AI编程与智能体开发

林子雨　副教授　厦门大学
E-mail: ziyulin@xmu.edu.cn

2026年9月

主讲教师

- 2013年度厦门大学奖教金获得者
- 2017年度厦门大学奖教金获得者
- 2020年度厦门大学奖教金获得者
- 2023年度厦门大学奖教金获得者
- 2026年度厦门大学奖教金获得者
- 2018年国家精品在线开放课程
- 2020年国家级线上一流本科课程
- 2023年国家级线上一流本科课程
- 2020年福建省线上一流本科课程
- 2021年福建省线上线下混合一流本科课程
- 2018年福建省高等教育教学成果奖二等奖
- 2022年福建省高等教育教学成果奖特等奖
- 2024年福建省高等教育教学成果奖特等奖
- 2023年教育部国家智慧教育公共服务平台应用典型案例

配套教材

教材目录

第1章 大模型
第2章 AI编程概述
第3章 AI编程环境搭建
第4章 基于提示词的AI编程
第5章 基于规范的AI编程
第6章 流程驱动的AI编程
第7章 智能体概述
第8章 智能体开发框架LangChain
第9章 智能体开发框架LangGraph

教材特色

- 教材内容来自互联网大厂真实实践
- 以实践教学为依托
- 理论实践贯通
- 覆盖主流技术

《AI编程与智能体开发》

林子雨 编著

人民邮电出版社 预计2026年11月上市销售

教材官网提供详细信息和样书申请

官网： https://dblab.xmu.edu.cn/post/ai-programming-agent/

扫码访问教材官网
了解详情、获取资源、申请样书

第5章
基于规范的AI编程

目录

Part 01

概述

5.1 概述

从个人提效到团队提效

基于提示词的AI编程的不足

规范驱动编程的核心理念

与提示词编程的关系

5.1.1 从个人提效到团队提效

01 个人使用 AI 编程工具时，可通过优化提示词写作技巧，提升 AI 代码采纳率，实现个人开发效率的显著提升。

02 个人 AI 编程效率可通过提示词优化提升，但该效果无法简单对等推演到团队层面，团队整体 AI 编程效率难以实现同等提升。

03 团队 AI 编程效率无法同等提升的核心原因：团队协作存在根本性挑战，这类问题无法仅依靠提升个人提示词技巧解决。

04 传统软件工程沉淀了成熟的团队对齐解决方案，包含编码规范、代码审查、设计文档、持续集成（CI）等机制。

05 传统软件工程各类团队协作机制的核心特征：不依赖个人最佳表现，为团队产出设定统一质量下限，保障团队整体产出质量。

06 规范驱动编程的定位：并非提示词编程的升级版或替代品，是适配 AI 编程场景、引入团队协作保障机制的全新方法论。

07 规范驱动编程的核心思路：借鉴传统软件工程经实践验证的规范文档化、流程结构化、质量可验证思路，并适配 AI 编程场景。

5.1.2 基于提示词的AI编程的不足

在团队协作中，常见以下情况：个人使用AI编写代码时效率很高，但与团队成员协作时却出现各种不对齐；新成员接手时，难以理解之前使用AI编写的代码逻辑。

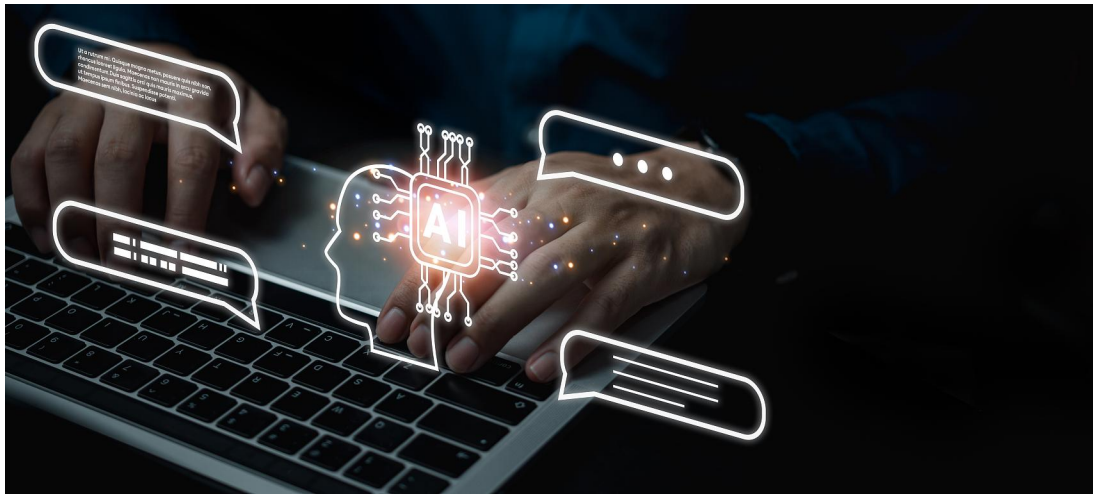
第4章的技巧是有效的，但它们有一个共同的前提：你（个人）知道项目要做什么，能为AI提供完整的上下文，能验证AI的输出是否符合预期。当这个前提在团队协作场景中不再成立时，提示词编程就会面临以下五个系统性问题：上下文无法跨会话持久化、规范以隐性知识的形式散落、需求理解偏差在代码审查阶段才被发现、质量保障依赖个人自觉而非流程机制、经验无法转化为可传承的工程资产。这些问题指向同一个根本原因：**提示词编程是一种"个人活动"的放大器，它把个人的技巧和判断放大N倍，但没有解决"团队如何对齐"这个本质上属于组织工程的问题**。

