匠人学院
JR ACADEMY

2026 实操版

VS Code 用的，装 Cursor 花 5 分钟

Cursor 中文上手手册

AI 编程 IDE 实战：从安装到 Agent Mode、MCP 集成、.cursor/rules/*.mdc 全解析。7 章配真实命令和配置，照抄能跑。

Agent Mode 自主编程

MCP 接数据库/文档

Rules 按需注入

写代码是 Tab，复杂任务是 Agent

jiangren.com.au · 在线版随官网更新 /wiki/cursor-guide

AI 来匠人 ✦ Cursor 上手手册 ✦ JIANGREN.COM.AU ✦ 写代码是 Tab，复杂任务是 Ag

CONTENTS

目录（7 章）

📖 这本手册怎么读

这本书和官网 wiki《Cursor 上手指南》同源（jiangren.com.au/wiki/cursor-guide，免费、不用注册，官网那份持续更新）。刚听说 Cursor 不知道值不值得换的，第 1 章 5 分钟搞清楚；VS Code 用户想零成本切过来的，第 2 章照着做；想把 Agent Mode 真正用起来的，第 3、4 章是重点——有具体的 prompt 写法和 .cursorrules 模板；想把 Cursor 接数据库、文档、GitHub 的，第 6 章的 MCP 配置可以直接抄；第 7 章的 .mdc Rules 是进阶内容，不急着用可以先跳过。

💬 装 Cursor 卡住了？扫码进群问

群里每周分享 Cursor 使用技巧和 AI 编程真实案例，都是干过活总结出来的。

CHAPTER 01

🤖 Cursor 是什么：AI 编程 IDE 的王者

Cursor 一句话介绍

Cursor 是 Anysphere 团队打造的 AI 编程 IDE，2023 年发布，目前是全球用户量最大的 AI 代码编辑器。它 fork 了 VS Code 内核，在此基础上把 AI 做成了编辑器的一等公民——不是插件，是底层能力。

2025–2026 年 Cursor 连续迭代：**Agent Mode**（自主编程 Agent）、**Background Agent**（云端异步执行任务）、**MCP 集成**（连接外部工具）、**BugBot**（PR 自动修 bug）。2026 年发布的 **Cursor 3.0** 带来了全新的 Agents Window 和 Agent Tabs——你可以同时开多个 Agent 并行干活。这些功能让 Cursor 从"一个带 AI 的编辑器"进化成了"一个会写代码的 Agent 平台"。

核心架构

Cursor 的 AI 能力分四层：

```
┌──────────────────────────────────┐
│      Agent Mode / Composer       │  ← 多文件自主编程
├──────────────────────────────────┤
│     Tab Autocomplete Engine      │  ← 预测性自动补全
├──────────────────────────────────┤
│    Codebase Indexing (向量检索)    │  ← 全项目语义搜索
├──────────────────────────────────┤
│        VS Code 内核               │  ← 编辑器 / 插件 / 终端
└──────────────────────────────────┘
```

- **Codebase Indexing**：打开项目后自动建索引，AI 回答问题时能搜索整个代码库找到相关代码，不只看你手动 `@` 的文件
- **Tab Engine**：不只是补全当前行，能预测你接下来要编辑的位置和内容——改完一行按 Tab，光标会跳到下一个需要改的地方
- **Agent Mode / Composer**：给它一个任务描述，它能自己创建文件、改代码、跑终端命令、读报错、修 bug，循环执行直到搞定

和 Windsurf、Claude Code 的区别

这三个是 2026 年 AI 编程的三大主力工具，各有侧重：

特性	Cursor	Windsurf	Claude Code
本质	VS Code fork + AI	AI 原生 IDE	终端 AI Agent
Agent 核心	Agent Mode	Cascade	Claude CLI
自动补全	Tab（有上限）	Supercomplete（无限）	无
上下文管理	@ 引用 + Notepads	Flow State 自动追踪	自动读代码库
社区生态	最大，教程最多	中等	增长快
独家能力	Background Agent（云端异步）	Web Preview	Extended Thinking

实际使用中的差别：

- **Cursor 的 Agent Mode** 是目前最成熟的 IDE 内 Agent——社区大、`.cursorrules` 生态丰富、和 VS Code 插件兼容性最好。如果你已经在用 VS Code，切 Cursor 的迁移成本接近零
- **Windsurf 的 Flow State** 在上下文自动感知上更强，免费 Tab 补全不限量。预算有限优先看 Windsurf
- **Claude Code** 是终端工具，没有 GUI，但在大型代码库重构和复杂推理上碾压两者。最佳搭配是 Cursor + Claude Code 双开

```
# 选型速查：
# 生态最大 + Background Agent → Cursor
# 免费额度最多 + 上下文追踪 → Windsurf
# 大型重构 + 架构级改动 → Claude Code
# 最佳组合：Cursor 写业务代码，Claude Code 做审查和重构
```

谁适合用 Cursor

- **VS Code 用户**：设置、插件、主题一键导入，体验无缝切换
- **全栈开发者**：Agent Mode 能跨前后端文件协作，一个 prompt 搞定完整功能
- **团队协作**：Business 计划支持集中管理 `.cursorrules`、统一模型配置、使用量审计
- **想要云端 Agent 的人**：Background Agent 能在后台跑任务，你去喝杯咖啡回来代码写好了
- **追求最新模型的人**：Cursor 第一时间接入 Claude、GPT、Gemini 最新模型

