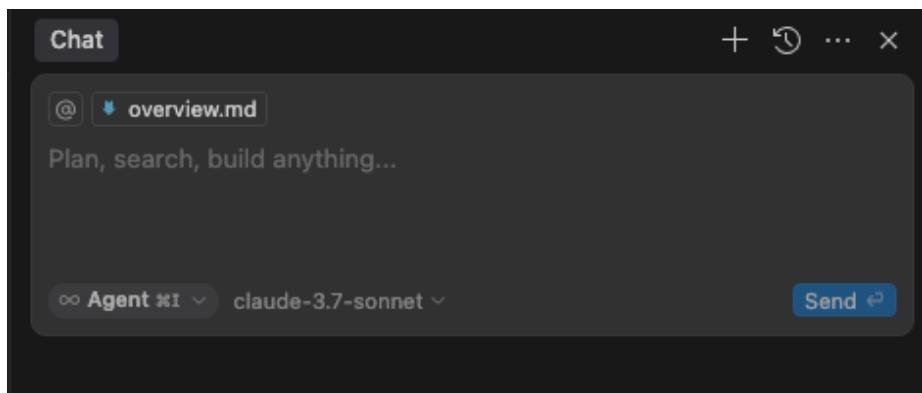
除了编辑代码，我们还可以快速向 AI 提问。在使用 `Command+K` 打开快速编辑弹窗后，按下 `Option+Enter` 组合键即可切换到提问模式。

这种模式下，AI 会给出对代码的解释和分析，而不会修改代码。这对于理解复杂代码逻辑或学习新的编程概念特别有帮助，可以大大提高开发效率。

```
3 | // 冒泡排序算法
4 | 
5 | 空间复杂度是多少
6 | 
7 | 冒泡排序的空间复杂度是 O(1), 因为它只使用了固定的额外空间（几个变量）来完成排序, 不需要额外的数组空间。
8 | 
9 | Follow-up or edit instructions
10| Esc to close
11| 
12| claude-3.5-sonnet ⌘K to toggle
13| 
14| function bubbleSort(arr) {
15|   // 获取数组长度
16|   const len = arr.length;
17| 
18|   // 外层循环控制需要比较的轮数
19|   for (let i = 0; i < len - 1; i++) {
20|     // 内层循环控制每轮比较的次数
21|     for (let j = 0; j < len - 1 - i; j++) {
22|       // 如果前一个元素大于后一个元素, 则交换它们的位置
23|       if (arr[j] > arr[j + 1]) {
24|         // 使用解构赋值进行交换
25|         [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
26|       }
27|     }
28|   }
29| 
30|   // 返回排序后的数组
31|   return arr;
32| }
```

Chat

我们可以用 `cmd + L` 打开聊天框, 来向 AI 询问关于代码的问题, 或者直接当作助手聊天



`cmd + L` 会默认将当前编辑文件作为上下文, 也可以选择其他文件作为上下文

添加上下文

在聊天框中, 我们可以添加不同的上下文, 来帮助 AI 更好地理解我们的需求

这里我们只做一个简单的介绍，如果需要更详细的了解，可以参考 [Cursor 官方文档](#)，后面会对经常用到的再详细的说明

Cursor 上下文引用

在编辑器中使用的各种引用类型及其功能

- @Files - 文件引用
- @Folders - 文件夹引用
- @Code - 代码引用
- @Docs - 文档引用
- @Git - Git相关引用
- @Codebase - 代码库引用
- @Web - 自动搜索互联网
- @Definitions - 附近定义代码引用(目前只支持在 cmd+k 中使用)
- @Link - 粘贴链接，自动获取网页内容
- @LintErrors - 代码错误引用
- @Recent Changes - 最近修改引用
- @Cursor rules - 使用 cursor 规则
- @Summarized Composers - 参考之前 Composer 会话的总结版本

使用AI修复代码

当代码出现错误时，Cursor提供了多种AI辅助修复方式：

1. 悬停修复：将鼠标悬停在报错位置，会出现选项：

- **Fix in Chat**: 在聊天界面中分析错误并自动修复，现在会自动选择 Agent 模式来修复代码错误

```
// 快速排序, 待补充
function quickSort(arr) {
let a = ''
意外的关键字或标识符。 ts(1434)

Fix in Chat (⇧⌘D)
⌘+click to open in new tab
△ Error(TS1434) ✎ | ⏺
意外的关键字或标识符。
any
查看问题 (⌃F8) 快速修复... (⌘.)

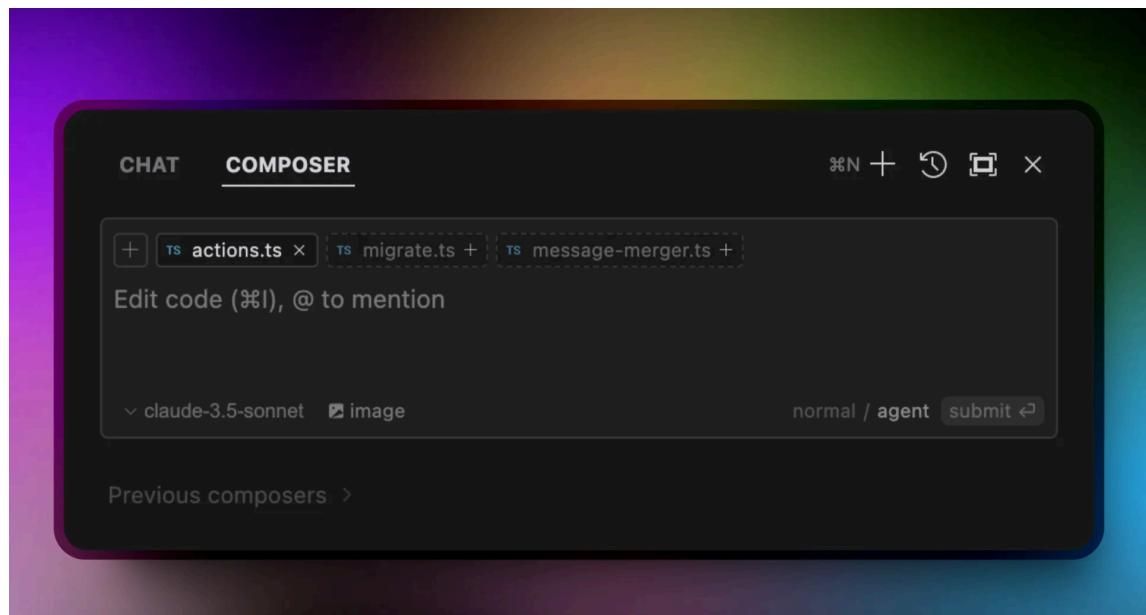
```

误

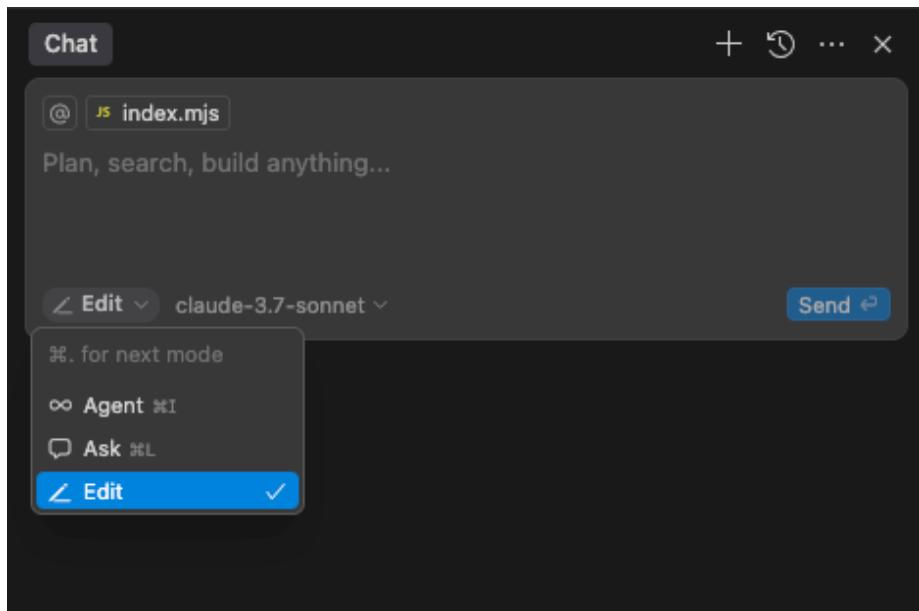
2. 快捷键修复: 在错误代码所在行按下 command/ctrl + ., 选择「Fix with AI」选项, 这将调用快速编辑功能修复代码。这种方式无需打开新窗口, 对简单错误的修复特别高效。

Agent 模式

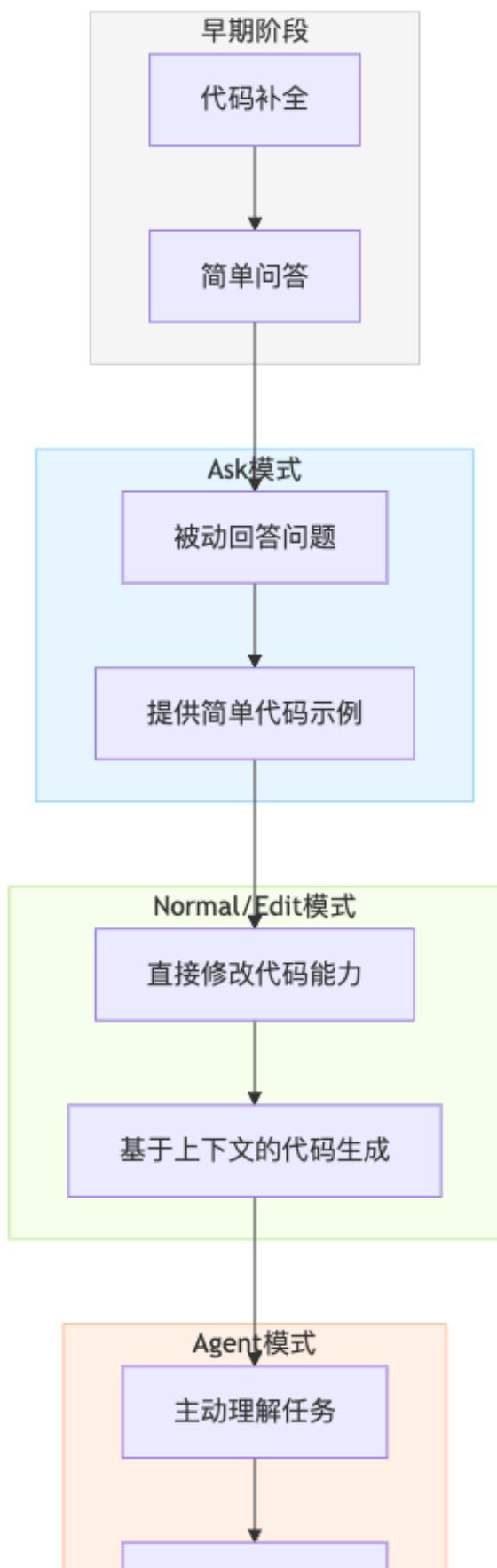
在原来, Cursor 的侧边栏有两种模式, 一种是 Chat 模式, 一种是 Composer 模式, Composer 模式下面又区分 Normal 和 Agent 模式