5.1.2 基于提示词的AI编程的不足

■ 问题一：上下文无法跨会话持久化

AI每次对话都是独立的，没有跨会话的持久记忆。当你开启一个新对话，或者团队其他成员接手时，之前积累的所有上下文（如项目背景、技术约束、关键决策等）会全部消失。每个人都需要从零开始向AI解释项目情况，效率显著降低。

第4章介绍了项目说明书（AGENTS.md）和规则（Rules）机制来应对这一问题。但这些机制的本质是"让每个开发者自行维护上下文配置"，在团队中，这意味着每名成员都需要理解并正确配置这些文件，这本身就需要培训和纪律保障。更关键的是，当团队成员对项目背景的理解出现偏差时，即使每个人都正确配置了各自的上下文，AI在不同成员的对话中仍然可能生成风格不一致的代码。

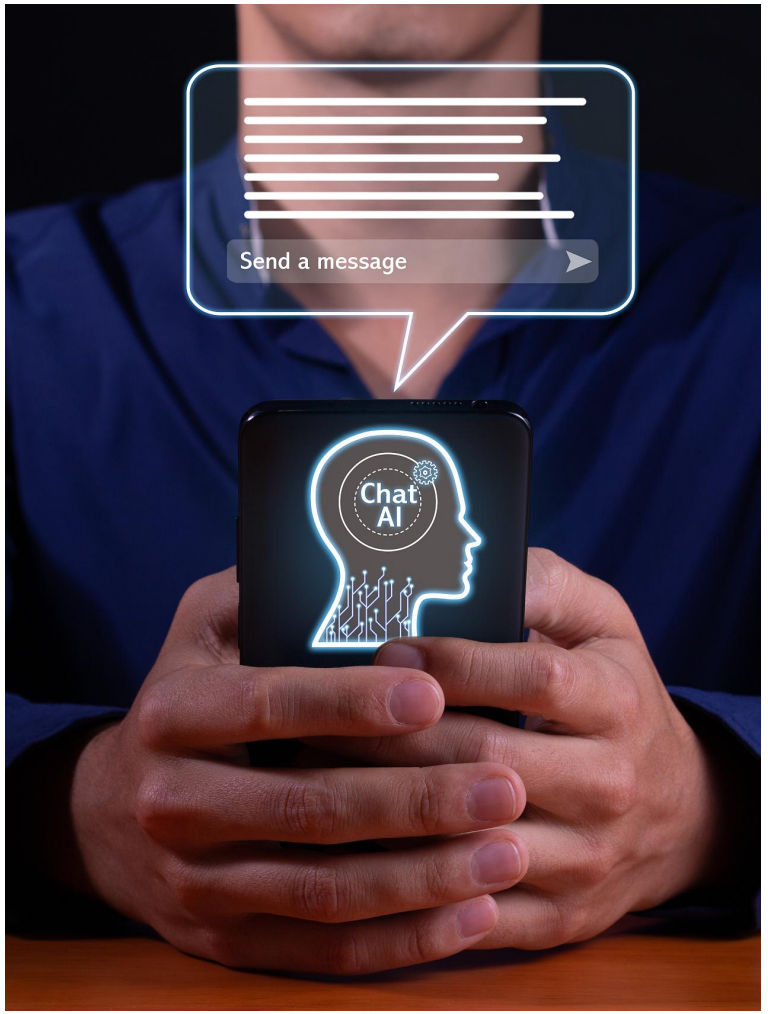


5.1.2 基于提示词的AI编程的不足

■ 问题二：规范以隐性知识的形式散落

第4章强调了在提示词中明确说明约束和边界条件的重要性。但在团队实践中，每个人对"什么需要明确说明"的理解并不一致，有人会写出详细的约束列表，有人只会写一句"参考项目中现有的实现"。这种差异导致了两个后果：第一，AI 生成的代码质量高度依赖个人技巧，团队内部的下限参差不齐；第二，最佳实践以隐性知识的形式存在于个人对话历史中，新成员无法直接获取。

以第4.3.2节的批量短信发送为例，并发安全、幂等性、重试机制、限流控制这四个边界条件，对于缺乏生产环境经验的初级工程师而言，通常不会在提示词中明确写出。而团队中经验丰富的工程师可能已通过实践总结出这些经验，但不可能每次都在提示词中详尽列出，这并非习惯问题，而是因为这些经验从未被正式文档化。



5.1.2 基于提示词的AI编程的不足

■ 问题三：需求理解偏差在代码审查阶段才被发现

即使团队成员都掌握了高质量提示词的写作技巧，AI仍然可能在"理解"需求时出现偏差。第4章将这种现象命名为"语义漂移"，即AI在模糊需求面前自动填补假设，导致输出结果与真实意图逐渐偏离。

在个人场景下，语义漂移的代价相对可控，因为你和AI在同一个对话中，可以随时发现偏差并纠正。但在团队协作中，情况更为复杂，团队成员A用AI生成了代码，团队成员B在此基础上继续开发，双方对"某个模糊需求"的理解是否一致可能并不清楚。更糟糕的是，这种偏差往往在集成测试或代码审查阶段才被发现，返工成本远超预期。