不太适合：纯 Vim/Emacs 用户（虽然有 Vim 模式但体验一般）；预算极有限只想免费用的人（Windsurf 免费额度更多）。

CHAPTER 02

🚀 安装与第一个 Agent Mode 项目

下载安装

Cursor 支持 macOS、Windows、Linux，去官网下载安装包：

```
# macOS（推荐 Homebrew）
brew install --cask cursor

# Linux（AppImage，下载后直接运行）
chmod +x cursor-*.AppImage
./cursor-*.AppImage

# Windows
# 去 cursor.com 下载 .exe，双击安装
```

第一次启动会问你要不要导入 VS Code 设置。**一定要导入**——快捷键、主题、插件、snippets 全部保留，零切换成本。

注册与登录

启动后右上角点头像登录。支持 Google、GitHub、邮箱注册。免费用户（Hobby 计划）的额度：

免费额度	说明
Tab 自动补全	2000 次/月
Premium 模型请求	50 次/月（Claude Sonnet、GPT-4o）
Agent Mode	包含在 Premium 请求内
慢速请求	无限（排队等，但不花钱）

50 次 premium 请求足够你体验完整功能。用完了还能无限次慢速请求，只是要排队等几秒。

第一个项目：用 Agent Mode 建一个天气应用

打开一个空文件夹，按 `Ctrl/Cmd + I` 打开 Composer，切到 **Agent** 模式（面板顶部有 Normal / Agent 切换），输入：

```
创建一个 React + TypeScript 天气查询应用：
- 用 Vite 脚手架
- 调用 OpenWeatherMap 免费 API
- 输入城市名显示当前天气和 5 天预报
- 用 Tailwind CSS 做响应式布局
- 加载状态和错误处理都要有
```

Agent Mode 会自动规划并执行：

1. 运行 `npm create vite@latest` 初始化项目
2. 安装 Tailwind CSS、axios 依赖
3. 创建 `WeatherApp.tsx`、`WeatherCard.tsx`、`api/weather.ts`

4. 编写 API 调用逻辑和 UI 组件
5. 运行 `npm run dev` 启动开发服务器

每一步都展示文件 diff 和终端输出。Agent 改错了？对话里直接说"回滚最后一步"就行。

四种交互模式速查

Cursor 有四种 AI 交互方式，搞清楚什么时候用哪个：

模式	快捷键	用途	会改代码吗
Tab	`Tab`	行内自动补全	会
Cmd+K	`Ctrl/Cmd + K`	选中代码后原地修改	会
Chat	`Ctrl/Cmd + L`	问问题、讨论方案	不会
Composer/Agent	`Ctrl/Cmd + I`	多文件任务、自主编程	会

日常使用频率：Tab > Cmd+K > Agent > Chat。大部分代码靠 Tab 补全搞定，复杂任务才需要开 Agent。

索引与 .cursorignore

Cursor 打开项目后会自动索引代码库。大项目可以用 `.cursorignore` 排除不需要索引的目录，和 `.gitignore` 语法一样：

```
node_modules/
dist/
build/
.git/
*.lock
coverage/
.next/
__pycache__/
```

索引完成前 AI 的回答可能缺少上下文。项目设置里可以看到索引状态：`Cursor Settings > Features > Codebase Indexing`。

导入 VS Code 插件的注意事项

Cursor 兼容绝大部分 VS Code 插件，但有几个冲突需要注意：

```
# 必须禁用的插件（和 Cursor AI 冲突）：
# - GitHub Copilot / Copilot Chat
# - Tabnine
# - Codeium（就是 Windsurf 那家）
# - Amazon CodeWhisperer

# 推荐保留的插件：
# - ESLint / Prettier（代码规范）
# - GitLens（Git 增强）
# - Error Lens（行内报错提示）
# - Thunder Client（API 测试）
```

装了 Copilot 不禁用的话，两个 AI 会同时给你补全建议，体验非常混乱。

想找一起学 AI 编程的同学?

加入匠人学院社群，每周有 Cursor 和 Claude Code 的真实案例分享。

CHAPTER 03

🧩 三大核心功能深度解析

Agent Mode：让 AI 自己写代码

Cursor Agent Mode 是目前 IDE 里最成熟的自主编程功能。按 `Ctrl/Cmd + I` 打开 Composer，顶部切到 Agent 模式，它就变成了一个能动手干活的 AI Agent。

Agent Mode 能做的事：

能力	说明
创建/修改文件	跨多个文件同时编辑，自动处理 import
跑终端命令	安装依赖、启动服务、跑测试
读报错并修复	终端报错后自动分析原因、改代码、重跑
搜索代码库	语义搜索找到相关代码，不需要你手动指定文件
调 MCP 工具	连接数据库、读 Figma 设计稿、跑浏览器测试

一个真实例子：给现有项目加用户认证

```
给这个 Express + MongoDB 项目加 JWT 认证：
- 注册、登录、刷新 token 三个接口
- 密码用 bcrypt 加密
- 保护现有的 /api/posts 和 /api/comments 路由
- 写 middleware，不要侵入现有路由代码
```

Agent 会自动执行 6–8 步：装依赖 → 建 `auth.middleware.ts` → 建 `auth.routes.ts` → 建 `auth.service.ts` → 改 `app.ts` 注册路由 → 改现有路由加 middleware → 跑 `npm run build` 验证。全程 2–3 分钟。

