

---

# Cursor Cookbook

王福强



2025-04-13

## Contents

0.1	Cursor 是怎么崛起的？	3
0.2	Cursor 与 Agent 之间是什么关系？	4
0.3	Cursor 三种模式有什么区别与妙用？	5
0.4	Cursor Rules 是用来干嘛的？	7
0.5	Cursor 里如何引用各种上下文内容？	9
0.6	如何使用 Cursor 完成复杂任务和项目？	10
0.7	Cursor 中如何选择使用哪种大模型？	11
0.8	使用 Cursor 如何进行任务迭代？	11
0.9	如何在 Cursor 中使用 MCP？	12
0.10	如何以所见即所得方式精确控制页面元素更改？	13
0.11	让 AI 做 Code Review？	13
0.12	后话	13

## 0.1 Cursor 是怎么崛起的？

Cursor 的服务场景，最早是由纯文本编辑器 + 编译器完成的，俗称码代码，这也就是为什么早年的大神都瞧不起后面的小辈儿一样，纯手工古早味写代码，有几个能搞定？

后来，有人发现，在各个工具和窗口之间切换来切换去很麻烦，就有了 IDE 段想法，也就是 Integrated Development Environment（集成开发环境）。

IDE 把所有工具集中到一个窗口之内，这样程序员就不用在使用不同工具的时候切换来切换去了，很多内容也一目了然，极大提高了开发效率。

IDE 早年最有名的就是 VB/Delphi/JBuilder 这些，尤其 JBuilder 背后的 Borland 公司，还以研发工具开发出名，算是上个时代的 JetBrains 吧！

在 Java 领域，继 JBuilder 之后，又出现了 Netbeans、Eclipse 以及 IntelliJ IDEA，这些都是 IDE 的典范。其它领域，像 Visual Studio，像 XCode，这些也都是 IDE，所以，IDE 是现在大规模项目研发的主流。

当然，不是说有了 IDE 之后，文本编辑器类软件就没有人干了，这两个属于并行领域，毕竟，IDE 相对于文本编辑器来说还是比较重的，它不但包含文本编辑器的功能，还包含开发环境内相关的其它功能，所以，对于喜欢轻量的人来说，文本编辑器依然是最受欢迎的。

像 TextMate、Sublime Text 以及现在估计市场份额最大的 Visual Studio Code（简称 VS Code），都是不同阶段比较受欢迎的软件。

尤其是 VS Code，不但吸收了前面一些文本编辑器产品的优点，比如统一命令入口，比如插件体系，还推动了生态的繁荣。

当 2023 年开始，GPT 类 AI 产品崛起之后（尤其 ChatGPT），一个新的产品诞生了，那就是 Cursor。

Cursor 是基于 VS Code 打造的新一代开发工具，最直接的特征就是引入了 AI 来加持。

最早的 Cursor 其实只是在代码补全方面做了个微创新，用户只要写好注释（comment），说明自己想要什么功能，甚至函数只要输入个开头，剩下的就是直接按 TAB 键自动补全就行了。

但是，单凭这个功能创新并不能让 Cursor 脱颖而出，所以，有一段时间 Cursor 其实是沉寂的，直到…

直到 Cursor 经历了 2, 3 版的改版之后，引入了基于 Chat（聊天方式）的 AI 开发新模式，Cursor 才重新进入甚至开辟和引领 AI IDE 这个领域。

所以，Cursor 算是用 AI 重新打造一遍 IDE，用 AI 赋能开发人员。☒

tip

AI IDE 领域还有很多竞争者，但 Cursor 属于头部领先者，所以，我们才重点讲解 Cursor，至于其它 AI IDE，暂时来看，产品思路和特性基本上也大同小异，各位可以根据自己喜好选择就可以了。

简单罗列一下现在的 AI IDE：

Desktop	Web	插件类
Cursor	Bolt.new	Cline
Windsurf	Lovable	Roo Code
Zed	Firebase Studio(Google)	Augment Code
Trae(字节)		