5.1.2 基于提示词的AI编程的不足

■ 问题四：质量保障依赖个人自觉而非流程机制

提示词编程的质量保障机制，本质上是"依赖每个开发者自觉写出高质量的提示词"。虽然存在多种提示词框架和工具，但它们是建议而非强制，团队中没有机制确保每个人都按提示词框架来写作提示词。

这在个人场景下不是问题，因为是你自己对自己写的提示词负责。但在团队中，这意味着AI输出的质量完全取决于每个人的技巧水平。一名刚加入团队、对项目尚不熟悉的新人，其提示词质量往往远低于团队平均水平，而他生成的代码在进入代码审查之前就已经埋下了隐患。

5.1.2 基于提示词的AI编程的不足

■ 问题五：经验无法转化为可传承的工程资产

用提示词编程产生的团队经验——哪些提示词模式效果好、哪些边界条件容易被忽略、哪些功能用AI实现时需要特别关注，往往以对话历史的形式存在。这些经验无法像代码一样被版本化管理，也无法像文档一样被新成员快速学习。人员变动时，这些经验随之流失，团队需要重新经历类似问题来积累相同的认知。

5.1.3 规范驱动编程的核心理念

规范驱动编程（Spec-Driven Development，SDD）的核心理念可以用一句话概括：在让AI编写代码之前，先用结构化文档明确阐述"做什么、怎么做、有何约束"，然后AI围绕这份文档工作，输出结果必须与文档严格对齐。

这一理念最早由美国技术创业者与AI研究者Sean Grove在2013年的"The New Development Workflow"系列文章中系统阐述，又在AI编程工具兴起的背景下被重新发现和扩展。Grove在文章中提出了一个深刻的观察："代码是意图的有损投影。当我们把想法转化为代码时，大量上下文信息会丢失，比如为什么这样做、权衡了哪些方案、考虑了什么约束。最终代码只保留'怎么做'，却丢掉了'为什么这样做'"。

规范驱动编程试图解决这个"有损投影"问题，它的核心思路是：把规范文档作为软件开发的第一性产物，代码是规范的一个实现结果，而非唯一的知识载体。

5.1.3 规范驱动编程的核心理念

规范驱动编程有三条铁律，它们共同构成了规范驱动编程的行为准则（见下图）：

三大铁律	铁律一：无规范不写代码	没有规范文档，不准开始写代码
	铁律二：规范即真理	规范文档是最高权威
	铁律三：逆向同步	发现Bug，先修规范，再修代码

5.1.3 规范驱动编程的核心理念

铁律一
无规范，不写代码

没有规范文档，不得开始写代码。这并非要求编写一份滴水不漏的需求文档，而是要求在让AI实际生成代码之前，先构建一份结构化的规范文档。这份文档明确定义了功能范围、输入输出、边界条件和验收标准，即使AI生成的代码与预期存在偏差，规范文档即是判断正确与否的共同基准。

铁律二
规范即真理

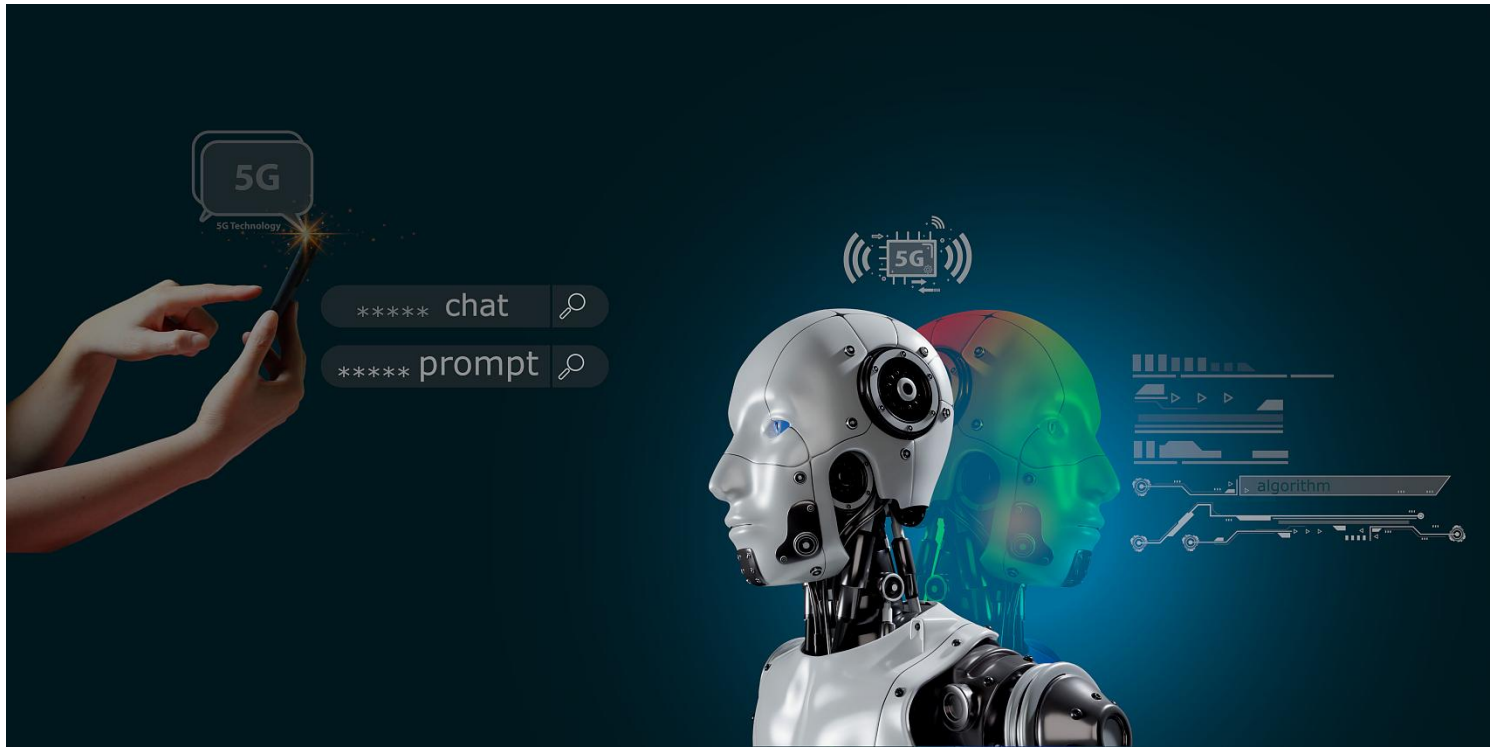
规范文档是最高权威。当代码行为与规范不一致时，错的永远是代码，而非规范。这条铁律的价值在于：它消灭了"规范说一套、代码做一套"的灰色地带。如果代码和规范对不上，就改代码，除非规范本身也需要修改（那就先改规范，再改代码）。