提高 Agent 成功率的三个技巧

1. **给足上下文**：用 `@` 引用关键文件。`@src/models/User.ts @src/routes/index.ts` 比让 Agent 自己翻代码库准确率高 30%
2. **分步给任务**：别一次给太大的任务。"先建数据模型和 API 路由"→ 验证没问题 → "再加前端页面"
3. **用 Notepads 存常用上下文**：把项目架构、技术栈规范存进 Notepads，Agent 每次都能读到

Tab 补全：不止补全，还能预测

Cursor 的 Tab 不只是补全当前行，它有一个独特能力叫 **Multi-line Edit Prediction**——改完一行按 Tab，光标会跳到下一个"应该改"的地方，继续帮你补全。

```
# 场景：你要把所有 print() 换成 logging.info()
# 手动改了第一处：
logging.info(f"User {user_id} logged in")  # 原来是 print(...)

# 按 Tab → 光标自动跳到下一个 print()，建议改成 logging.info
# 继续按 Tab → 又跳下一个
# 连按几次 Tab，全文件 print 都换完了
```

这个能力在重命名变量、统一代码风格、批量修改时特别省时间。和 Find & Replace 的区别是：Tab 能理解上下文，不会机械替换——如果某个 `print` 是在 debug 代码里的，它会跳过。

Tab 补全调优

在 `Cursor Settings > Features > Cursor Tab` 里可以精调：

```
{
  "cursor.cpp.enablePartialAccepts": true,
  "cursor.general.enableShadowWorkspace": true
}
```

- **Partial Accepts**：按 `Ctrl/Cmd + →` 只接受补全建议的一个词，而不是整行。适合补全大部分对但结尾需要改的情况
- **Shadow Workspace**：后台隐形工作区，Tab 在这里跑类型检查确保补全建议编译通过，减少错误建议

Composer：多文件协作利器

Composer 是 Cursor 的多文件编辑核心。`Ctrl/Cmd + I` 打开，有两个模式：

模式	用途
Normal	你指定改哪些文件，AI 执行
Agent	AI 自己决定改哪些文件 + 跑什么命令

Normal 模式适合你已经知道要改什么、改哪几个文件的场景。Agent 模式适合"我描述需求，你自己规划怎么做"的场景。

@ 引用：精确控制上下文

Composer 和 Chat 里都能用 `@` 引用各种上下文：

```
@file — 引用具体文件
@folder — 引用整个目录
@code — 引用选中的代码块
@web — 让 AI 搜索互联网
@docs — 引用已索引的文档（如 React 官方文档）
@git — 引用 git diff / commit 历史
@notepads — 引用 Notepads 里存的上下文
@definitions — 引用符号定义（函数、类、变量）
```

实际使用中，`@file` 和 `@folder` 用得最多。给 Agent 精确的上下文，比写一大段 prompt 描述"那个文件"有效得多。

CHAPTER 04

⚙️ 进阶配置与高级玩法

.cursorrules：让 Cursor 按你的规矩写代码

Cursor 最值得花时间配的东西就是 `.cursorrules`。在项目根目录创建这个文件，AI 每次生成代码都会遵守里面的规则——相当于给 AI 一份项目编码规范。

```
# .cursorrules

## 技术栈
- Frontend: Next.js 15 + TypeScript 5.7 + Tailwind CSS 4
- Backend: tRPC + Drizzle ORM
- Database: PostgreSQL 17
- Testing: Vitest + Playwright

## 编码规范
- React 组件一律用函数式 + hooks，禁用 class component
- 状态管理用 Zustand，不用 Redux / Context
- 数据获取用 TanStack Query，不用 useEffect + fetch
- 类型定义放 types/ 目录，用 Zod schema 做运行时验证

## 文件命名
- 组件：PascalCase（UserProfile.tsx）
- 工具函数：camelCase（formatDate.ts）
- 常量：SCREAMING_SNAKE_CASE

## 禁止事项
- 不用 any 类型，实在不知道用 unknown
- 不在 component 里直接调数据库
- 不硬编码 API URL，用环境变量
- 不用 var，只用 const / let
```

新版 Rules 系统

Cursor 已经支持更灵活的 `.cursor/rules/` 目录结构：

```
.cursor/
  rules/
    global.mdc      # 全局规则，所有对话生效
    react.mdc       # 只在 .tsx 文件生效（文件级规则）
    api.mdc         # 只在 src/api/ 下生效（目录级规则）
    testing.mdc     # 手动引用的规则（Agent 按需读取）
```

每个 `.mdc` 文件的 frontmatter 指定作用范围：

类型	说明
`always`	每次对话都自动加载
`auto-attached`	匹配 glob 模式时自动加载，如 `globs: ["**/*.tsx"]`
`agent-requested`	Agent 根据任务需要自己决定是否读取
`manual`	只有你用 `@rules` 手动引用时才加载

建议每个项目 5-8 个规则文件，每个不超过 100 行。社区资源 awesome-cursorrules 有 React、Vue、Next.js、Python、Go 等上百个模板可以直接抄。

MCP：给 Agent 连接外部工具

MCP（Model Context Protocol）让 Cursor 的 Agent 调用外部工具——数据库、设计稿、浏览器、API 平台。在 `Cursor Settings > Features > MCP` 里添加，或者手动编辑 `.cursor/mcp.json`：