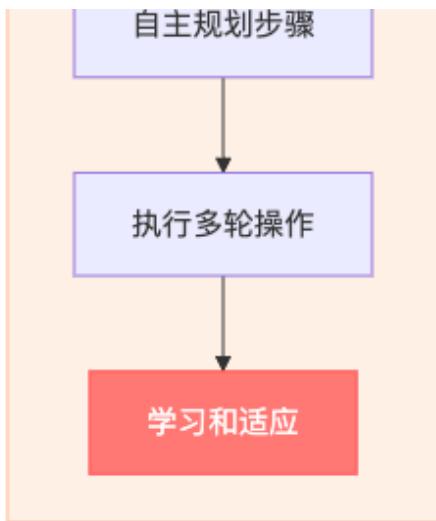


但是在 0.46 版本之后, Cursor 将这两种模式合并, 统一为 Chat 模式, 而 Chat 模式下又区分了 Agent, Ask 和 Edit 模式, 如下图所示



这其实反应了 AI 辅助编程的发展方向。最初的 AI 编程助手主要是被动地回答问题或提供简单的代码补全，也就是 Ask 模式，而现在正在向更主动、更智能的方向发展。Normal 模式是 Ask 模式的增强，但仅仅是拥有了直接修改代码的能力，而 Agent 模式则更进一步，拥有了自主规划和执行的能力。





Agent 模式代表了一种更高级的交互方式，它能够：

1. **主动理解任务**: AI 不再只是回答问题，而是能够理解整个任务的上下文和目标
2. **自主规划步骤**: 能够分解复杂任务，规划解决方案的步骤
3. **执行多轮操作**: 可以进行多步骤的代码编写、调试和优化
4. **学习和适应**: 根据用户的反馈不断调整和改进
5. **调用外部工具**: 可以调用外部工具，比如 命令行, MCP

这种模式下，AI 更像是一个真正的编程助手或合作伙伴，而不仅仅是一个工具。随着大语言模型能力的不断提升，我们可以预见未来 AI 编程助手将更加智能化，能够处理更复杂的编程任务，甚至可能在某些领域达到接近人类程序员的水平。

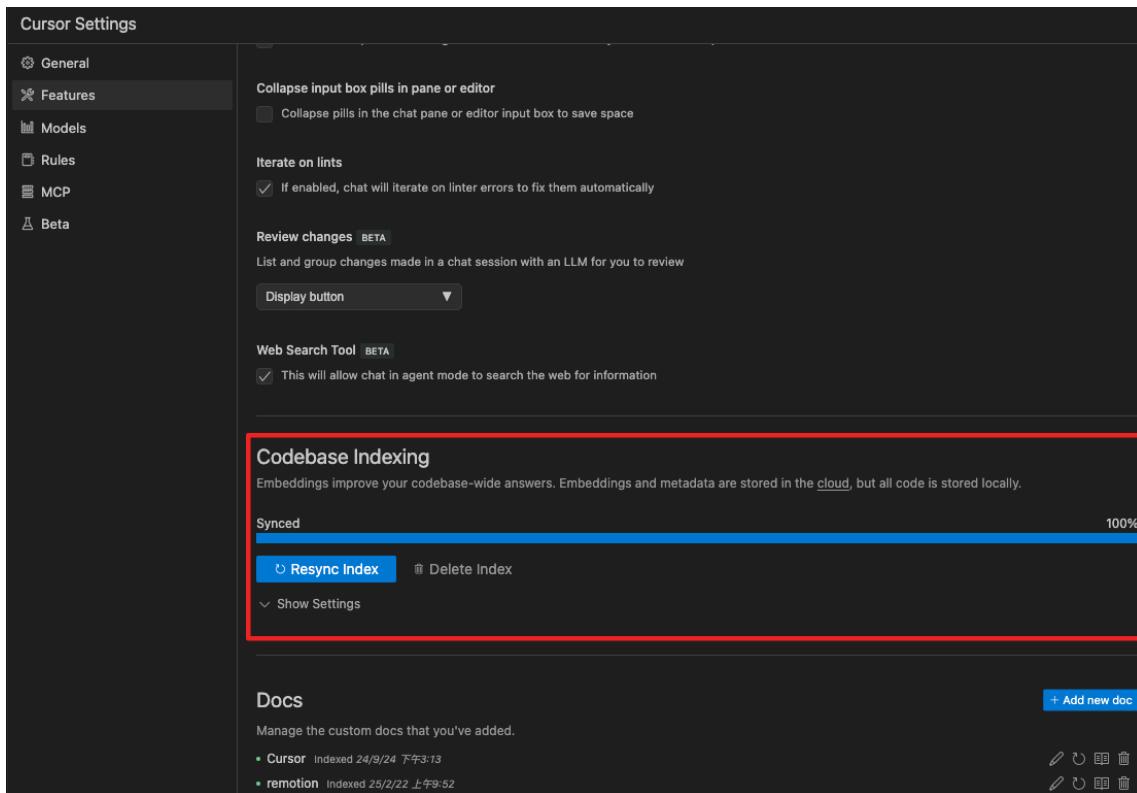
所以如果你还没明白为什么有 Agent 和 Edit 两种模式，你可以直接忽略掉 Edit 模式，因为 Agent 模式已经足够强大，可以满足你大部分的需求。

我们可以通过 `cmd + I` 来快速切换到 Agent 模式，也可以通过 `cmd + L` 来快速切换到 Ask 模式。

代码库索引 (Codebase Indexing)

在深入使用 Cursor 之前，我们需要了解代码库索引的重要性。由于当前所有大语言模型都无法直接处理整个代码库，Cursor 通过建立代码库索引来解决这一问题。这一功能使 AI 能够更全面地理解项目结构和代码关系，从而提供更精准的建议和生成更符合项目风格的代码。

索引状态可以在 `Cursor Settings > Features > Codebase Indexing` 中查看。如果系统没有自动为您的项目创建索引，您可以在此手动触发索引过程：

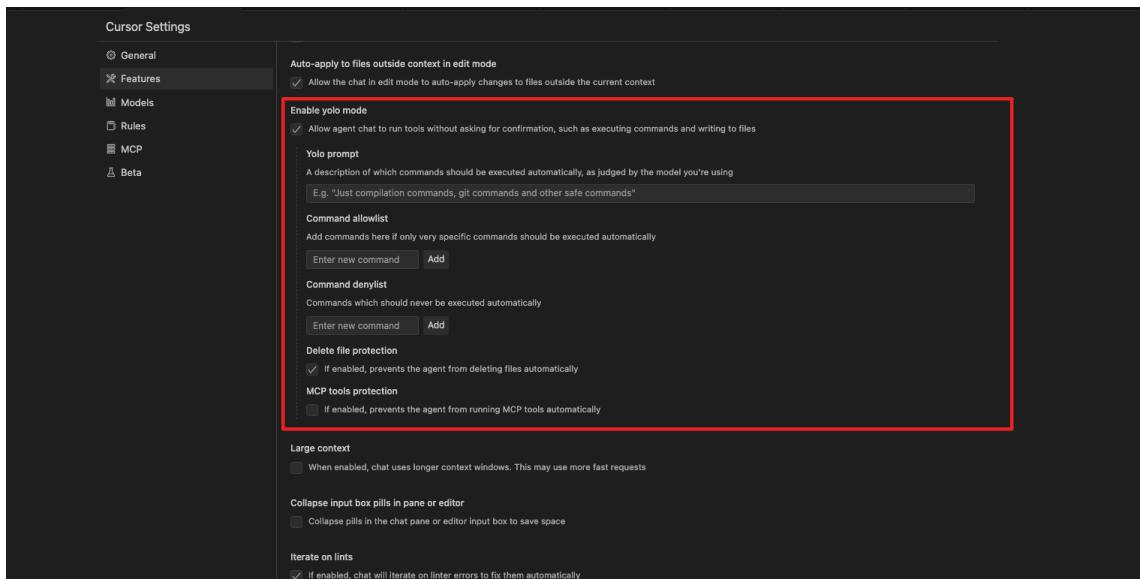


为了提高索引质量和效率，您可以创建 `.cursorignore` 文件，类似于 `.gitignore`，指定哪些文件或目录应被排除在索引之外。

提示：当您发现 Cursor 在 Ask 或 Agent 模式下的表现不如预期时，重新索引代码库通常能解决问题，特别是在项目结构发生较大变化后。

yolo 模式

由于 Agent 是会调用工具，甚至执行命令行的，就有可能会执行一些危险操作，所以默认情况下，Agent 时不允许直接调用工具，或者执行命令行的，而是需要确认后再执行。如果你对 Agent 的时候足够熟悉，或者你想提高 Agent 的使用效率，那么你可以到 Cursor Settings > Features 中开启 yolo 模式，开启后，Agent 会自动调用工具，或者执行命令行，而无需你手动确认。



同时，我们可以细粒度的控制命令行可以执行的命令，比如我们可以配置不允许执行 `rm -rf` 命令，这样即使 Agent 想删除文件，也会被阻止。

使用 Agent 模式

接下来我们会通过一个花店的项目来介绍 Agent 模式的使用过程，以及如何使用 Agent 模式来完成一个复杂的需求。并且几乎不需要手动编写代码。

我已经有一个创建好的 nextjs 项目了（当然你也可以让 Agent 帮我们创建一个 Next.js 项目），他也会调用命令行来初始化项目。

在 Agent 模式下，我们直接可以提出请求：

在当前nextjs项目中初始化一个花店官网，卡片化展示各种花的照片
可以点击卡片进入二级页面，展示花的详细信息，主题色为浅绿色



首先，我将创建一个数据模型和示例数据：

```
ts flowers.ts (new) • × ✓

export interface Flower {
  id: string;
  name: string;
  description: string;
  price: number;
  image: string;
  category: string;
  details: {
    careInstructions: string;
    bloomSeason: string;
    color: string;
    height: string;
  }
}
```

现在，我需要创建花卉图片目录和示例图片。首先，让我创建目录：

```
mkdir -p public/images/flowers
● bllic/images/flowers
  mkdir -p public/images/flowers
○ ~/Workspace/learning/cursor-learning/cursor-learning-2 ➜
```

Pop out terminal

现在，我将创建花卉卡片组件：

```
FlowerCard.tsx (new) • × ✓

import Image from 'next/image';
import Link from 'next/link';
import { Flower } from '@/data/flowers';

interface FlowerCardProps {
  flower: Flower;
}

export default function FlowerCard({ flower }: FlowerCardP
  return (
    <Link href={`/flowers/${flower.id}`} className="group">
      <div className="overflow-hidden rounded-lg shadow-md

```

现在，我将修改首页以展示花卉卡片：

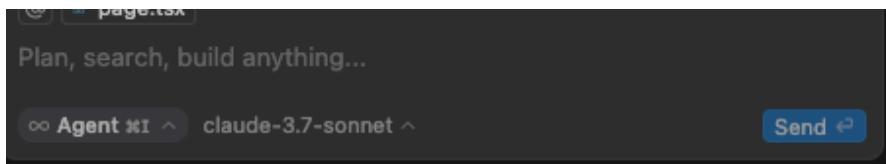
```
page.tsx +77 -89 • × ✓ ⓘ

import Image from "next/image";
import FlowerCard from "@/components/FlowerCard";
```

> Generating..