铁律三
逆向同步

发现Bug时，先修规范，再修代码。这条铁律看似违反直觉——表面上看Bug出在代码，实则根源在规范。原因在于，Bug的出现，通常意味着规范中有描述不清的地方。如果不先完善规范，即使修复了这个Bug，同类问题还会在其他地方出现。正确的做法是：先审视规范，找到描述不清的地方，更新规范，再基于更新后的规范修复代码。

5.1.4 与提示词编程的关系

在深入学习规范驱动编程之前，需要先澄清一个常见的误解：**规范驱动编程并不是要取代提示词编程，两者解决的是不同层面的问题**。提示词编程解决的是"如何与AI沟通"的问题，即通过高质量的输入获得高质量的输出，它关注的是提示词的内容、结构、上下文传递技巧。规范驱动编程解决的是"团队如何协同"的问题，即如何让团队成员在统一的框架下使用AI编程工具，如何将个人经验转化为团队可复用的知识资产，如何建立不依赖个人技巧的质量保障机制，它关注的是**流程、文档、标准和团队协作机制**。



5.1.4 与提示词编程的关系

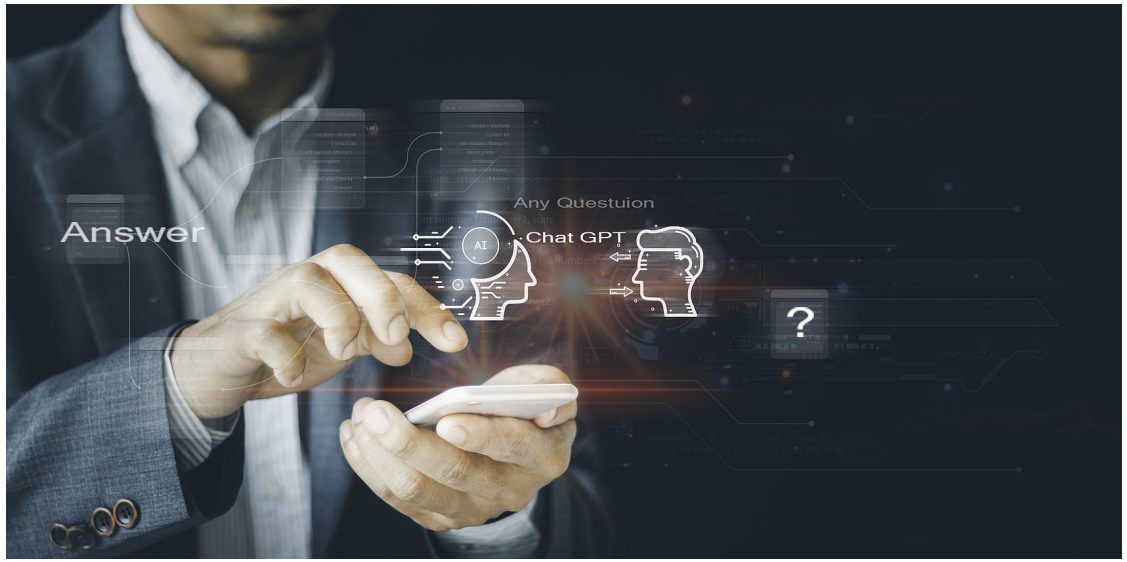
下表给出了二者的详细对比。

维度	提示词编程	规范驱动编程
核心目标	提高个人AI编程效率	建立团队AI编程保障机制
主要产出	高质量代码	规范文档 （规范是核心，代码是结果）
AI自由度	高，AI在给定范围内自由发挥	低，AI严格按规范文档执行
团队协作	依赖个人技巧，难以标准化	共享规范，统一标准，可追溯
知识沉淀	散落在对话历史中	规范文档纳入版本控制
适用场景	个人快速开发、探索性任务	团队协作、生产级项目交付
学习曲线	低，上手快	高，需要掌握规范写作和流程管理
质量保障	依赖个人提示词质量	依赖规范质量和流程执行