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "POSTGRES_CONNECTION_STRING": "postgresql://user:pass@localhost:5432/mydb"
      }
    },
    "playwright": {
      "command": "npx",
      "args": ["-y", "@anthropic-ai/mcp-server-playwright"]
    },
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "ghp_xxx"
      }
    }
  }
}
```

配好后，Agent Mode 会自动识别可用的 MCP 工具。对话里说"查一下数据库里有哪些表"或"用 Playwright 截个图看看首页效果"，Agent 就会调用对应的 MCP Server。

实用 MCP 推荐：

MCP Server	用途
Playwright	浏览器自动化、截图、端到端测试
PostgreSQL/MySQL	查 schema、跑 SQL、验证数据
GitHub	读 issue、创建 PR、查 CI 状态
Figma	读设计稿生成 UI 代码
Filesystem	安全地读写指定目录的文件

Background Agent：云端异步编程

Background Agent 是 Cursor 的杀手级功能——把任务丢给云端 Agent，它在远程环境里自主执行，你关掉电脑都能继续跑。

使用方法：Composer 里点 Background Agent 图标（或搜索 `Background Agent`），描述任务后提交。Agent 会在云端 fork 一个分支，完成后你能看到所有改动的 diff，确认后合并。

```
# 适合 Background Agent 的任务：
# - 给整个项目写单元测试
# - 迁移数据库 schema + 更新所有查询
# - 把项目从 JavaScript 迁到 TypeScript
# - 批量重命名 API 接口 + 更新前端调用

# 不适合的任务：
# - 需要实时反馈的 UI 调试
# - 依赖本地环境变量或数据库的任务
# - 需要你频繁确认方向的探索性任务
```

Notepads：可复用的上下文片段

Notepads 在左侧边栏，用来存项目级的上下文信息。和 `.cursorrules` 的区别：rules 是强制规则，Notepads 是参考资料。

创建几个常用 Notepad：

- **架构说明**：项目整体结构、目录规范、数据流
- **API 契约**：前后端接口定义、字段说明
- **设计系统**：颜色、字号、间距、组件规范

在 Composer 或 Chat 里用 `@notepads` 引用。Agent 会自动读取相关的 Notepad 内容，不用每次重新解释项目背景。

YOLO Mode：全自动执行

Cursor Settings 里搜索 `auto-run`，开启后 Agent 可以自动执行终端命令不需要你逐条确认。适合你信任 AI 输出、想让它一口气跑完的场景。

可以配置白名单，只允许自动执行特定命令：

```
npm install, npm run build, npm test, npx prisma generate
```

危险命令（`rm -rf`、`DROP TABLE`）即使开了 YOLO 也会弹确认。但还是建议只在个人项目里开——团队项目用默认的逐步确认更安全。

配好 .cursorrules 和 MCP 之后，想聊聊怎么把 AI 编程能力转化成职业竞争力？

CHAPTER 05

❓ 常见问题、定价与选型建议

常见问题

Agent Mode 改错了代码怎么办?

Cursor 有 checkpoint 机制。Agent 每一步操作前都会保存快照，在 Composer 面板里找到对应步骤，点 **Restore Checkpoint** 即可回滚。比手动 `git stash` 或 `Ctrl+Z` 靠谱。

如果整个对话都跑偏了，直接关掉 Composer 开一轮新的。之前的改动可以用 Git 回滚：

```
# 查看 Agent 改了哪些文件
git diff --name-only

# 只回滚某个文件
git checkout -- src/components/Header.tsx

# 全部回滚
git checkout -- .
```

Tab 补全不准怎么调?

几个排查方向：

1. **索引没跑完**——设置里看 Codebase Indexing 状态，大项目首次索引可能要几分钟
2. **`.cursorignore` 没配**——`node_modules` 和 `dist` 进了索引会拉低准确率
3. **冲突插件没禁**——Copilot、Tabnine、Codeium 必须禁用
4. **模型选错了**——Tab 用 `cursor-small` 模型时速度快但准确率稍低，试试切回默认

```
// settings.json - 推荐 Tab 配置
{
  "cursor.cpp.enablePartialAccepts": true,
  "cursor.general.enableShadowWorkspace": true
}
```

Agent 跑到一半网络断了?

Agent Mode 的对话不会丢，重新联网后在 Composer 历史里找到那轮对话继续就行。Background Agent 更不怕断网——它跑在云端，你断网它也在继续。

中文支持如何?

Cursor 完全支持中文对话和中文注释。Agent Mode 用中文描述需求、中文写 `.cursorrules` 都没问题。唯一建议：代码本身还是写英文（变量名、函数名），中文变量名在某些框架和工具链里会出问题。

定价详解（2026 年）

Cursor 在 2026 年改用了 **token 消耗制**——不再是固定次数，而是按模型和任务复杂度扣 credit。年付打八折。

方案	月费	额度	Tab 补全	独有功能
Hobby	$0	有限 Agent 请求	有限	基础功能
Pro	$20	$20 前沿模型 credit	无限	Agent Mode + MCP + BugBot
Pro+	$60	3x Pro 额度	无限	高频用户首选
Ultra	$200	20x Pro 额度	无限	优先体验新功能
Teams	$40/人	同 Pro	无限	SSO + RBAC + 用量分析

我的建议：先用 Hobby 体验一周。如果每天都在用 Agent Mode，直接上 Pro——$20/月省下来的时间远超这个价。大部分开发者 Pro 就够，Pro+ 适合每天高强度使用 Agent 的人。

Cursor vs Windsurf 价格对比

```
Cursor Pro $20/月:
  ✓ 500 premium 请求
  ✓ 无限 Tab 补全
  ✓ Background Agent
  ✓ 社区最大、教程最多
  ✗ Tab 补全有上限（但 Pro 无限）