## 0.2 Cursor 与 Agent 之间是什么关系？

我之前发过一句话，说 Cursor 其实就是开发场景里的 Agent，为什么会这么讲呢？

我在「大模型调教手册」里介绍过，Agent 其实就是围绕着大模型（LLM）打造的某种程序，虽然它有一些通用特征和流程，比如：

1. 感知环境
2. 推理思考
3. 执行和使用工具
4. 评估并给出结果

但拨去所有这些表象，它骨子里就是一段调用大模型的程序，核心能力完全来自大模型，而 Agent 的关键职能其实是外围的协调与调度。

我们可以为不同的场景开发不同的大模型程序，而这些不同场景里的大模型程序，其实就是 Agent(s)。

从这个角度来说，**Cursor** 就是面向开发场景的 **Agent** 程序。

它依然是围绕大模型在做事情，只不过，做的这些事情都是服务程序开发。

比如，它依然会自动索引代码库，然后把这些索引的代码库自动或者手动的喂给大模型作为上下文的一部分；

比如，它会在“背后”对多轮对话的历史进行总结（Summary）以避免其超出大模型的上下文窗口；

比如，它会考虑开发人员如何在不同的流程或者细分场景里使用 AI，然后引入多种使用模式（mode）；

比如，...

总之，它就是定位在程序开发这个场景的 Agent，所做的事情就是围绕 AI（LLM），在开发工具这块儿重新赋能新一代的研发人员。（当然，普通人用来写文章也不是不可以）

### 0.3 Cursor 三种模式有什么区别与妙用？

Cursor 的 Chat（聊天方式）有三种模式（Mode）：

1. Agent
2. Ask
3. Manual（原来叫 Edit）

那 Cursor 为什么要引入这么三种模式呢？它们之间又有什么区别？分别有什么妙用呢？

让我们带着这些问题一起来探索下..

首先，让我们回想一下，在做一件事情的时候，我们一般会采用什么流程或者过程？

通常，对于一个不太熟悉的事情或者领域，我们是不是得先做调研？

当调研做的差不多了，心里大体上对这个事情怎么做有谱儿了之后，我们才会着手做具体的事项，对吧？

然后就是不断的迭代和修正某些细节。

其实，Cursor 给出的这三种模式（Mode），个人感受跟我们做事流程中的不同阶段很吻合。

Ask 模式是只读模式，使用这个模式，我们可以先跟 AI 做对谈，这有点儿跟人聊或者上网查资料的意思，总之，就是对做一个事情先做个调研，先要自己大体上心里有谱儿。因为 Ask 模式是只读模式，不会对项目的内容做任何更改，所以，这个模式就很适合做类似调研的事情。