5.1.4 与提示词编程的关系

实际上，规范驱动编程内置了提示词编程的技巧。在规范驱动编程的工作流中，你需要用第4章学到的结构化提示词技巧来编写规范文档，也需要在AI执行规范时提供清晰的上下文，范驱动编程本质上也是基于提示词的编程，规范可以被看作是结构化表示的提示词技巧。

在实践中，提示词编程和规范驱动编程并非互斥。可以先用提示词编程快速验证想法（这种编程方式也被称为“氛围编程（Vibe Coding）”），验证可行后，将经验提炼为规范，再用规范驱动编程交付生产级代码（Spec Coding）。



Part 02

规范驱动编程

5.2 规范驱动编程

什么是规范驱动编程

为什么需要规范驱动编程

规范驱动编程的五大优势

三种规范驱动方法论

5.2.1 什么是规范驱动编程

规范驱动编程（Spec-Driven Development，SDD）是一种以规范（Specification）作为软件开发第一性产物的开发方法论。在规范驱动编程中，规范不是辅助性的参考文档，而是整个开发过程的起点和终点。起点是因为一切从规范开始，终点是因为代码最终要回到规范来验证。

为了理解规范驱动编程的本质，这里将其与传统开发方法做一个对比：

传统开发的典型流程

需求文档→编码→测试→交付。在这个流程中，需求文档往往在编码开始后就被束之高阁，代码逐渐偏离原始需求，最终交付的系统与最初的需求之间可能出现显著偏差。

VS

规范驱动编程的典型流程

规范文档→编码→验证规范→交付。在这个流程中，规范文档在整个开发周期中持续发挥作用，编码前提供方向，编码时提供标准，验证时提供对照。每次代码变更都要检查是否与规范一致，任何偏离都要被记录和修正。

5.2.1 什么是规范驱动编程

规范驱动编程包含三个核心关键词：

第一性产物

规范是整个开发过程的起点。传统开发是"代码优先"（Code-First），规范驱动编程强调"规范优先"（Spec-First）。这里的"规范"不是指产品需求文档，而是技术层面的、包含可验证验收标准的结构化文档，它描述系统应该做什么、如何做、达到什么标准才算完成。

结构化与可验证

规范不是自然语言描述的散文，而是结构化的、可被工具解析的、可直接转化为测试用例的文档。一份高质量的规范应当包含明确的输入输出定义、详细的验收标准列表、清晰的边界条件说明。这些内容不仅人能读懂，AI也能准确理解并据此生成代码。

可演进

规范不是一次性产物，而是随着项目演进持续更新的资产。当需求变化时，先更新规范，再更新代码。规范变更触发相应的代码变更，形成"规范-代码"的双向同步。当代码中发现问题时，先审视规范是否描述不清，完善规范后再修复代码。

5.2.2 为什么需要规范驱动编程

要理解规范驱动编程的价值，我们需要先理解传统AI编程中"只写代码、不写规范"会带来什么问题。

假设某开发团队需要实现一个用户注册接口。团队中的工程师A编写了详细的提示词，AI生成了代码，测试通过。三个月后，工程师B需要修改该接口，由于无法查看工程师A与AI的原始对话历史，工程师B只能基于现有代码让AI进行调整。又过了两个月，第三方安全审计发现该接口存在SQL注入漏洞——原来工程师B让 AI做的"调整"中，无意间引入了一段字符串拼接的SQL代码。

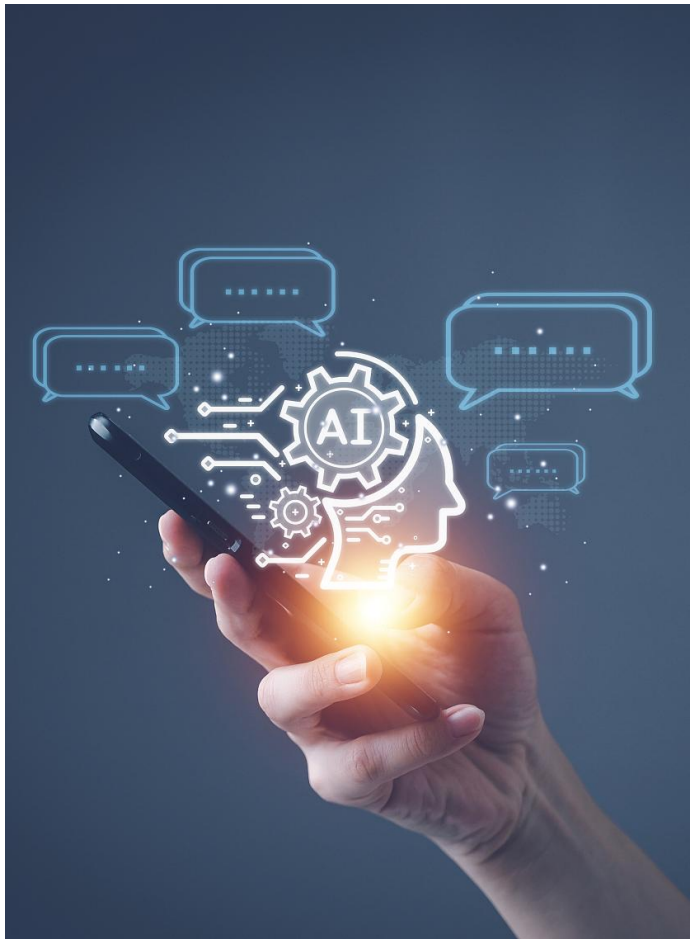
这个问题为什么会发生？因为，关于"用户注册接口不允许使用字符串拼接 SQL"这个约束，只存在于工程师A原始的提示词中，当团队成员变动、对话历史丢失后，这个约束就随之消失了。AI在下一次生成代码时，不会主动"记得"这个约束，除非有人在提示词中再次明确说明。