Stop ⓘ Accept all ⓘ

page.tsx



现在，我们已经完成了花店官网的基本功能。让我们总结一下我们所做的工作：

1. 创建了花卉数据模型和示例数据
2. 创建了花卉卡片组件，用于在首页展示花卉
3. 修改了首页，添加了花店的头部、横幅、花卉展示和页脚
4. 创建了花卉详情页面，展示花卉的详细信息
5. 创建了 404 页面，用于处理找不到花卉的情况
6. 创建了简单的 logo

由于我们使用了图片，但实际上没有真实的花卉图片，所以在实际运行时，图片可能无法正常显示。您需要添加实际的花卉图片到 `/public/images/flowers/` 目录中，或者修改数据模型中的图片路径以指向实际存在的图片。

您现在可以运行项目来查看效果：

```
☒ pnpm dev
```

您可能需要添加一些示例图片，以便能够正常显示花卉卡片。您可以使用任何花卉图片，或者修改数据模型中的图片路径以指向实际存在的图片。

以上就是花店官网的基本实现。您可以根据需要进一步完善和扩展功能，例如：

1. 添加购物车功能
2. 添加用户登录和注册功能
3. 添加搜索和筛选功能
4. 添加更多花卉分类
5. 添加订单管理功能

希望这个实现能够满足您的需求！如果您有任何问题或需要进一步的帮助，请随时告诉我。

```
☒ Review changes
```

↶ ↻ ⏪

可以看到，Agent 会自动调用命令行来生成对应的路由文件夹，卡片组件，以及二级页面组件，最后还会贴心的帮我们启动 dev 服务，让我们可以预览效果。

在执行完毕之后，我们查看结果

The screenshot shows the homepage of a flower shop. At the top, there is a dark header bar with the logo "花之语花店" on the left and navigation links "首页" (Home), "关于我们" (About Us), and "联系方式" (Contact) on the right. Below the header is a large teal-colored promotional banner with the text "美丽花朵，传递情感" (Beautiful flowers, conveying emotion) and a subtext "为您精选各类鲜花，让每一次送花都成为难忘的体验" (We carefully select various types of flowers, making every flower delivery a unforgettable experience). A "了解更多" (Learn more) button is located at the bottom of the banner. Below the banner is a section titled "精选花卉" (Selected Flowers) featuring four flower products: 红玫瑰 (Red Rose), 向日葵 (Sunflower), 郁金香 (Tulip), and 百合花 (Lily). Each product card includes a small image, a name, a brief description, a price (e.g., ¥99 for Red Rose), and a "玫瑰" (Rose) tag.

The screenshot shows a detailed product page for the Red Rose. At the top, there is a back-link "返回首页" (Return to Home) and a header "红玫瑰". The main product image is labeled "红玫瑰". Below the image is the product name "红玫瑰" and a brief description "象征爱情与浪漫的经典红玫瑰，适合送给心爱的人。". The price is listed as "¥99". A prominent green "加入购物车" (Add to Cart) button is located below the price. To the right of the product name is a "玫瑰" (Rose) tag. Below the main product information is a "花卉详情" (Flower Details) section containing tables for color (red), height (30-60 cm), flowering period (spring to autumn), and care instructions (water twice or three times a week, keep soil moist but not waterlogged). At the bottom of the page, there is a section titled "您可能还喜欢" (You might also like) featuring four other flower products: 向日葵 (Sunflower), 郁金香 (Tulip), 百合花 (Lily), and 紫罗兰 (Violet).

可以看到，Agent 已经帮我们生成了一个完整的花店官网，包含主页和二级页面，整体的风格也非常不错，虽然简单，但是作为一个初始项目来说也很够用了。

Restore Checkpoint

在下一步之前，我们要保证任何的操作都是可以恢复的，而 Cursor 提供了非常方便的恢复功能，我们只需要在执行完某个操作之后，点击聊天框中的 `Restore Checkpoint` 按钮，就可以恢复到在这次请求之前的结

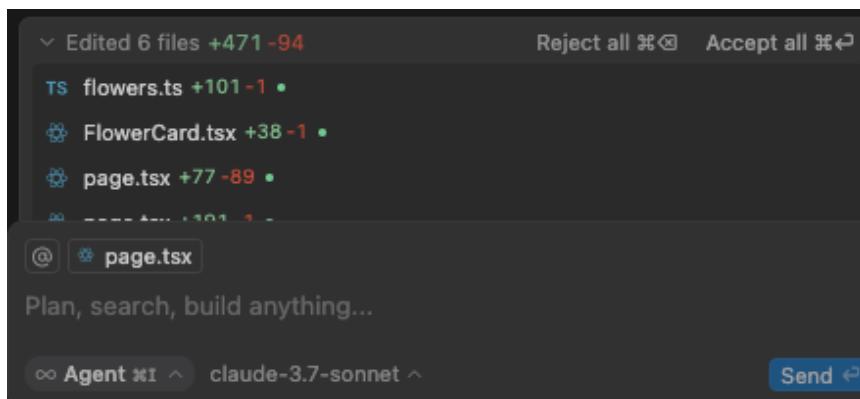
果。



所以我们不用太担心 AI 的修改会出错，可以大胆的让 AI 帮我们修改代码。当然，为了防止 Cursor 自身的 bug 导致我们无法恢复，我们也可以用 Git 来保存我们的代码，这样即使 Cursor 出问题了，我们也可以从 Git 中恢复到之前的状态。

Review

接下来，我们就要开始 Review 代码了



可以看到，本次 AI 生成了 6 个文件，右侧提供了两个快速处理的按钮，分别是 `Accept All` 和 `Reject All`

- `Reject All` 会拒绝所有修改，如果觉得 AI 的修改完全不符合需求，可以点击这个按钮
- `Accept All` 会接受所有修改，如果觉得 AI 的修改完全符合需求，可以点击这个按钮

但是大部份时候，我们可能需要逐个查看 AI 的修改，然后选择接受或者拒绝，这就像是我们平时在进行 Code Review 一样。需要逐行查看，然后选择接受或者拒绝。

此时进入单个页面

A screenshot of a code editor showing a file named 'Home.tsx'. The code is a Next.js component. On the right side of the editor, there are two small buttons: 'Accept' with a green checkmark icon and 'Reject' with a red X icon. The code includes several AI-generated suggestions, such as a logo image with 'dark:invert' class and a 'Save and see your changes instantly' note.

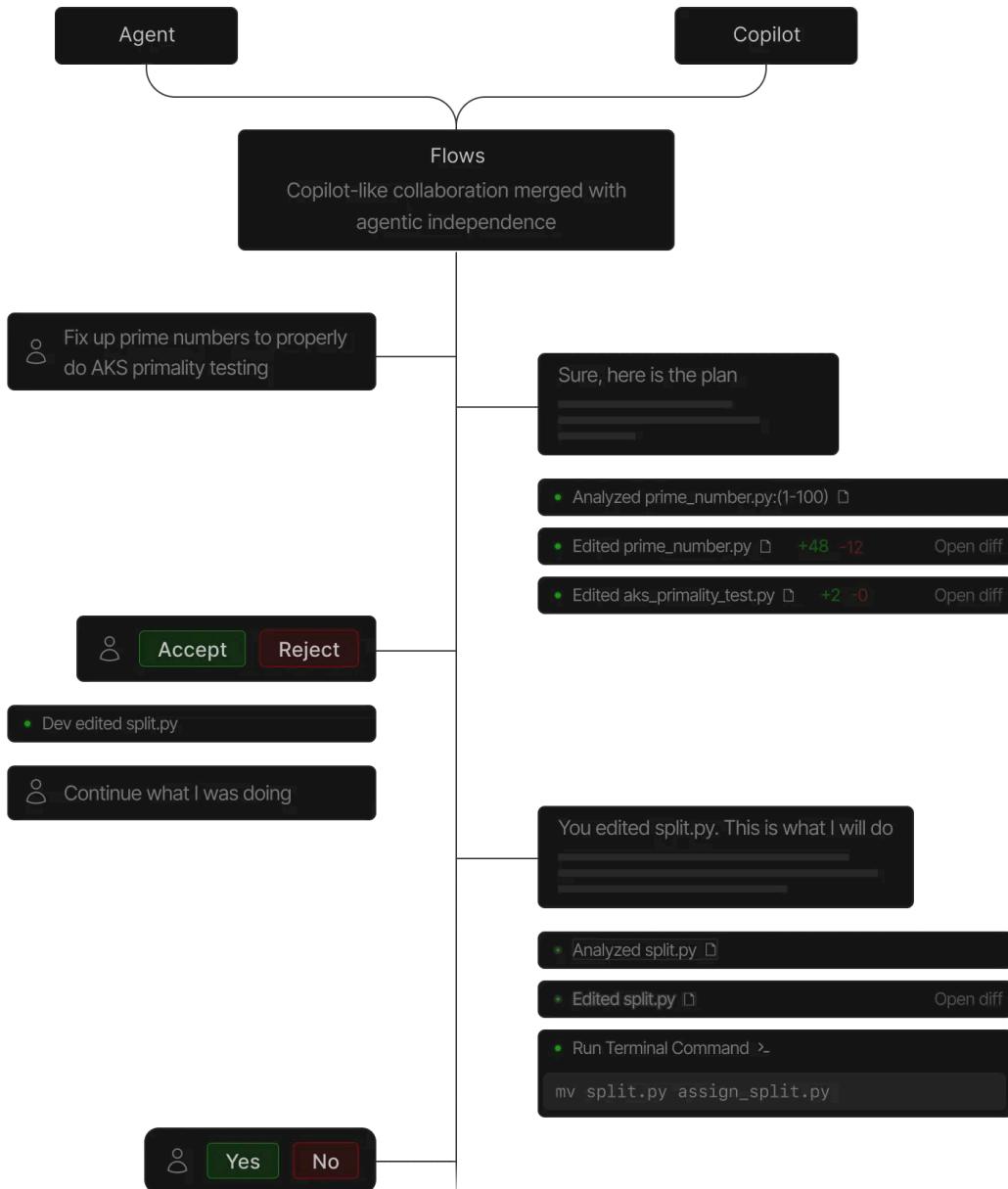
```
1 import Image from "next/image";
2 import FlowerCard from "@/components/FlowerCard";
3 import { flowers } from "@/data/flowers";
4
5 export default function Home() {
6   return (
7     <div className="grid grid-rows-[20px_1fr_20px] items-center justify-items-center min-h-screen p-8 pb-20 gap-16 sm:p-20">
8       <main className="flex flex-col gap-8 row-start-2 items-center sm:items-start">
9         <Image
10           className="dark:invert"
11           src="/next.svg"
12           alt="Next.js logo"
13           width={180}
14           height={38}
15           priority
16         />
17         <ol className="list-inside list-decimal text-sm text-center sm:text-left font-[family-name:var(--font-geist-mono)]">
18           <li className="mb-2">
19             Get started by editing{" "}
20             <code className="bg-black/[.05] dark:bg-white/[.06] px-1 py-0.5 rounded font-semibold">
21               src/app/page.tsx
22             </code>
23             .
24           </li>
25           <li>Save and see your changes instantly.</li>
26         </ol>
27       <div className="min-h-screen bg-gray-50 dark:bg-gray-900">
28         {/* 头部 */}
29         <header className="bg-white dark:bg-gray-800 shadow-sm">
30           <div className="container mx-auto px-4 py-6 flex items-center justify-between">
31             <div className="flex items-center gap-2">
32               <Image
33                 src="/images/logo.svg"
34                 alt="花之语花店"
35                 width={40}
36                 height={40}
37                 className="dark:invert"
38               />
39               <h1 className="text-2xl font-bold text-gray-900 dark:text-white">花之语花店</h1>
40             </div>
41             <nav className="hidden md:flex items-center gap-6">
42               <a href="/" className="text-emerald-600 dark:text-emerald-400 font-medium">首页</a>
43               <a href="#" className="text-gray-600 dark:text-gray-300 hover:text-emerald-600 dark:hover:text-emerald-400">关于
44               <a href="#" className="text-gray-600 dark:text-gray-300 hover:text-emerald-600 dark:hover:text-emerald-400">联系
45             </nav>
46           </div>
47         </header>
48       </div>
49     </main>
50   </div>
51 }
```