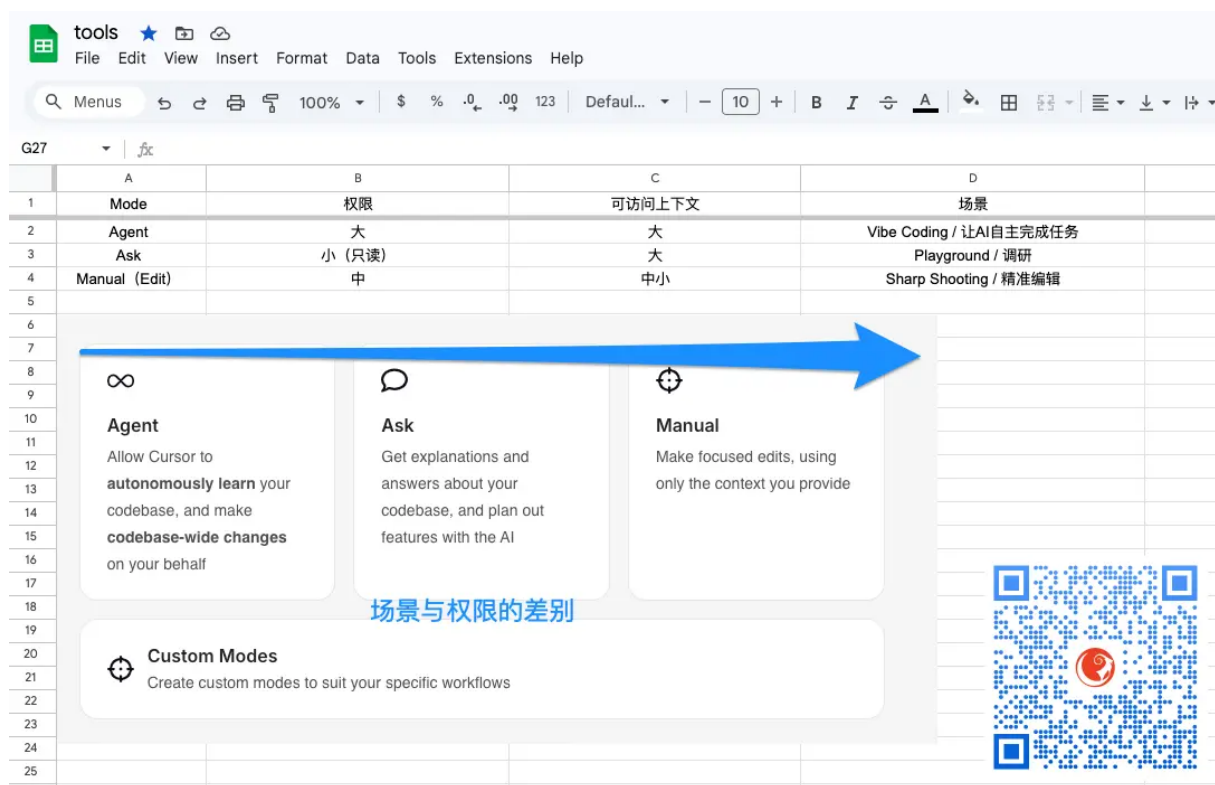
当我们对事情怎么做心里有谱儿了之后，就可以将模式切换为 Agent 模式了。

Agent 模式可以按照我们的指示（Prompt）开始干活儿了，这个模式下，Cursor 拥有最大的自主权，可以创建项目结构、内容甚至调用各种工具（比如命令行执行命令），总之，指示到位，具体任务怎么完成，它基本上可以安排的一步到位，再也不用有了一个 idea，就差一个程序员了 ☑

当项目基本成型之后，如果有后续需求要持续迭代，既可以用 Agent 继续迭代，我们也可以针对具体 bug 或者小功能项进行具体“狙杀”，这时候，我们就可以切换到 Manual 模式（原来叫 Edit 模式），然后引入尽可能少的必要的上下文，引导 Cursor 只对局部进行编辑。

这其实就是个理解的使用 Cursor 这三种模式的正确姿势。

最后，对三种模式做个总结和对比如下：



	A	B	C	D
1	Mode	权限	可访问上下文	场景
2	Agent	大	大	Vibe Coding / 让AI自主完成任务
3	Ask	小 (只读)	大	Playground / 调研
4	Manual (Edit)	中	中小	Sharp Shooting / 精准编辑


**Agent**  
Allow Cursor to **autonomously learn** your codebase, and make **codebase-wide changes** on your behalf

**Ask**  
Get explanations and answers about your codebase, and plan out features with the AI

**Manual**  
Make focused edits, using only the context you provide

**Custom Modes**  
Create custom modes to suit your specific workflows

场景与权限的差别



基本上是从三种模式的读写权限、可访问的上下文大小以及适合的场景三个维度进行的对比。

1. Agent 模式权限最大，可访问的上下文也最多，主力干活儿的模式；
2. Ask 模式只有只读权限（Read Only），可访问上下文跟 Agent 类似，所以，它不干活儿，但可以陪聊，适合调研和预演；
3. Manual 模式权限适中，一般会对给的上下文进行适当限定，目的是局部手术、精

准打击，适合有明确目标的修改和编辑；

就这么些，希望说明白了 ☺

## 0.4 Cursor Rules 是用来干嘛的？

Cursor 里的 Rules 的概念其实是为大模型（LLM）提供上下文内容的。

毕竟，你也不想每次聊天都适合都重新写一遍 Prompt 吧？这不成了跟傻子聊天了吗？完全记不住上次聊了啥。

所以，Cursor 引入了 Rules 的概念来记录和引用可以重用的 Prompt<sup>1</sup>。

Cursor 中的 Rules 分两类：

1. Project Rules
2. User Rules（也叫 Global Rules，即全局规则）

它们之间的区别如下：

Project Rules	Global Rules
Project scope	IDE scope
项目级别共享	个人全局共享
可以做Version Control	无法做Version Control
纯文本文件	配置项内容
.cursor/rules/目录下多个定义	Cursor Settings配置里Rules配置组下User Rules配置项

Project Rules 属于项目级别/范围的规则定义，通常放在 `.cursor/rules/` 目录下，属于纯文本内容（虽然后缀通常用 `.mdc`，即 Markdown for Cursor 的缩写），一般以 Markdown 格式编写。