5.2.2 为什么需要规范驱动编程

规范驱动编程解决的正是这样一个问题：将约束、决策和规范从个人经验与对话历史中抽离，固化为团队共享的、版本化的、可传承的结构化文档。

OpenAI研究员Sean Grove在阐述这一理念时，用了一个非常形象的比喻："想象你在一家餐厅后厨工作。每位厨师都有自己的烹饪习惯和操作要点。当一位新厨师加入时，他只能靠观察和猜测来学习——观察其他厨师怎么做，猜测某些步骤背后的原因。成品可能相似，但难以保证一致。现在换一种方式，把每位厨师的操作要点都写下来，形成一本标准的'菜谱规范'，新厨师加入时，直接按照菜谱规范做菜。任何对菜谱的修改都要经过讨论和验证，然后更新菜谱。最终顾客在任何时候、由任何厨师做出的同一道菜，出品质量保持一致"。

规范驱动编程就是软件开发中的"菜谱规范化"，它确保不同的人、在不同的时间、用不同的AI工具，都能产出质量一致、风格一致、行为一致的代码。



5.2.3 规范驱动编程的五大优势

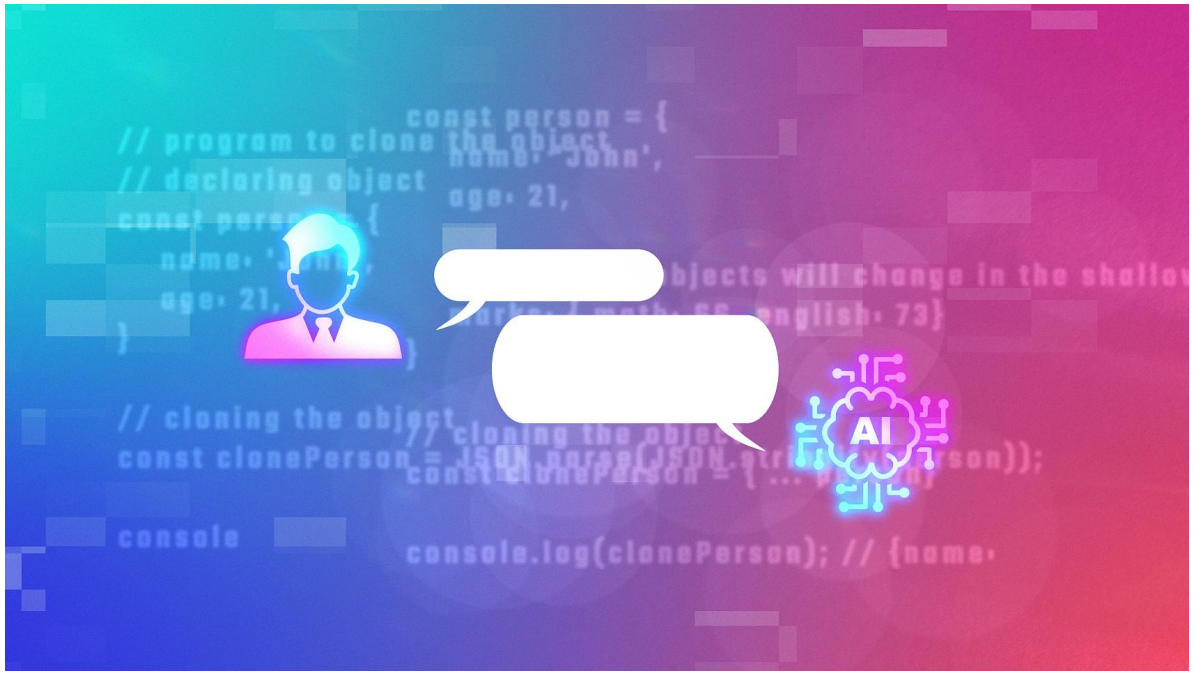
提示词编程的问题	规范驱动编程的优势
问题一：上下文无法跨会话持久化	优势一：一致性——规范统一了上下文，所有成员在同一基准线上工作
问题二：规范以隐性知识散落	优势二：知识沉淀——隐性知识被固化为显性文档，新成员可快速获取
问题三：需求理解偏差在审查阶段才发现	优势三：零偏差——规范是唯一的判断标准，偏差在编码前即被发现
问题四：质量保障依赖个人自觉	优势四：可追溯——规范纳入版本控制，质量保障由流程而非个人驱动
问题五：经验无法传承	优势五：效率提升——经验沉淀为可复用规范，新成员无需从零开始

5.2.3 规范驱动编程的五大优势

■ 优势一：一致性——团队在同一基准线上工作

规范是团队共享的事实来源。所有成员在同一起点上使用AI编程工具——相同的规范文档、相同的验收标准、相同的质量要求。这意味着AI输出的代码风格、质量水平自然对齐，不需要依赖每个人的个人技巧来保障下限。