可以看到每一个部分的修改，右侧都会提醒可以 Accept 或者 Reject，我们也可以点击 Accept 或者 Reject 按钮来接受或者拒绝修改。当然我更喜欢用快捷键 cmd + Y 来接受修改，cmd + N 来拒绝修改。

当然，此时我们也可以修改 AI 的修改，比如我们认为 AI 的修改有点问题，我们可以自己修改，然后再点击 Accept 按钮来接受修改。

同时在单个文件中，我们也可以用 cmd + Enter 来接受整个文件的修改，或者用 cmd + Backspace 来拒绝整个文件的修改。

最后完成所有代码的 review，我们就算通过与 AI 协作，完成一次功能开发了，当然，其中 AI 的生成占据了大头，我们只需要进行一些微调即可。



上面是一张官方的图，展示了 Agent 模式下，AI 的生成和我们的修改过程。整个流程非常清晰，即：

1. 我们提出需求
2. AI 生成代码
3. 我们进行 Review
4. 我们进行修改
5. 再次提出需求

接下来我们可以继续在同一个聊天框中，继续添加需求，比如我们想添加一个购物车功能，我们只需要继续输入需求即可，AI 会根据我们的需求继续生成代码。

需要注意，由于大模型上下文窗口的限制，如果需求过于复杂，建议新建一个聊天框，来继续添加需求。这样可以让 AI 更好地理解我们的需求。也能保证功能之间的独立性。

例如我们如果想单独修改花卉卡片，我们只需要新建一个聊天框，引用之前的花卉卡片文件，然后继续输入需求即可。

使用 MCP 工具

回到我们的花店网站，这里还有一个问题，就是我们没有每种花的图片，我们希望 AI 能帮我们生成一些图片，但是显然 Cursor 本身没有生成图片的能力，此时我们就可以借助 MCP 工具来生成图片。

什么是 MCP 工具

MCP，全称 [Model Context Protocol](#)，是 Claude 推出的开源协议工具。它标准化了应用程序如何向LLMs提供上下文。将 MCP 想象成 AI 应用的 USB-C 端口。就像 USB-C 提供了一种标准化的方式将您的设备连接到各种外围设备和配件一样，MCP 提供了一种标准化的方式将 AI 模型连接到不同的数据源和工具。

简单来说，就是一个标准化的方式将 AI 模型连接到不同的数据源和工具。

我们可以在 `Cusor Settings > MCP` 中查看和配置 MCP 工具。

The screenshot shows the 'Cursor Settings' interface with the 'MCP' tab selected. The 'MCP Servers' section lists four tools:

- sequential-thinking**: Enabled, Tools: sequentialthinking, No resources available, Command: npx -y @modelcontextprotocol/server-sequential-thinking
- ai image**: Enabled (highlighted with a red box), Tools: image_generator, No resources available, Command: node /Users/acring/Workspace/ai-image-gen-mcp/dist/server.js
- get weather**: Enabled, Tools: get_alerts_for_state, get_forecast_for_location, No resources available, Command: node /Users/acring/Workspace/weather/build/index.js
- vizforge**: Enabled, Tools: generate_chart, list_components, get_component_props, No resources available, Command: node /Users/acring/Workspace/VizForge/dist/mcp/server.js

这里我已经配置好了四个 MCP 的工具，第二个工具的名称是 `ai image`，就是用来生成图片的工具。

使用 MCP 工具生成图片

先演示一下如何使用这个工具，首先我们来到 `src/data/flowers.ts` 文件，这里有所有的花卉的配置信息，包括每个花的图片的位置，当然现在都还是没有图片的，然后我们只需要在聊天框中输入

这里的 图片暂时都还没有，请调用图片生成工具，帮我把这几个图片生成出来，`prompt` 使用英文

Chat + ⏱ ⋮ ×

ts flowers.ts
这里的图片暂时都还没有，请调用图片生成工具，帮我把这几个图片生成出来

我理解您需要为花卉数据中的图片生成实际的图像文件。我将使用图片生成工具为每种花卉创建对应的图片。

首先，我需要创建一个目录来存放这些图片。

```
mkdir -p public/images/flowers
```

```
mkdir -p public/images/flowers
~/Workspace/learning/cursor-learning/cursor-learning-2
~/Workspace/learning/cursor-learning/cursor-learning-2
```

现在我将为每种花卉生成图片。我会按照数据文件中的顺序依次生成。

首先，让我生成红玫瑰的图片：

```
> Called MCP tool image_generator ✓
```

接下来，我将生成向日葵的图片：

```
> Called MCP tool image_generator ✓
```

现在生成郁金香的图片：

```
> Called MCP tool image_generator ✓
```

接下来生成百合花的图片：

```
> Called MCP tool image_generator ✓
```

现在生成紫罗兰的图片：

```
> Called MCP tool image_generator ✓
```

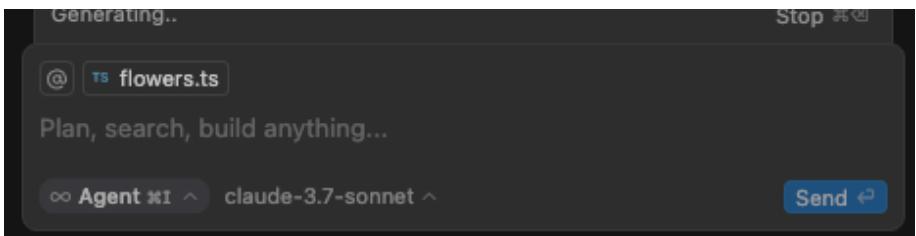
最后生成康乃馨的图片：

```
> Called MCP tool image_generator ✓
```

让我们检查一下所有图片是否已经成功生成：

```
ls -la public/images/flowers | cat
```

```
ing-2 master ls -la public/images/flowers | cat
total 1368
drwxr-xr-x@ 8 acring staff 256 2 27 17:19 .
drwxr-xr-x@ 4 acring staff 128 2 27 16:37 ..
-rw-r--r--@ 1 acring staff 124692 2 27 17:19 carnation.
```



可以看到，AI 会自动识别到上下文中的图片地址，然后调用 MCP 工具来生成图片，并把图片保存到本地。

此时再看网页，可以看到图片已经生成出来了，

美丽花朵，传递情感
为您精选各类鲜花，让每一次送花都成为难忘的体验
了解更多

精选花卉
为您精心挑选的各类鲜花

红玫瑰	向日葵	郁金香	百合花
象征爱情与浪漫的经典红玫瑰，适合送给心爱的人。 ¥99	象征着阳光、希望和忠诚的向日葵，能带来温暖和快乐。 ¥79	优雅高贵的郁金香，象征着完美的爱和祝福。 ¥69	纯洁高雅的百合花，象征着纯洁、高贵和永恒的爱。 ¥89

可以看到，通过 MCP 工具，我们将不在局限于 Cursor 本身的能力，而是可以通过各种外部工具来辅助我们完成任务。

我们可以在 [MCP 的 github 仓库](#) 找到关于 MCP 的很多官方例子，同时也有很多人开源的 MCP 工具，比如 [Figma-Context-MCP](#)