Windsurf Pro $20/月:
  ✓ 500 Cascade credit
  ✓ 无限 Tab 补全（免费版也不限量）
  ✓ Flow State 自动上下文
  ✓ Web Preview 内置
  ✗ 没有 Background Agent
  ✗ 社区和教程较少
```

两者同价，选 Cursor 看生态和 Background Agent，选 Windsurf 看免费额度和上下文追踪。

选型决策树

```
你需要 AI 编程工具吗?
├── 是 → 你用 VS Code 吗?
│   ├── 是 → Cursor（零迁移成本）
│   └── 否 → 你习惯终端吗?
│       ├── 是 → Claude Code
│       └── 否 → Windsurf
├── 你预算有限吗?
│   ├── 是 → Windsurf（免费额度最多）
│   └── 否 → Cursor Pro + Claude Code（最强组合）
└── 你需要云端异步 Agent 吗?
    ├── 是 → Cursor Background Agent
    └── 否 → 三个都行，试了再说
```

最佳实践总结

1. **先配 `.cursorrules`**——投入 10 分钟，之后每次生成的代码都符合你的规范
2. **用 Notepads 存项目上下文**——不用每次对话都重新解释架构
3. **Agent Mode 分步走**——大任务拆成 3–5 个小任务，成功率翻倍
4. **Tab 补全是主力**——90% 的代码靠 Tab 搞定，Agent 只在大任务时开
5. **配 MCP**——连上数据库和 Playwright 后，Agent 的能力上一个台阶
6. **定期清理 Composer 历史**——对话太长上下文会混乱，每个任务新开一轮

CHAPTER 06

🔌 Cursor MCP 集成实战

MCP（Model Context Protocol）是 Anthropic 提出的一个开放协议，定义了 AI 如何与外部工具和数据源通信。Cursor 从 0.43 版本起内置 MCP 支持，现在你可以把 GitHub、Notion、Postgres 这些工具直接接进 Agent Mode，让 AI 不只是"建议"——而是真的帮你执行操作。

截至 2026 年初，社区已有超过 5000 个 MCP 服务端，覆盖数据库、文档、监控、支付等几乎所有常见工具。

MCP 是怎么工作的

每个 MCP 服务端对外暴露一批"工具"（tool），Agent 在对话时可以调用这些工具。比如：

- Postgres MCP 暴露 `query_database` 工具
- GitHub MCP 暴露 `create_issue`、`list_pull_requests`、`get_file_contents` 等工具
- Notion MCP 暴露 `search_pages`、`create_page`、`update_block` 等工具

Agent 接到你的指令后，会自动判断要调哪些工具，构造请求，拿到结果，继续推理——全程不需要你手动切 tab。

注意：MCP 工具只在 **Agent Mode** 下可用，普通 Ask / Chat 模式里不触发。

配置文件位置

Cursor 的 MCP 配置写在 JSON 文件里，有两个层级：

配置文件	作用范围
`.cursor/mcp.json`（项目根目录）	只对当前项目生效，可以提交进 git（注意不要 hardcode 密钥）
`~/.cursor/mcp.json`（用户主目录）	全局生效，所有项目共用

基本格式：

```
{
  "mcpServers": {
    "服务名": {
      "command": "运行命令",
      "args": ["参数列表"],
      "env": {
        "环境变量名": "值"
      }
    }
  }
}
```

密钥不要 hardcode，用 `${env:变量名}` 从系统环境变量读取：

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${env:GITHUB_TOKEN}"
      }
    }
  }
}
```

配置完后，打开 Cursor Settings → Tools & MCP，能看到每个服务端的状态指示灯。绿色表示正常，红色说明启动失败——通常是命令不存在或 token 无效。

GitHub MCP：让 Agent 直接操作仓库

安装配置

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${env:GITHUB_TOKEN}"
      }
    }
  }
}
```

`GITHUB_TOKEN` 用 Personal Access Token（PAT）。权限按需给：

- 只读代码审查：勾选 `repo:read`
- 需要创建 Issue / PR：勾选 `repo`（读写）
- 不需要 Actions：不要勾 `workflow`

在终端里先 export：

```
export GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

或者加进 `~/.zshrc` / `~/.bashrc` 让每次启动自动生效。

实际场景

场景 1：调查一个陌生仓库

打开 Cursor，新建 Composer，输入：

```
用 github MCP 读一下 facebook/react 仓库最近 10 个 open PR，
找出哪些涉及 concurrent rendering，给我一个简短摘要
```

Agent 会调 `list_pull_requests` + `get_pull_request`，直接返回结果，不用你开浏览器。

场景 2：修完 bug 顺手开 Issue

你刚修了一个线上 bug，想把问题记录成 Issue 以便追踪：

```
在 my-org/my-repo 创建一个 Issue，标题是"Fix: 登录页 token 刷新竞态条件"，
描述用中文，把刚才我们对话里找到的根因和修复方案写进去
```

Agent 调 `create_issue`，自动填好 body——你不用离开 IDE。

场景 3：Code Review 辅助

```
读一下 my-org/my-repo 的 PR #142，找出潜在的安全问题
```

Agent 拉 PR diff，分析代码，直接给你 review 意见。

Notion MCP：把文档接进工作流

Notion 官方在 2025 年发布了 `@notionhq/notion-mcp-server`，配置方式：

```
{
  "mcpServers": {
    "notion": {
      "command": "npx",
      "args": ["-y", "@notionhq/notion-mcp-server"],
      "env": {
        "OPENAPI_MCP_HEADERS": "{\"Authorization\": \"Bearer ${env:NOTION_TOKEN}\", \"Notion-
Version\": \"2022-06-28\"}"
      }
    }
  }
}
```