对于AI编程工具来说，规范的存在还有一个额外价值：AI的输入变得更加稳定。当团队成员A提交了一个规范文档，AI可以在不了解"这是谁写的"的情况下，直接根据规范生成符合团队标准的代码，不再依赖于每个人的提示词写作水平。



5.2.3 规范驱动编程的五大优势

■ **优势二：知识沉淀——形成可传承的工程资产**

规范文档纳入版本控制，与代码一起演进。每次“踩坑”后的经验、每个设计决策的理由、每个边界条件的处理方式，都沉淀在规范文档中。新成员加入团队时，通过阅读规范文档就能快速理解项目的设计决策和实现标准，而不需要翻阅大量的代码和对话历史。

规范文档还有一个独特的价值，它是决策的记录，而不仅仅是决策的结果。当团队日后需要回顾"为什么这个功能这样实现"时，规范文档能给出清晰的答案——不是"代码是这样写的所以这样实现"，而是"经过分析，我们认为这样实现最合理，因为……"。

5.2.3 规范驱动编程的五大优势

■ 优势三：零偏差——规范是唯一的判断标准

规范明确定义了系统应该做什么、不应该做什么、做到什么程度才算完成。AI生成代码时，严格按照规范执行，偏差概率大幅降低。更重要的是，当偏差出现时，规范即是判断正确与否的唯一标准，无需在代码审查时争论原始意图，因为意图已在规范中明确阐述。

这种"零偏差"不是指AI完全不会犯错，而是指偏差更容易被及时发现和纠正。因为规范文档在代码之前就已经写好，当代码与规范不一致时，人或AI都可以通过对比规范和代码来发现差异。

5.2.3 规范驱动编程的五大优势

■ **优势四：可追溯——任何变更都有据可查**

任何代码变更都可以追溯到对应的规范变更。任何功能的行为都可以通过规范来解释和验证。当系统出现问题时，首先检查规范是否被正确实现；当需要重构代码时，首先审视规范是否需要更新。

这种可追溯性在团队协作中尤为重要。当两个模块之间的接口出现不一致时，规范文档可以明确指出"模块A的接口是根据规范X实现的，模块B的接口是根据规范Y实现的，两者不一致是因为规范X和规范Y之间存在冲突"，而这种分析在纯代码环境中是难以做到的。

5.2.3 规范驱动编程的五大优势

■ 优势五：效率提升——对话轮次大幅降低

经验数据显示，使用规范驱动编程后，典型功能的AI对话轮次从15～20轮降至3～5轮。这是因为**规范文档一次性解决了大量"来回沟通"的问题**，输入输出格式、边界条件、验收标准、代码风格，所有这些原本需要在多轮对话中逐步澄清的内容，在规范文档中已经明确定义。

当然，规范文档本身需要前期投入来编写。但这笔投入是值得的，编写规范的时间远小于后续多轮返工的时间。更重要的是，**规范文档是一次性投入、长期收益，一份好的规范可以在多个项目中复用，也可以在团队中传承。**

5.2.4 三种规范驱动方法论

■ 类型一：结构化规范

结构化规范（Structured Specification）是最灵活的一种方式。不强制要求固定的文档格式，而是通过结构化的字段来组织规范内容——功能需求、输入输出、验收标准、技术约束、边界条件等。每个字段都有明确的内容要求，但不限制具体的写作风格。

这种方式的代表工具是OpenSpec（第5.3.3节将详细介绍），它使用单一 spec.md文档管理所有规范，通过目录结构（即用changes目录记录活跃变更，用 archive目录归档知识库），组织规范的演进。

5.2.4 三种规范驱动方法论

■ **类型二：分阶段规范**

分阶段规范（Phased Specification）是更正式的一种方式。它将规范过程分解为多个阶段，每个阶段有明确的产出物和门控（Gate）检查，规范在阶段之间递进演进，从高层次的系统需求到低层次的具体实现。

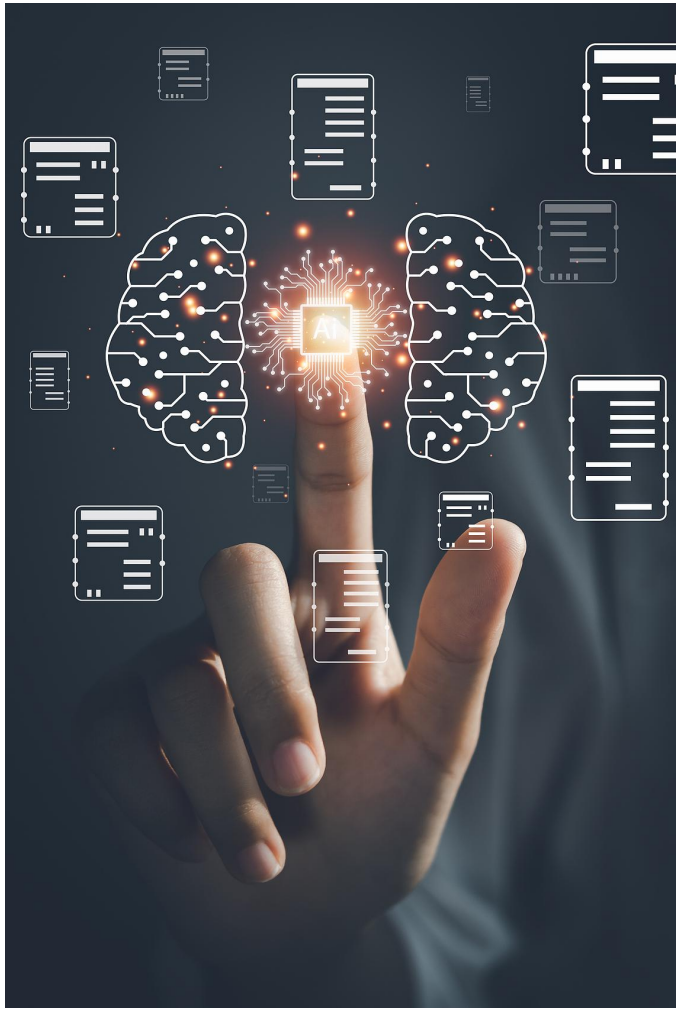
这种方式的代表工具是Spec-Kit（第5.3.2节将详细介绍），它定义了五层规范链路：Constitution（工程宪章）→ Specification（系统级规范）→ Plan（实现计划）→Tasks（任务级拆分）→Implementation（实现）。