我开发的 MCP

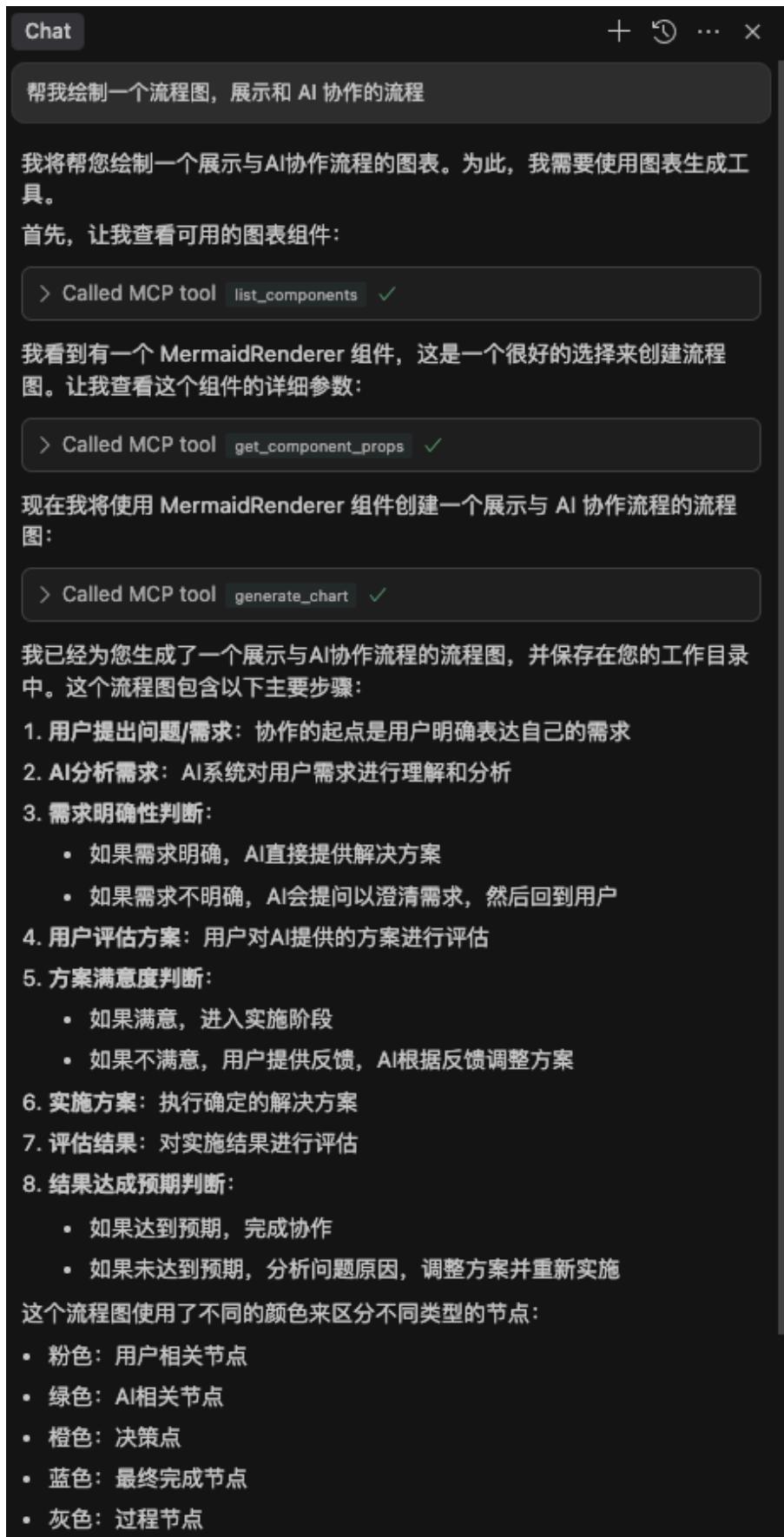
- [VizForge](#) : 在写作过程中利用 LLM 快速生成 图表
- [ai-image-gen-mcp](#) : 利用 LLM 快速生成 图片

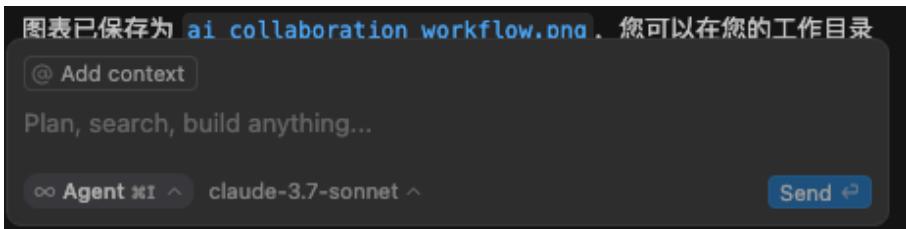
当然，自己实现一个 MCP 工具并不是一件很难的事情，特别是在 AI 的帮助之下，关键在于如何结合自己的业务场景，来实现一个适合自己的 MCP 工具。

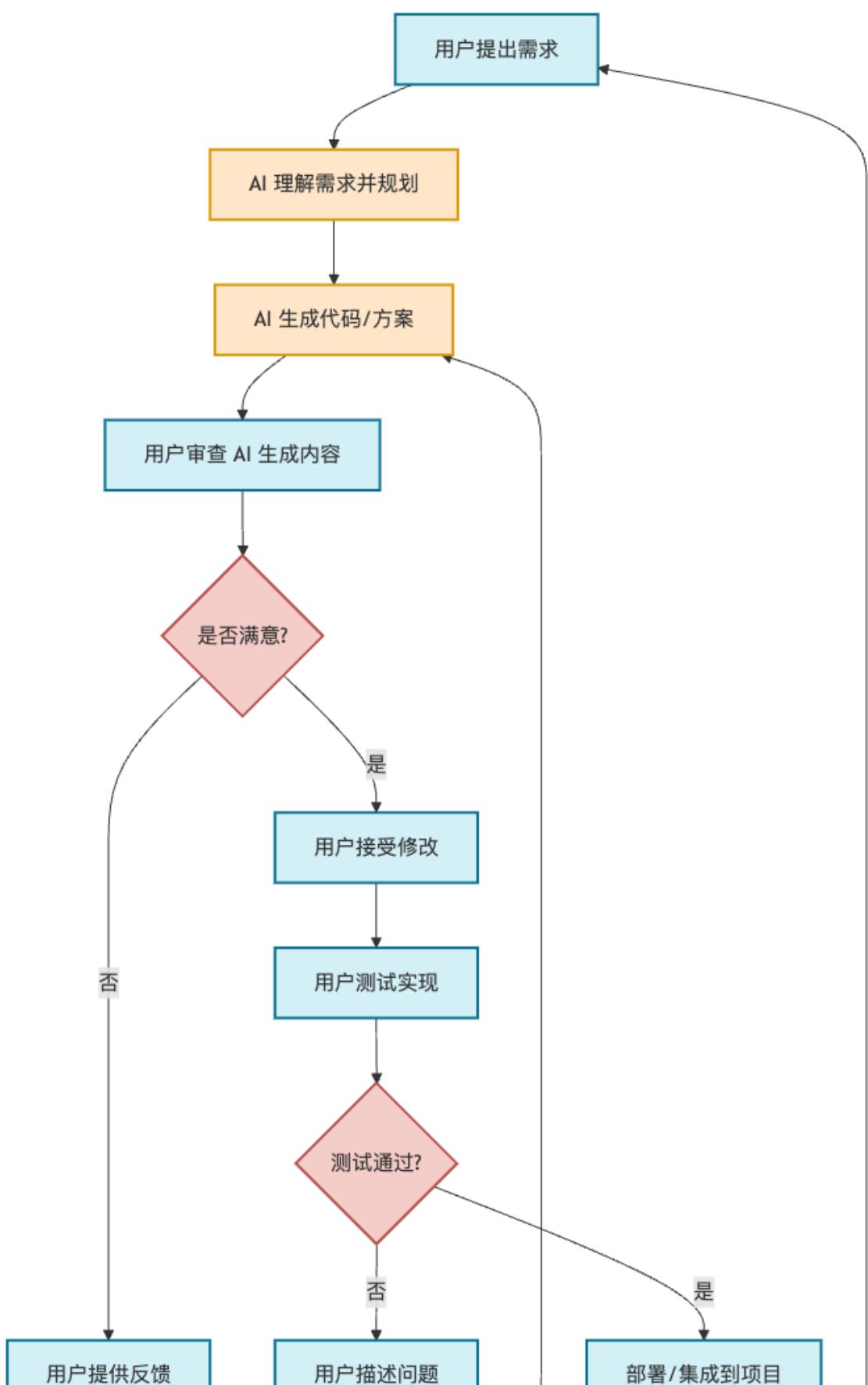
对于 VizForge 来说，我想绘制一个流程图，我只需要在聊天框中输入：

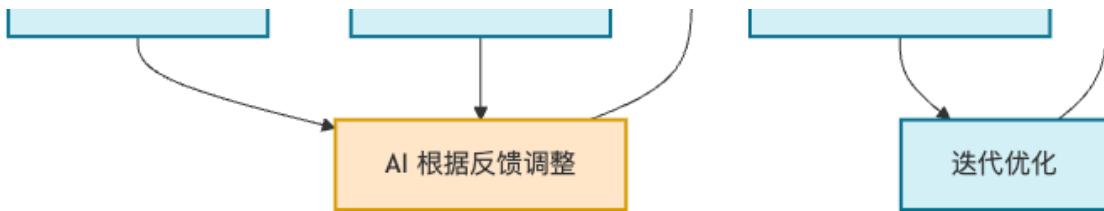
```
帮我绘制一个流程图，展示和 AI 协作的流程
```

AI 就会主动去查找相关的组件，然后调用对应的工具来生成图片，大大提高了效率。









这里由于时间原因，我不过多的介绍 MCP 工具，后续可能会单开一次分享来介绍，总的来说，MCP 工具是一个非常强大的工具，将很多繁琐的步骤集成到平时的日常工作中，并利用大模型来控制比如：

- 我们可以通过 MCP 来和 Figma 文档协作，让 AI 根据 Figma 的文档来自动生成前端的业务组件。
- 我们可以通过 MCP 的 resources，将企业内部的各种文档，比如组件库文档，产品文档，设计文档，技术文档等，集成到 AI 的上下文中，让 AI 根据这些文档来完成任务。

这里就需要靠我们的想象力了。

工作流

虽然我们已经完成了花店官网的初始化开发，但是如果仅仅如此，还完全打不到企业级应用的要求，我们还需要继续完善这个项目。

首先我们需要让项目更加复杂一点，我们将花卉的数据配置继续拆分

将配置进行拆分，花的主要信息和花的详情展示按不同的花，单独放置，
`src/data/flowers/flowerBasics.ts` 文件中放置主要信息，
`src/data/flowers/flowerDetails.ts` 文件中放置详情信息，最后整合在一起

这样，项目的复杂度稍微大了一点，在这种情况下，假如我们需要新增一种花，该怎么办？

即使省去详细的数据，我们也要这样告诉 AI

新增一种花，先叫玫瑰，然后生成一张玫瑰的图片，然后在 `src/data/flowers/flowerBasics.ts` 文件中新增一种花的配置，然后在 `src/data/flowers/flowerDetails.ts` 文件中新增一种花的详情配置