比如：

```
☒ .cursor/rules/  
  - backend_structure_document.mdc  
  - app_flow_document.mdc  
  - tech_stack_document.mdc  
  - frontend_guidelines_document.mdc
```

---

<sup>1</sup>其实就是某种 Prompt Template

- backend\_structure.mdc
- implementation\_plan.mdc
- cursor\_project\_rules.mdc

因为属于标准文件和目录，所以，Project Rules 可以跟项目里其它物料（比如源代码、编译配置等）一起做版本管理（Version Control），团队协作的时候，这些 Project Rules 都可以共享。

#### tip

添加 Project Rules 的时候，也可以指定匹配规则，这样，Cursor 可以自动根据匹配规则关联相应的 Project Rules 作为上下文的一部分。

这有点儿类似 scala 里的 Partial Function 概念（如果你了解 scala 的话 ☺）

User Rules（全局规则）虽然是 IDE 范围内共享，但通常是针对单用户的，也就是使用当前 IDE 的用户使用，用当前 IDE 打开的多个项目也可以共享使用，但 Users Rules 的可迁移就没那么灵活了。

User Rules 需要到 Cursor Settings -> Rules -> User Rules 配置项下配置，内容格式一般也使用 Markdown 格式。

所以，团队协作的项目，首选 Project Rules 来共享可复用的上下文内容，只有开发人员个人喜好的设定才建议放到 User Rules 里。

#### tip

我们当然可以从头开始编写自己的 Rules，但很多通用场景下的 Rules 已经有人写好了，所以，直接借用过来根据情况修改下就可以了（或者直接用），比如：

- <https://cursor.directory/rules>
- <https://github.com/PatrickJS/awesome-cursorrules>

去 Github 搜 Awesome cursor rules，应该也会搜到很多别人已经编写好的 Rules。

从另一个角度来说，私有的 rules 反而更有价值了：

为什么 Cursor Rules 有价值？

AI 工具里面最重要的地方是里面隐藏的专业知识，而 Cursor Rules 里面则可以定义大量的专业知识给到 AI 在不同上下文中准确地使用到。



那有了这些 Rules 定义，如何引用它们呢？

下一篇马上开讲…

## 0.5 Cursor 里如何引用各种上下文内容？

Cursor 提供了一个统一的引用上下文内容的触发符号，即 @

在聊天窗口中，只要敲入 @ 字符，Cursor 会自动弹出菜单，供我们选择通过什么方式引入不同的上下文内容，这些不同方式包括但不限于（仅罗列比较典型的几种，完整类型请参考 Cursor 官方文档 或者自己直接在 Cursor 中选择）：

- @Files 指定使用哪些文件内容作为上下文内容
- @Folders 指定使用哪些文件目录内容作为上下文内容范围
- @Code 指定使用哪些代码块内容作为上下文内容
- @Docs 指定使用哪些文档库内容作为上下文内容，可以在 Cursor 中添加像 Tailwindcss 等框架的文档并引用
- @Git 可以指定 Git 相关的内容做上下文内容，比如哪次提交的内容
- @Web 是 Web Search，也就是会调用搜索引擎来查找内容并作为上下文内容
- @Link 指定某个链接的内容作为上下文内容的一部分
- @Cursor Rules 指定使用哪个 Rules 内容作为上下文一部分（有些 Project Rules 可以自动匹配，这里更多是手动明确指定）
- …

以上内容简单了解下就好了，因为在 Cursor 里直接操作就可以选择了，so easy～

tip[通过 @ 与 # 引入上下文有什么区别？]