`NOTION_TOKEN` 从 Notion 开发者页面创建 Integration 拿到（Internal Integration Token）。创建后在 Notion 页面右上角「连接」里授权给这个 Integration，它才能读写那个页面。

如果想用社区版（更轻量）：

```
{
  "mcpServers": {
    "notion": {
      "command": "npx",
      "args": ["-y", "@suekou/mcp-notion-server"],
      "env": {
        "NOTION_API_TOKEN": "${env:NOTION_TOKEN}"
      }
    }
  }
}
```

实际场景

场景 1：读文档写代码

团队把 API 设计文档放在 Notion：

```
读 Notion 页面「用户服务 API v2 设计」，
根据里面的接口定义帮我生成 TypeScript 类型声明
```

Agent 调 `retrieve_page`，拿到 Notion 内容，直接生成类型文件。

场景 2：Bug 修完更新文档

```
刚修了用户登录流程的一个 bug，找到 Notion 里「登录服务已知问题」这个页面，
把今天修的问题追加进去（日期 2026-05-10，问题描述：token 刷新竞态，修复方式：加锁）
```

Agent 调 `search`（找页面）→ `append_block_children`（追加内容），文档更新到位。

场景 3：把 Sprint 任务转成代码骨架

```
读 Notion 数据库「2026 Q2 Sprint」里本周分配给我的任务，
帮我在 src/features/ 下生成对应的文件骨架
```

Postgres MCP：自然语言查数据库

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "DATABASE_URL": "${env:DATABASE_URL}"
      }
    }
  }
}
```

`DATABASE_URL` 格式：`postgresql://user:password@host:5432/dbname`

生产数据库建一个**只读用户**专门给 MCP 用，不要用 owner 账号：

```
-- 创建只读角色
CREATE USER cursor_mcp WITH PASSWORD 'xxxxx';
GRANT CONNECT ON DATABASE myapp TO cursor_mcp;
GRANT USAGE ON SCHEMA public TO cursor_mcp;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO cursor_mcp;
-- 对未来新建的表也生效
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO cursor_mcp;
```

实际场景

场景 1：调试时查线上数据

你在排查一个用户反馈的订单问题：

```
在 postgres 里查一下 user_id = 12345 最近 7 天的订单记录，
看看 status 和 payment_status 有没有不一致的情况
```

Agent 调 `query`，直接出结果，不用开 Postico / DBeaver / psql。

场景 2：理解陌生数据库

接手老项目，不熟悉 schema：

```
列出 postgres 里所有表，重点说明 users、orders、payments 这三张表的字段和关联关系
```

Agent 调 `list_tables` + `describe_table`，3 秒给你一份关系图谱。

场景 3：写迁移脚本

```
users 表目前没有 email_verified 字段，
帮我写一个 migration SQL，加这个字段并把现有所有 is_active=true 的用户设为已验证
```

Agent 查了表结构再生成 SQL，不会出现字段名写错的低级错误。

多个 MCP 服务端同时配置

实际工作中往往会同时接多个服务端：

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${env:GITHUB_TOKEN}"
      }
    },
    "notion": {
      "command": "npx",
      "args": ["-y", "@suekou/mcp-notion-server"],
      "env": {
        "NOTION_API_TOKEN": "${env:NOTION_TOKEN}"
      }
    },
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "DATABASE_URL": "${env:DATABASE_URL}"
      }
    }
  }
}
```

这份配置放在 `~/.cursor/mcp.json`（全局），所有项目共用，只在特定项目需要覆盖时才写 `.cursor/mcp.json`。

安全注意事项

最小权限原则：每个 MCP 服务端的凭证只给它真正需要的权限。GitHub token 只需要 read 就不要给 write。

工具审批模式：在 Cursor Settings → Agent 里可以开启"Ask before tool use"——每次 Agent 准备调用 MCP 工具时会先弹框让你确认，防止误操作。生产环境推荐打开。

不要提交密钥：`.cursor/mcp.json` 里用 `${env:VAR_NAME}` 引用环境变量，密钥通过 `.env.local` 或系统环境变量注入，`.env.local` 加进 `.gitignore`。

常见问题排查

MCP 服务端显示红色 / 离线

99% 是 `npx` 找不到包或者 node 版本不对。先手动跑一下：

```
npx -y @modelcontextprotocol/server-github
```

看报什么错。常见原因：npm registry 超时（换源），node < 18（升级）。

Agent 说"我没有那个工具"

确认你在 Agent Mode（不是 Ask 模式）。然后看 Cursor Settings → Tools & MCP 里那个服务端是不是绿色在线。

Postgres 连接超时

检查 `DATABASE_URL` 里的 host 是否可从本机直接连（生产 RDS 一般要配 VPN 或 bastion）。本地开发用 `localhost` 就行。

下一步扩展

接完这三个之后，工程师们常见的进阶配置：

- **Playwright MCP**：让 Agent 直接操控浏览器，做 E2E 测试、爬数据、自动填表单
- **Slack MCP**：在 Cursor 里直接查频道消息、发通知
- **Stripe MCP**：Agent 查订阅状态、退款记录，不用开 Dashboard
- **File System MCP**：Agent 在本机指定目录里读写文件，适合批量处理脚本

MCP 目录站 cursor.directory 能搜到大部分社区维护的服务端，按需安装。

MCP 接了哪些工具最有用?