当项目变得复杂之后，每个需求涉及到的代码和目录越来越多，如果每次都需要手动编写，那么效率会非常低。

我们可以用 project rule 来告诉 AI，我们希望它如何完成这个需求。即定义我们的工作流。

Project Rule

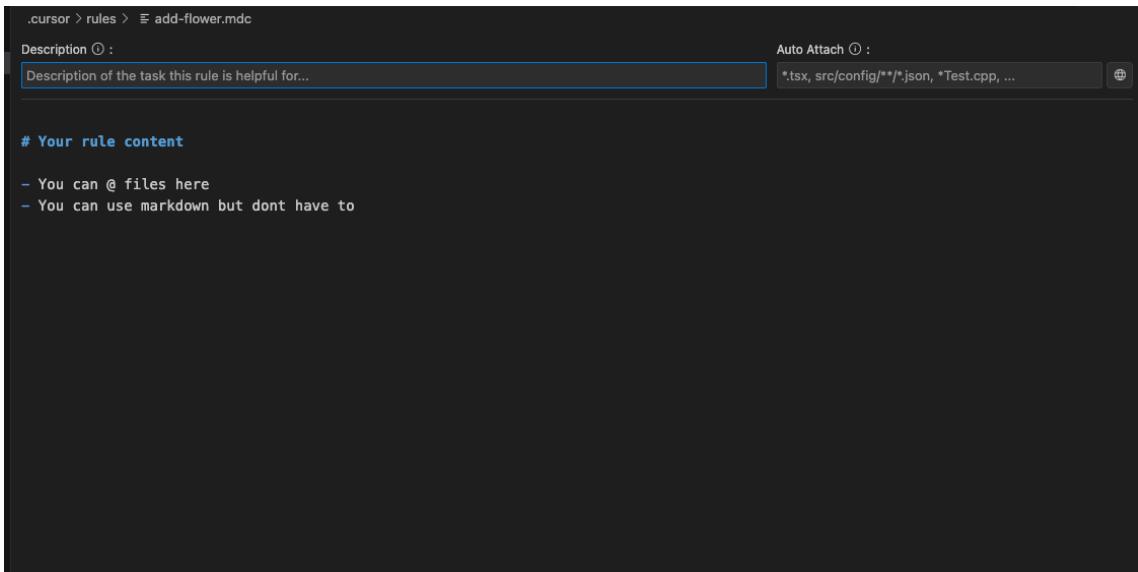
用 Project Rule 定义工作流

Project Rule 是 Cursor 提供的一种项目级别规范，比如定义项目中代码的规范，或者定义项目中的一些约定俗成的规范。

但是不要被 Project Rule 的名称所迷惑，Project Rule 不仅仅局限于代码规范，还可以定义项目中的工作流。

我们先来创建一个 Project Rule，来告诉 AI，我们希望它如何完成这个需求。

通过按下 shift + cmd + P , 然后输入 Project Rule , 然后选择 Create New Rule , 然后输入规则的名称, 名称通常不是很重要。比如这里叫 add-flower



可以看到 rule 有三个部分, 分别是:

- Description
- Auto Attach
- 正文

Description 这个名称也非常的具有迷惑性, 似乎是想描述这个规范, 但查看官方文档, 会发现他的作用是:
The agent can see this description and decide to read the full rule if it wants it.If no description is provided we will only use the attach rules below.

翻译过来就是:

如果提供了描述, 那么 AI 会根据描述来决定是否阅读完整的规则。如果没有任何描述, 那么 AI 只会使用下面的规则。

意思就是, 如果 AI 根据描述觉得用户的需求符合描述, 那么 AI 就会阅读完增的 rule, 所以我们可以这么写

Description:

当用户需要新增一种花的时候, 请按照以下步骤执行:

正文:

1. 根据花的名称, 生成一张花的近景特写, prompt 使用英文
2. 在 src/data/flowers/flowerBasics.ts 文件中新增一种花的配置
3. 在 src/data/flowers/flowerDetails.ts 文件中新增一种花的详情配置

那么当用户提出需求的时候, AI 就会按照这个规则来执行。我们可以试试效果

只需要输入, 想添加一个荷花, 这里我们甚至不需要任何的上下文, 也不需要主动引用规则

Chat + ⏪ ⋮ X

新增一个荷花

↶ Restore checkpoint

我理解您想要新增一个荷花。这看起来与"add-flower"规则相关，我需要先获取这个规则的详细信息。

Rule: add-flower

当用户需要新增一种花的时候，请按照以下步骤执行：

1. 根据花的名称，生成一张花的近景特写，prompt 使用英文
2. 在 src/data/flowers/flowerBasics.ts 文件中新增一种花的配置
3. 在 src/data/flowers/flowerDetails.ts 文件中新增一种花的详情配置

我需要先了解项目的结构，看看相关文件的位置和内容。

> Listed directory src + 3 results

> Listed directory src/data/flowers + 3 results

现在我需要查看这些文件的内容，以了解如何添加新的花。

Read file ts flowerBasics.ts

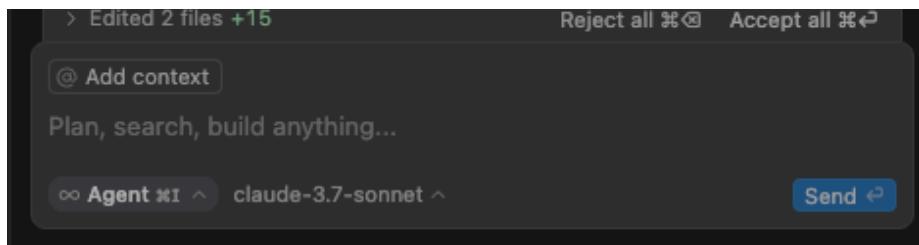
Read file ts flowerDetails.ts

> Listed directory public/images/flowers + 7 results

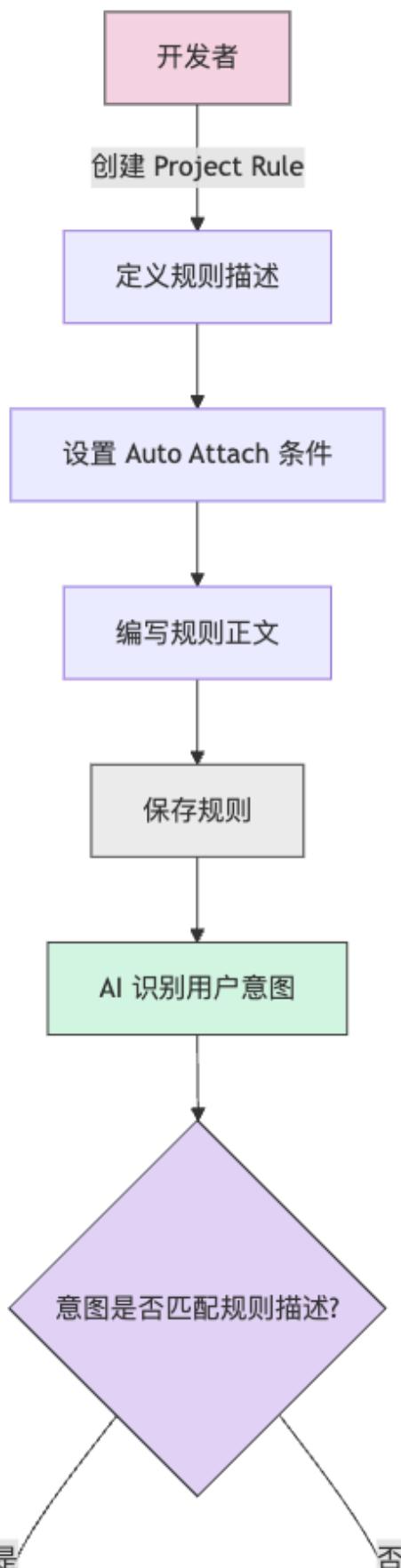
根据规则，我需要先生成一张荷花的近景特写图片。

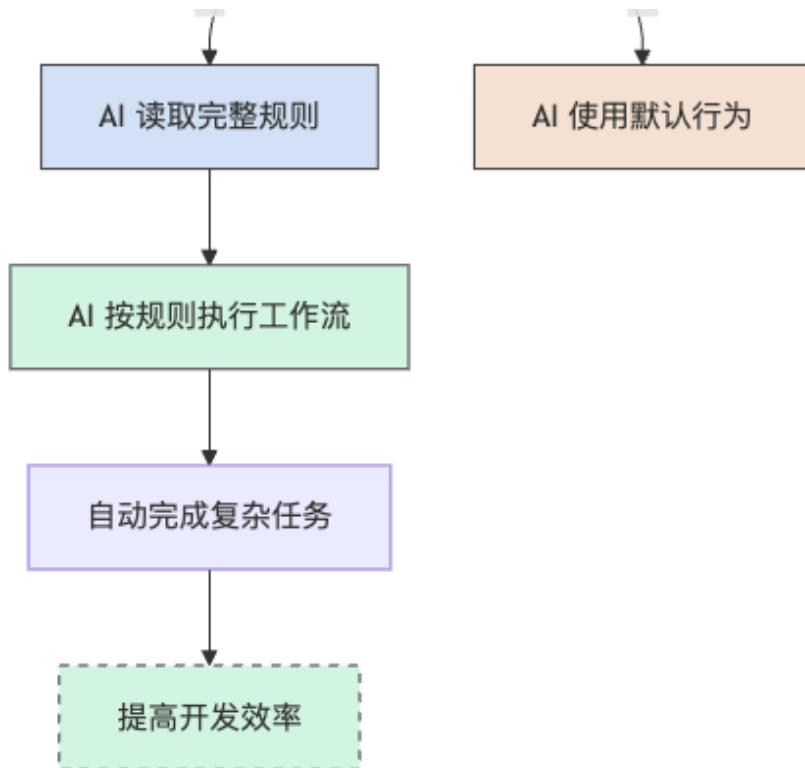
Called MCP tool image_generator ✓

```
"/Users/acring/Workspace/Learning/cursor-learning/cursor-learning-2/public/images/flowers/lotus.jpeg"
],
"metadata": {
  "model": "black-forest-labs/flux-schnell",
  "inference_time_ms": 3469,
  "cache_hit": false
}
```



可以看到, AI 根据我的描述, 识别到了我的意图, 并根据 Project Rule 中对于新增花的意图的后续行为, 自动执行了所有的步骤, 这样, 我们只需要提供一个工作流的模版, 和对应的意图描述, AI 就可以自动执行所有的步骤, 大大提高了效率。





虽然不指定工作流时 Agent 也能自动查找相关文件进行修改，但不同编辑器对项目上下文的处理机制存在差异（例如 WindSurf 具备更优秀的上下文理解能力）。这种依赖编辑器工程化处理能力的方式存在不稳定性，因此建议通过明确的工作流规范来约束 AI 行为。这种主动定义工作流的方式不仅能提升任务执行的准确性，也更符合企业级开发的规范化要求。

另一方面来说，定义工作流不仅仅能提升任务执行的准确性，也能让新加入的成员快速上手，因为工作流是写在项目中的，所以新成员只需要查看工作流，就能快速上手。而减少在开发过程中理解不同导致的修改遗漏。

用 Project Rule 定义代码规范

Project Rule 不仅仅能定义工作流，还能定义代码规范。这也是更加传统的使用方式，这需要用到 Auto Attach 功能。他的描述是

When you specify file patterns here (e.g.*.py or clienttsx),this rule will automatically be included in AI responses for files matching those patterns