Cursor 里除了可以用 @ 来引用上下文内容（最常用），也可以用 #，那它们之间有啥分别呢？

- @ 其实更多是限定上下文内容的范围
- # 则是直接指定选定的文件内容作为上下文内容

@ 和 # 其实可以配合使用，比如使用 @ 限定范围，然后用 # 选定这个限定范围内的某个文件。

## 0.6 如何使用 **Cursor** 完成复杂任务和项目？

大模型的典型限制是什么？

上下文窗口大小，对吧？

什么叫复杂任务和项目？

内容很多，对吧？

那如何弥合二者之间的矛盾呢？

打造一个列表，然后让大模型一个一个干就好了。

所以，使用 Cursor 完成复杂任务和项目，通常的做法是：

1. 编写一个 PRD 文档 (Product Requirement Definition)，当然，这个可以自己写，也可以让 AI 代劳 ✎
  - e.g. Can you generate me a PRD using the @example-prd.txt as an example and then save it as PRD.txt in @scripts
2. 通过 Prompt 指示 AI 按照 PRD 的定义，逐个完成 PRD 定义里的任务（或者手动挑选完成），aka. Implementation Plan
  - e.g. parse @PRD.txt and generate tasks, 然后 finish tasks as per the task list and start from task 1
3. 检查和 review 并局部完善和修正。
  - e.g. check your progress on the tasks and continue, 过程中如果有发现有错误，可以中止，然后根据 task 序号独立指定 agent 重新只针对特定 task 开始干活儿。

<https://github.com/eyaltoledano/claude-task-master> 就是满足这个场景需求的一个工具项目，也可以配合 MCP 使用。

通过 claude-task-master，我们可以罗列任务，可以挑选任务让 AI 去完成，而这些，都可以通过自然语言指示完成。

总之呢，要让 Agent 模式真正发挥最大作用，PRD 文档不能少 ✎

把复杂拆解为简单的任务，然后一个个干就可以了。

## 0.7 Cursor 中如何选择使用哪种大模型？

这个问题分两个层面来看…

假如你问的是 UI 操作上怎么选择，那么，只要打开 Chat 窗口，在最下面模式选择按钮的旁边，点击选择按钮选择模型就可以了，默认是 auto，也就是 Cursor 会帮我们自动选择最合适的模型。

也可以使用快捷键 CMD+K(MacOS 上) 打开模型选择下拉框。

假如你问的是如何从各种大模型中选择合适的模型，那么，我们就可以多展开一些。

截止这篇文章成型，现在排在头部的研发类大模型分别是：

1. Gemini 2.5 Pro 系列
2. Claude 3.5 / 3.7
3. DeepSeek V3 / V3 0324

当然，OpenAI 的 o1 系列也很不错。

在大模型家族里，经常提到一类模型叫 Thinking 模型或者 Reasoning 模型（推理模型），以上都属于这一类模型，这类模型的特点是，可以处理复杂任务的拆解和规划，所以，特别适合配合 Agent 模式一起使用。

假如给大模型分分类，那么，Thinking 模型或者说推理模型类似于原来研发团队的架构师，而传统的 GPT 类模型，则类似工程师。

也有人喜欢用慢思考和快思考里的系统一和系统二来类比，原因就在于 Thinking 模型（推理模型）需要的处理时间会更多，表现的更慢，但出来的结果可以更复杂、更完善，而简单的 Zero-shot 类请求，一般的基础模型则可以更快出结果。

对于调研类或者简单对话类场景，通常用一般的基础大模型就可以了（虽然 Thinking 模型也 ok，就是慢点儿），对于复杂任务，则必须交给 Thinking 模型（推理模型）。

tip

OpenAI 有文档介绍 Reasoning 模型的最佳实践，大家感兴趣也可以看下：  
<https://platform.openai.com/docs/guides/reasoning-best-practices>