群里有人整理了 10 个最常用的 MCP Server 配置，进群发口令"MCP"领取。

CHAPTER 07

📋 .cursor/rules/*.mdc 深度解析

`.cursorrules` 是 Cursor 早期的规则方案：一个文件，写啥都往里扔，每次对话都全量注入上下文。规则多了之后这个方案开始露出问题——token 消耗直接拉满，不相关的规则也在占位，Agent 的注意力被稀释。

从 0.45 版本起，Cursor 正式推出 `.cursor/rules/` 目录 + `.mdc` 文件的新方案，核心改变是**按需注入**：一条规则只有在真正需要的时候才出现在上下文里。

.mdc 文件的基本格式

`.mdc` 全称 Markdown with Cursor metadata，本质是在普通 Markdown 文件头部加一段 YAML frontmatter，告诉 Cursor 这条规则什么时候应该注入：

```
---
description: 这条规则的用途，给 Agent 看的说明
globs:
  - src/**/*.tsx
  - src/**/*.ts
alwaysApply: false
---

（以下是规则正文，普通 Markdown）

## TypeScript 代码规范

- 所有组件用 function 声明，不用箭头函数
- Props 类型用 interface，不用 type alias
- 禁止 any，用 unknown + 类型收窄
```

frontmatter 里有三个字段控制触发行为：

字段	类型	作用
`description`	string	规则的用途描述，Agent Requested 模式下 AI 靠这个决定要不要加载
`globs`	string[]	文件 glob 模式，Auto Attached 模式下匹配文件自动触发
`alwaysApply`	boolean	为 true 时规则永远注入，不管对话内容是什么

四种触发类型

1. Always Apply — 全局生效

```
---
alwaysApply: true
---
```

只要 `alwaysApply: true`，这条规则在**每一次**对话里都会被注入，不管你在讨论 TypeScript 还是 SQL 还是在问天气。适合放全局性约定：

- 项目技术栈（"本项目用 Next.js 14 App Router，不用 Pages Router"）
- 代码风格底线（"禁止 console.log 提交到 main"）
- 安全红线（"所有用户输入必须经过 zod 校验"）

注意：`alwaysApply: true` 时，`globs` 字段会被完全忽略——规则无条件注入，不需要文件匹配。

每条 Always Apply 规则都消耗固定 token。保守上限：总 Always Apply 规则不超过 **2000 token**（约 1500 中文字）。超了就开始影响 Agent 的有效推理空间。

2. Auto Attached — 文件触发

```
---
globs:
  - "**/*.test.ts"
  - "**/*.spec.ts"
alwaysApply: false
---
```

当对话里引用的文件（`@file`、Composer 里打开的文件、Agent 读到的文件）匹配 `globs` 里的 pattern 时，规则自动注入。

这是最常用的模式，因为它做到了精准匹配：写测试时只注入测试规范，改 React 组件时只注入组件规范，互不干扰。

典型配置：

```
---
globs:
  - "src/components/**/*.tsx"
  - "src/app/**/*.tsx"
alwaysApply: false
---

## React 组件规范

- 组件文件名用 PascalCase（UserCard.tsx，不是 user-card.tsx）
- 每个组件文件只 export 一个组件
- 客户端组件顶部必须有 "use client"
- 样式用 Tailwind，不要 inline style 除非动态值
```

`globs` 字段支持数组，多个 pattern 之间是**或**的关系，只要有一个匹配就触发。

3. Agent Requested — AI 自主决定

```
---
description: "数据库 schema 设计规范，当需要设计或修改数据库表结构时使用"
alwaysApply: false
---
```

没有 `globs`，没有 `alwaysApply: true`，只有 `description`——这种情况下，Cursor Agent 会把 description 读一遍，**自行判断**当前对话是否需要这条规则。

这个模式适合：

- 专项规范（数据库 schema 规则、API 设计规范）
- 不和特定文件类型绑定，但只在特定场景下需要的规则

Agent Requested 的准确性高度依赖 description 的质量。写得模糊（"代码规范"）AI 可能漏拉；写得具体（"当需要写 Prisma schema 或 SQL migration 时加载此规则"）成功率显著提高。

4. Manual — 手动召唤

```
---
alwaysApply: false
---
```

三个字段全空（或只有规则正文没有 frontmatter），这条规则只能通过 `@rule-name` 手动引用：

```
@performance-checklist 帮我看看这段代码有没有性能问题
```

适合那些"偶尔需要但不想常驻"的规则：

- 发版前的代码审查清单
- 特定架构迁移指南
- 临时的代码风格统一要求

Glob 模式详解

glob 是一种路径匹配语法，Cursor 遵循标准 glob 规范：

模式	含义	例子
`*`	匹配单层目录内的任意字符（不含 `/`）	`src/*.ts` 匹配 `src/index.ts`，不匹配 `src/utils/helper.ts`
`**`	匹配任意深度的目录	`src/**/*.ts` 匹配 `src/utils/helper.ts`
`{a,b}`	匹配多个可选项	`src/**/*.{ts,tsx}`
`!pattern`	排除	`!**/*.test.ts`

实际工程中几个常见写法：

```
globs:
  # 所有 TypeScript 文件（含 TSX）
  - "src/**/*.{ts,tsx}"

  # 只匹配测试文件
  - "**/*.{test,spec}.{ts,js}"

  # API 路由（Next.js App Router）
  - "src/app/api/**/route.ts"

  # 配置文件
  - "*.config.{ts,js,mjs}"
  - ".env*"
```