翻译过来就是：

当您在此处指定文件模式（例如 *.py 或 clienttsx）时，此规则将自动包含在匹配这些模式的文件的 AI 响应中。

比如说，我们新建一个组件文件的规范 component.mdc:

```

globs: src/components/*.tsx
## 组件规范

```

1. 需要在组件头部添加注释，描述组件功能
2. 需要有 `className` 属性，传入根元素
3. 使用 `memo` 包裹组件，避免不必要的渲染
4. 添加了 `displayName` 属性，便于在 React 开发工具中识别组件

这样，我们在使用 Agent 的时候，假如引用的文件符合 `glob`s 的规则，那么 AI 就会自动按照这个规则来执行。

比如我们要求，优化组件代码，那么 AI 就会自动按照这个规则来优化代码。

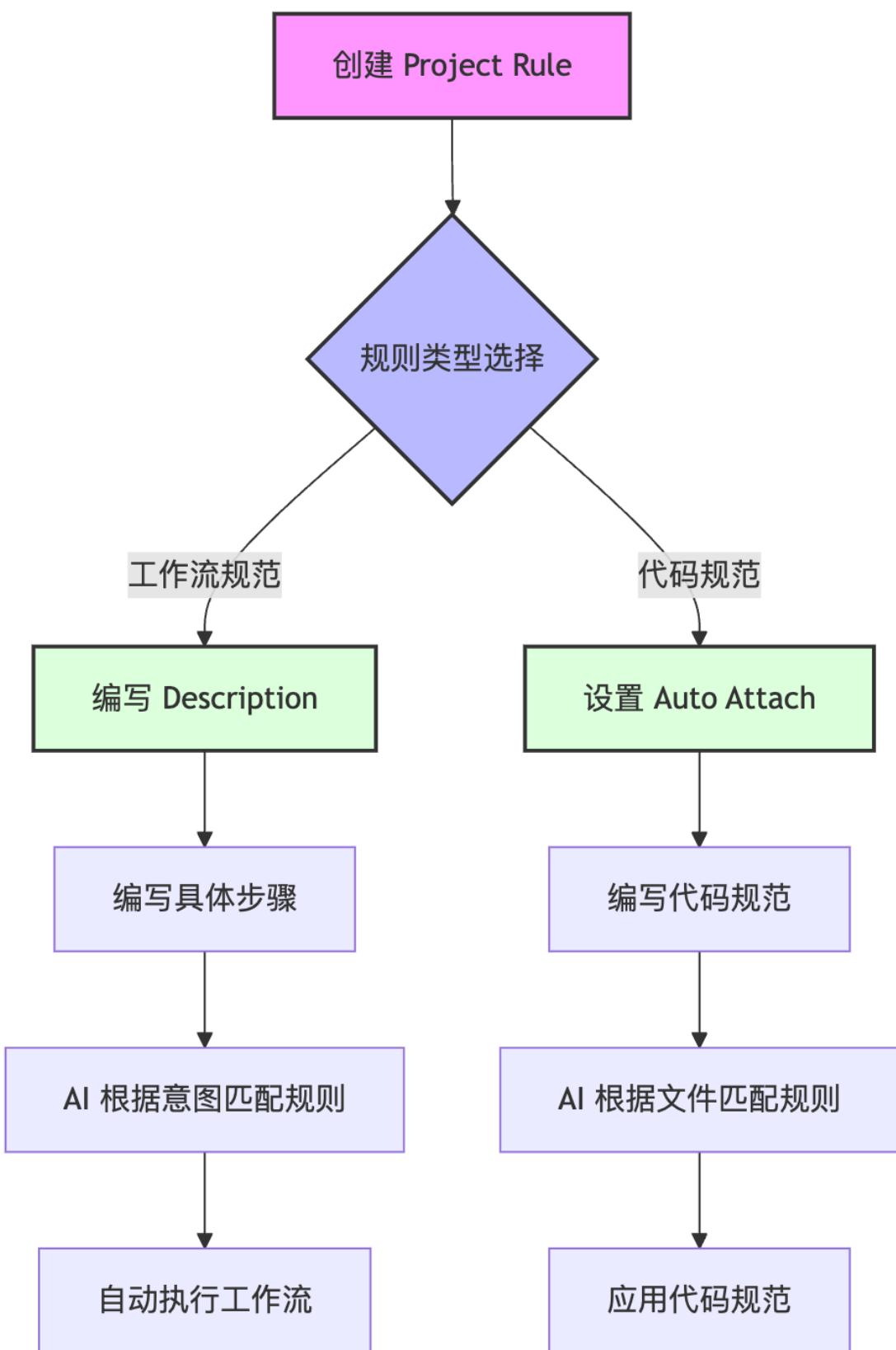
但是需要注意，必须是在引用文件匹配的情况下，AI 才会根据这个规则来执行。

如果只是生成的文件符合 `glob`s 的规则，那么 AI 不会自动按照这个规则来执行

比如，我们要求修改 `src/components/flower-card.tsx` 文件，那么 AI 就会自动按照这个规则来执行。但是如果说我们说，在 `src/components` 新增一个 `xxx` 组件，那么 AI 不会自动根据 `glob`s 来匹配规范。

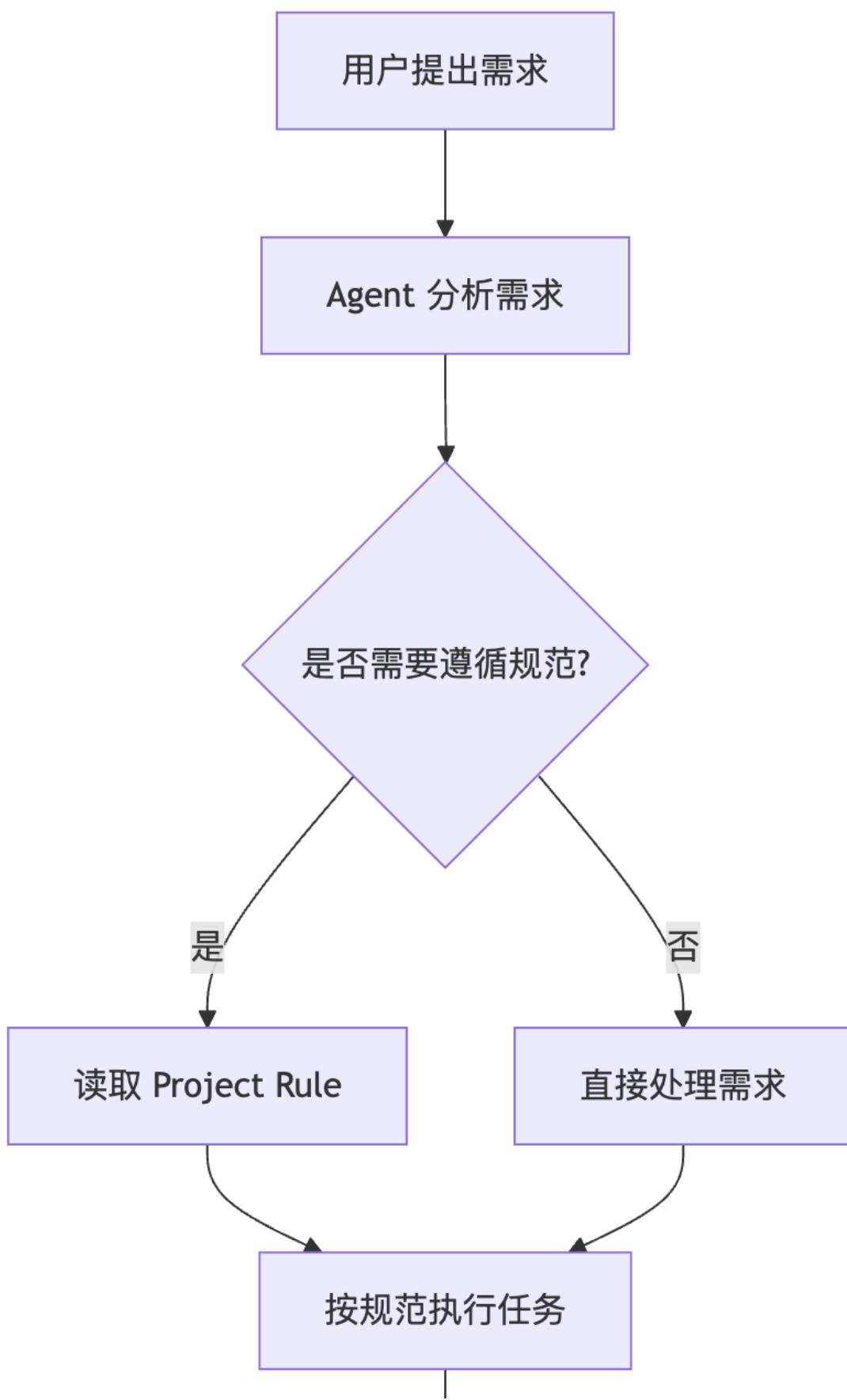
通过测试，在 0.45 中，`cursor` 是能看到 `glob`s 的规则的，但是 0.46 中，`cursor` 是看不到 `glob`s 的规则的，所以导致了这个问题。不知道后续版本会不会优化这个问题。并且，在 0.46 中，即使我们在工作流中要求遵循某一个规范，也大概率会被忽略掉，这也是 `cursor` 的 bug，希望后续版本会修复这个问题