## 0.8 使用 Cursor 如何进行任务迭代？

软件研发的一个关键特性是可以持续迭代。

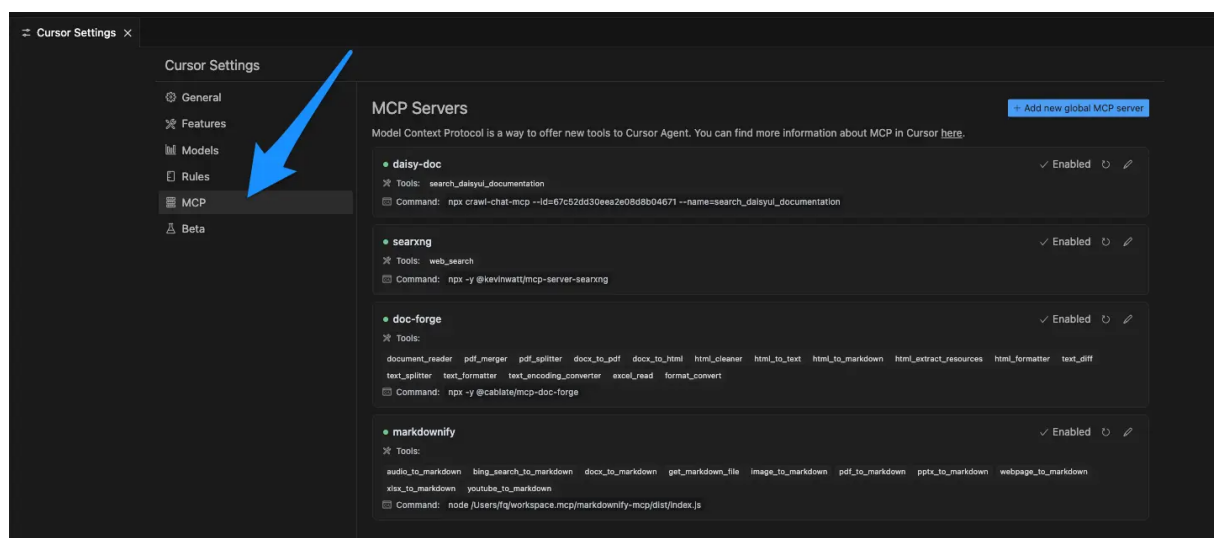
虽然 Cursor 提供了 Checkpoint 这个特性，我们觉得生成的结果不满意的时候，可以使用这个特性回滚（rollback），但最常见的实践，依然是配合版本控制系统（比如 git）做阶段性的结果封印。

当生成的结果通过 review 和初步测试之后，就可以作为一个 snapshot 进行版本控制，提交之后，再进行下一个任务。

这样可以最大限度的保证阶段性的结果可用。

## 0.9 如何在 Cursor 中使用 MCP？

Cursor Settings 中有专门的 MCP 设置入口：



进入 MCP 设置点击右上角的 `Add new global MCP server` 按钮添加自己想用的 MCP 服务就可以了。

关于 MCP 的更多信息，建议参考“架构百科的 MCP”相关条目。

大部分情况下，

使用 stdio 传输模式的 MCP 服务器在本地使用是最常见的方式，SSE 模式的 MCP 服务更多适合那些企业内部或者公开提供服务的场景。

### note

Cursor Settings 的 Features 设置里，在 Chat 设置下有一个 `Enable auto-run mode` 设置，原来叫 `yolo-mode`，这个设置如果启用，MCP 服务调用中间需要人工确认授权的步骤会被省略，假如出于安全或者其它层面的考虑，不希望 MCP 中间步骤

不经确认就自动执行，可以启用下面的 MCP tools protection 设置。

默认 MCP 协议其实是有要求中断与人工确认的环节。

## 0.10 如何以所见即所得方式精确控制页面元素更改？

最近新出了一个 VSCode 的 Extension 叫 stagewise，确切的说，这不是 Cursor 专用，但目前它只支持 Cursor 和 Windsurf，而且感觉对于前端 web 开发很有用的样子，所以跟大家介绍下。

stagewise 的实现思路是在 Web 应用开发期间（dev stage），在页面上设置一个工具栏（toolbar）可以用来输入更改请求（类似跟 agent 聊天的形式），当用户选择了页面上某个元素之后，通过输入栏输入更改请求，点击更改之后，选中的元素以及更改请求等都会被一起采集作为发送给 AI Agent 的上下文信息。

stagewise 工具栏会把这些上下文信息发给它自己的 VSCode Extension（对，你还得先安装一个 stagewise 的 VSCode Extension），然后这个 stagewise 的 VSCode Extension 收到上下文信息之后，再转发给 Cursor 或者 Windsurf 的 Agent 进行处理。

处理结束后的响应内容会再次交给 stagewise 的 VSCode Extension，然后返回给工具栏去应用对哪些页面元素的变更。

整个过程不走 Cursor 或者 Windsurf 的输入栏，而直接用它们的输入与 Agent 处理能力，整个过程其实是从 stagewise 的工具栏发起，在工具栏和页面上工作的。

真正实现了所见即所得的更改和页面编辑。

## 0.11 让 AI 做 Code Review ?

CodeRabbit 

## 0.12 后话

持续内容更新，尽在「福强私学」

福强私学  
传道授业  
解惑



早一天入手  
早一天受益

福强老师更多内容产品：

## 全网原创内容矩阵zhui全面



# 王福強个人出品

<https://afoo.me>