一个容易踩的坑：`src/*.tsx` 和 `src/**/*.tsx` 的区别。前者**只匹配** `src/` 直属的文件，后者匹配所有子目录。大多数场景都应该用 `**`。

多 rule 并发：优先级与冲突处理

当一次对话里同时触发了多条规则，它们的内容会合并注入。问题来了：两条规则说的话互相矛盾怎么办？

优先级层级

Cursor 有三层规则来源，按优先级从高到低：

```
Team Rules（团队规则）
    ↓ 若有冲突，上层覆盖下层
Project Rules（.cursor/rules/*.mdc）
    ↓
User Rules（全局用户设置）
    ↓
Legacy（.cursorrules）
```

同层级的多个 `.mdc` 文件之间，Cursor **不保证**哪一条优先。如果两条 Project Rules 互相矛盾（一条说"用 default export"，另一条说"禁止 default export"），Agent 会产生不一致的输出——有时遵循这条，有时遵循那条。

解决冲突的实际做法

拆分关注点，避免 glob 重叠

把不同关注点的规则分开，让它们不会在同一个文件上同时触发：

```
.cursor/rules/
├── react-components.mdc    # globs: src/components/**/*.tsx
├── api-routes.mdc          # globs: src/app/api/**/*.ts
├── database.mdc            # globs: prisma/**/*.prisma, **/migrations/**
└── global-security.mdc    # alwaysApply: true（安全底线，全局）
```

这样打开 `src/components/UserCard.tsx` 只触发 `react-components.mdc`，不会和 `api-routes.mdc` 撞。

优先级明确化：关键约束写进 Always Apply

如果有一条规则是你不希望被任何其他规则覆盖的底线，把它单独抽出来做成 Always Apply 并置顶：

```
---
alwaysApply: true
---

## 安全底线（所有代码必须遵守，不受其他规则覆盖）

- 禁止在代码里 hardcode 任何密钥、token、密码
- 所有 SQL 必须用参数化查询，禁止字符串拼接
- 用户输入必须在边界处 validate，不信任任何来源
```

定期审查重复规则

随着项目演进，.mdc 文件容易越积越多，老规则和新规则形成冲突。建议每季度 grep 一下：

```
# 找所有 .mdc 文件里包含 "export" 相关规则
grep -r "export" .cursor/rules/ --include="*.mdc" -l

# 列出所有 alwaysApply: true 的规则（最高成本）
grep -rl "alwaysApply: true" .cursor/rules/
```

目录组织最佳实践

Cursor 会递归读取 `.cursor/rules/` 下所有 `.mdc` 文件，子目录纯粹是给人看的，不影响规则加载：

```
.cursor/rules/
├── core/
│   ├── security.mdc        # alwaysApply: true — 安全红线
│   └── git-commits.mdc     # alwaysApply: true — commit message 格式
├── frontend/
│   ├── react.mdc           # globs: **/*.tsx
│   ├── styling.mdc         # globs: **/*.{css,scss,tailwind}
│   └── testing.mdc         # globs: **/*.{test,spec}.{ts,tsx}
├── backend/
│   ├── api-design.mdc      # description: API 设计时
│   └── database.mdc        # globs: prisma/**
└── ops/
    └── deployment.mdc      # manual — 手动 @deployment 召唤
```

几个经验规则：

- 单个 `.mdc` 文件保持在 **500 行以内**。太长的规则文件 AI 读到后面会开始走神
- 所有 Always Apply 规则加起来 **不超过 2000 token**，约 1500 中文字或 500 行英文
- 每条规则只做一件事。"前端规范.mdc" 是个反面例子，拆成 react.mdc + styling.mdc + testing.mdc 会准确很多
- `description` 字段要具体——"当需要写 xxx 时加载" 比 "xxx 规范" 的触发成功率高

从 .cursorrules 迁移

如果你的项目有一个胖 `.cursorrules` 文件，迁移到 `.mdc` 的方式：

1. 按关注点把内容切段（代码风格 / 测试规范 / 安全要求 / API 设计……）
2. 每段创建一个 `.mdc` 文件，判断它适合哪种触发类型
3. 全局底线 → `alwaysApply: true`；和文件类型挂钩的 → `globs`；专项规范 → `description`
4. 迁移完后保留 `.cursorrules` 一段时间做并行对照，确认没有规则丢失

如果 `.cursorrules` 和 `.mdc` 规则同时存在，当两者冲突时 `.mdc` 优先，`.cursorrules` 内容会被静默覆盖。

快速参考

```
# 全局生效
---
alwaysApply: true
---

# 文件匹配触发
---
globs: ["src/**/*.tsx", "src/**/*.ts"]
alwaysApply: false
---

# AI 按需决定
---
description: "当需要设计数据库 schema 或写 migration 时使用"
alwaysApply: false
---

# 手动 @rule-name 召唤
---
alwaysApply: false
---
```

四种模式对应四种场景，选对了比堆规则数量更有效。

匠人学院
JR ACADEMY

学 AI 来匠人

Cursor 的 Agent Mode 配好了之后，真正的问题是：你的项目架构让 AI 看得懂吗？想把 AI 编程能力变成真正的竞争力，看看匠人学院的 IT 求职课程。

在线版随官网持续更新：

jiangren.com.au/wiki/cursor-guide

扫码关注匠人学院

jiangren.com.au