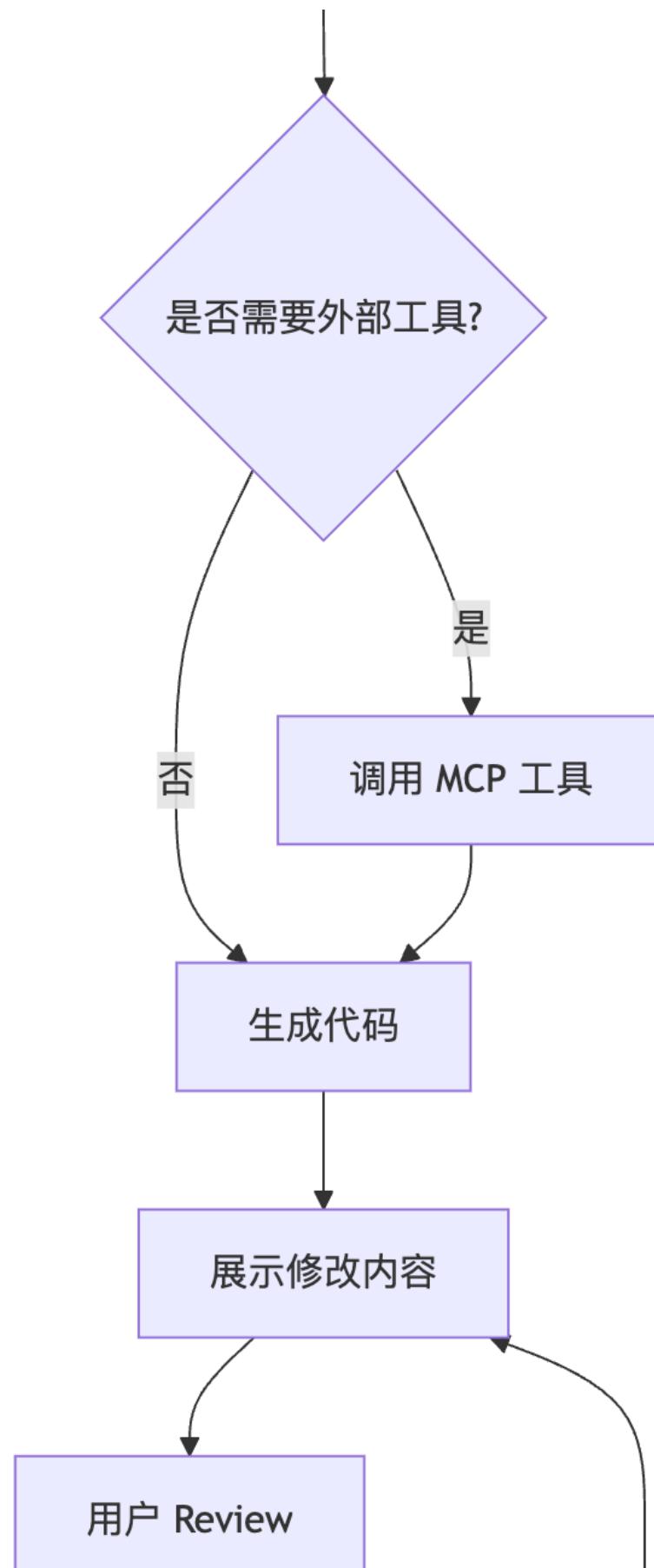
此时还是需要依靠 `Description` 来告诉 AI，什么时候可以执行这个规则。所以我们需要在 `Description` 中告诉 AI，这是一个通用组件规范。

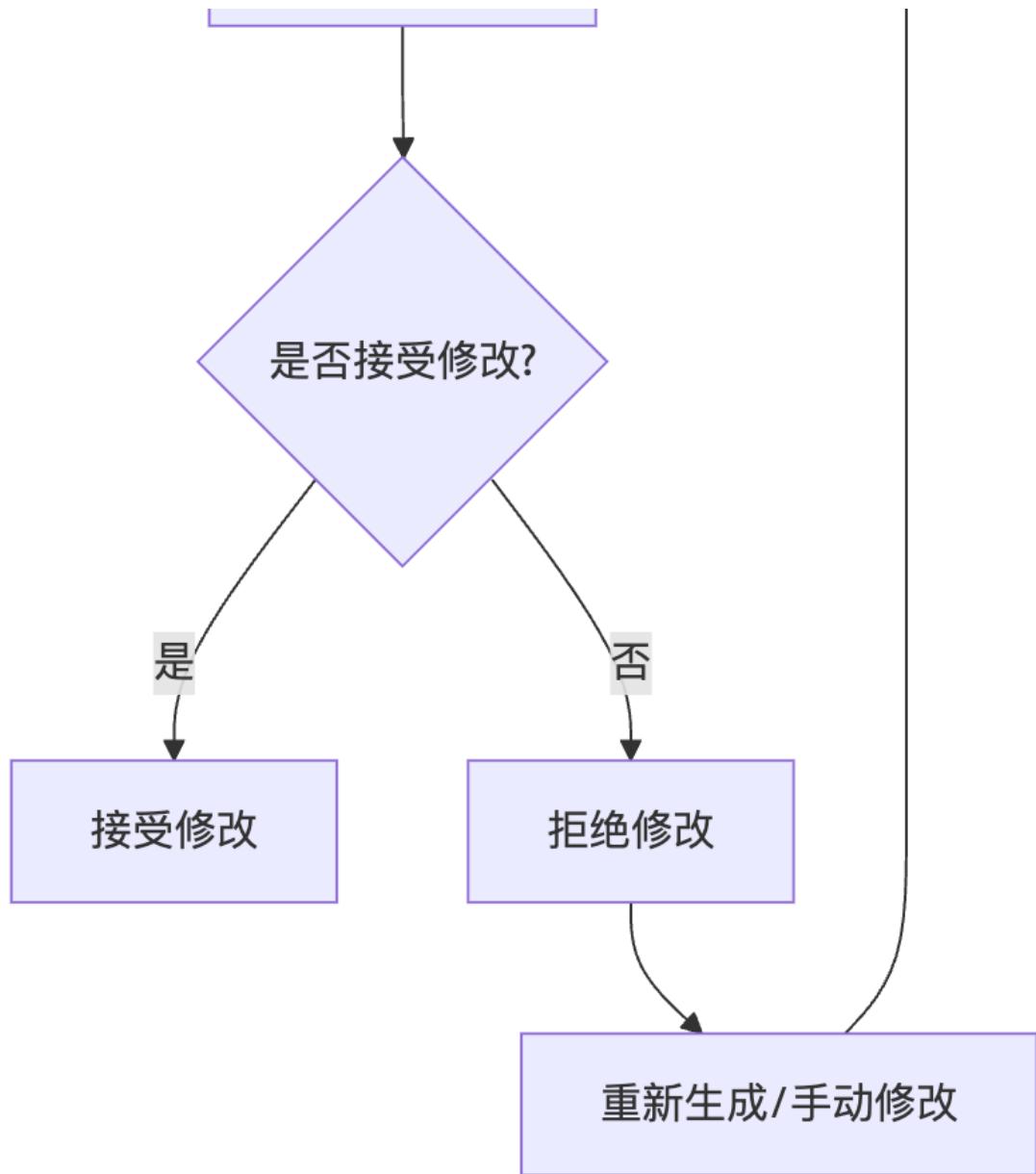


Agent 小结

现在我们通过 Agent 模式，已经可以完成一个复杂的需求了，并且几乎不需要手动编写代码。我们用自然语言描述需求，AI 会自动生成代码，我们只需要进行 Review 和修改即可。在项目逐步增长之后，我们还可以通过 Project Rule 来定义常见工作流和代码规范，让 AI 按照我们定义的规则来快速完成通用任务。遇到复杂的需求，我们还可以通过 MCP 工具来调用外部工具，比如图片生成，视频生成，读取外部文件等。







使用 Agent 与传统开发方式对比

特性	传统开发方式	使用 Agent 开发
代码编写效率	需要手动编写大量代码	通过自然语言描述快速生成代码
工作流程	需要记忆并手动执行各种步骤	自动按照预定义规则执行工作流
代码质量	依赖个人经验和规范执行	自动遵循项目规范，保持一致性
重复性工作	需要频繁手动执行	自动化处理，减少重复劳动
工具集成	需要手动调用和切换工具	通过 MCP 自动调用外部工具
学习曲线	需要学习各种工具和命令	只需掌握自然语言交互方式

我们能做什么

- 找到平时开发中，重复性高的工作流，并定义工作流，让 AI 按照我们定义的规则来快速完成任务

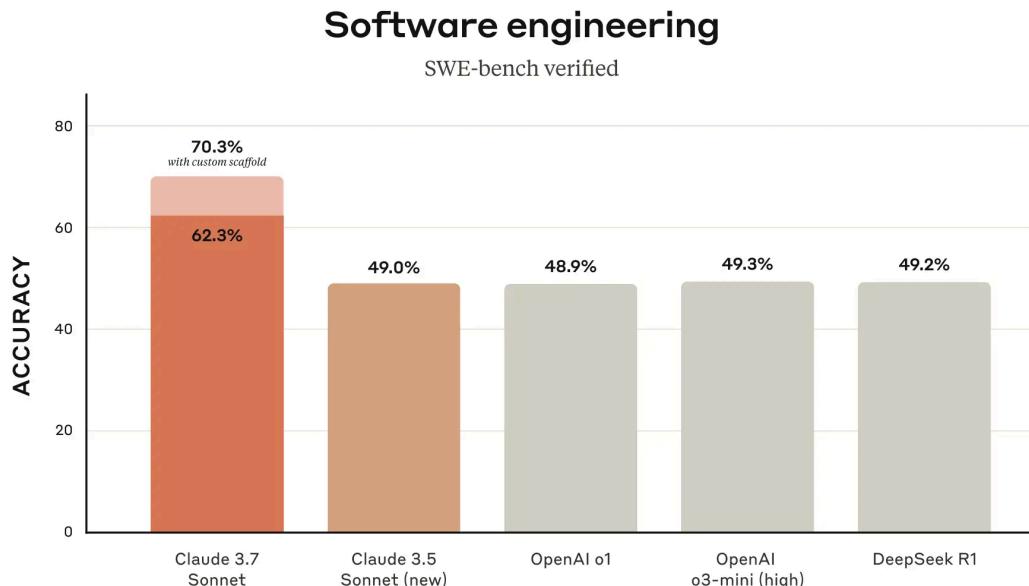
2. 建立团队级别的代码规范，通过 Project Rule 确保 AI 生成的代码符合团队标准
3. 开发和集成自定义 MCP 工具，扩展 AI 的能力边界，比如接入内部文档、设计稿、组件库等
4. 将重复性高的文档工作交给 AI，比如编写 README、API 文档、更新日志等
5. 利用 AI 进行代码重构和优化，提升代码质量
6. 建立 AI 辅助开发的最佳实践，包括如何提问、如何审查代码、如何处理异常情况等
7. 培训团队成员如何有效使用 AI 工具，建立知识共享机制

正确的理念

1. 不要过度依赖 AI，AI 只是辅助工具，不要让 AI 替代你的思考
2. 不要过度信任 AI，AI 的输出需要经过严格的审查和测试
3. AI 不是万能的，AI 也有局限性，比如 AI 的上下文窗口有限，AI 的思维方式和人类不同，AI 可能会出现幻觉等

该使用哪款大模型

根据 [Claude 3.7 Sonnet 的发布](#)，Claude 3.7 Sonnet 在编码和前端网页开发方面表现出特别显著的改进。



- 在代码编辑方便，无脑选择 Claude 3.7 Sonnet
- 在文字编码，特别是中文编码方面，还是可以选择 deepseek r1.

如果没有发现你想要的模型，可以在 `Cursor Settings > Model` 中查看所有支持的模型，并勾选你想要的模型。

后续

后续我还会继续探索 Cursor 的更多用法，特别是如何结合自己的业务场景，来使用 Cursor 来提升开发效率。以及哪些有用的 MCP 工具，来进一步提升 LLM 的能力。