5.2.4 三种规范驱动方法论

■ 类型三：嵌入式规范

嵌入式规范（Embedded Specification）是最激进的一种方式。将规范直接嵌入到代码中，通过代码注释和类型注解来表达规范内容。AI工具直接读取代码中的规范信息，据此生成或审查代码。

这种方式的优势是无需维护单独的规范文档，规范信息始终与代码同步；劣势是规范信息可能变得冗长和难以阅读，对于复杂项目的整体规划能力较弱。



5.2.5 规范驱动编程的运作机制

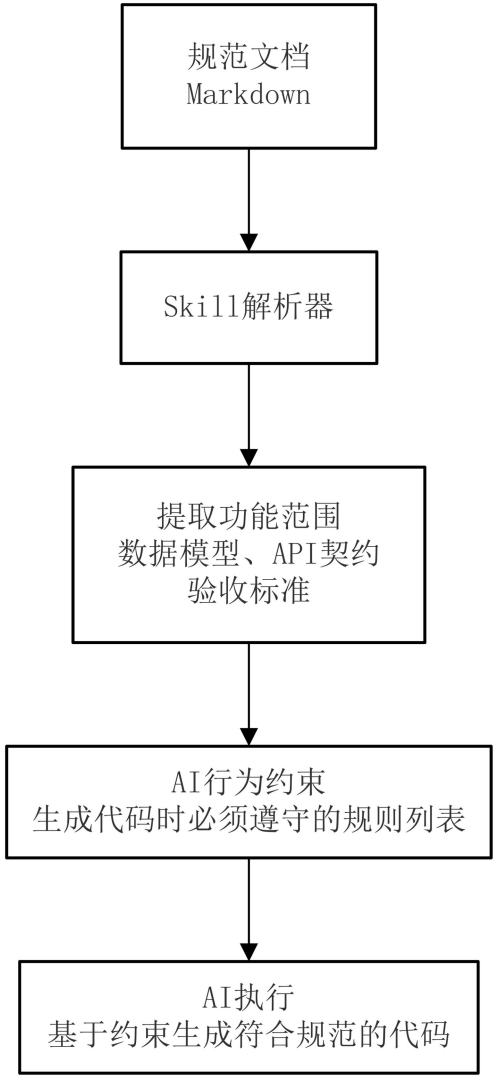
规范驱动编程的核心机制是Skill（技能）——一种可复用的、结构化的AI行为约束模板。Skill的本质是"预编写的提示词模板+执行逻辑"，它将规范文档中的结构化信息转化为AI可以理解和执行的指令。

Skill是规范驱动编程从"理念"走向"工程实践"的关键桥梁。没有Skill，规范只是一份静态文档；有了Skill，规范才能动态地约束AI的每一次输出。

5.2.5 规范驱动编程的运作机制

■ **Skill的工作流程**

Skill的工作流程如图所示。



5.2.5 规范驱动编程的运作机制

■ Skill与传统提示词的区别

Skill与传统提示词在多个维度存在显著差异，对比情况如表所示。

维度	传统提示词	Skill
复用性	每次重新编写	一次定义，多次使用
一致性	依赖个人写作水平	模板化，输出稳定
约束力	建议性（AI可能忽略）	强制性（通过解析器强制执行）
可组合	独立使用	多个技能可串联成工作流
版本管理	无	可纳入版本控制

Part 03

规范驱动编程工具选型

5.3 规范驱动编程工具选型

工具选型概述

Spec-Kit

OpenSpec

在TRAE中安装与配置OpenSpec

工具选型建议

5.3.1 工具选型概述

在选择规范驱动编程工具之前，需要先明确几个评估维度：

成熟度与社区支持

工具是否经过大量项目验证，社区是否活跃，遇到问题时能否获得足够的支持。

与现有工作流的兼容性

工具是否能够与你当前使用的AI编程编辑器（如 TRAE）良好集成，是否需要显著改变现有的开发流程。

学习曲线与采用成本

团队需要投入多少时间学习工具的使用和规范写作方法，才能达到有效的使用状态。

灵活性与约束力

工具是提供一个轻量级的框架让你自行填充，还是提供严格的流程让你必须遵循。

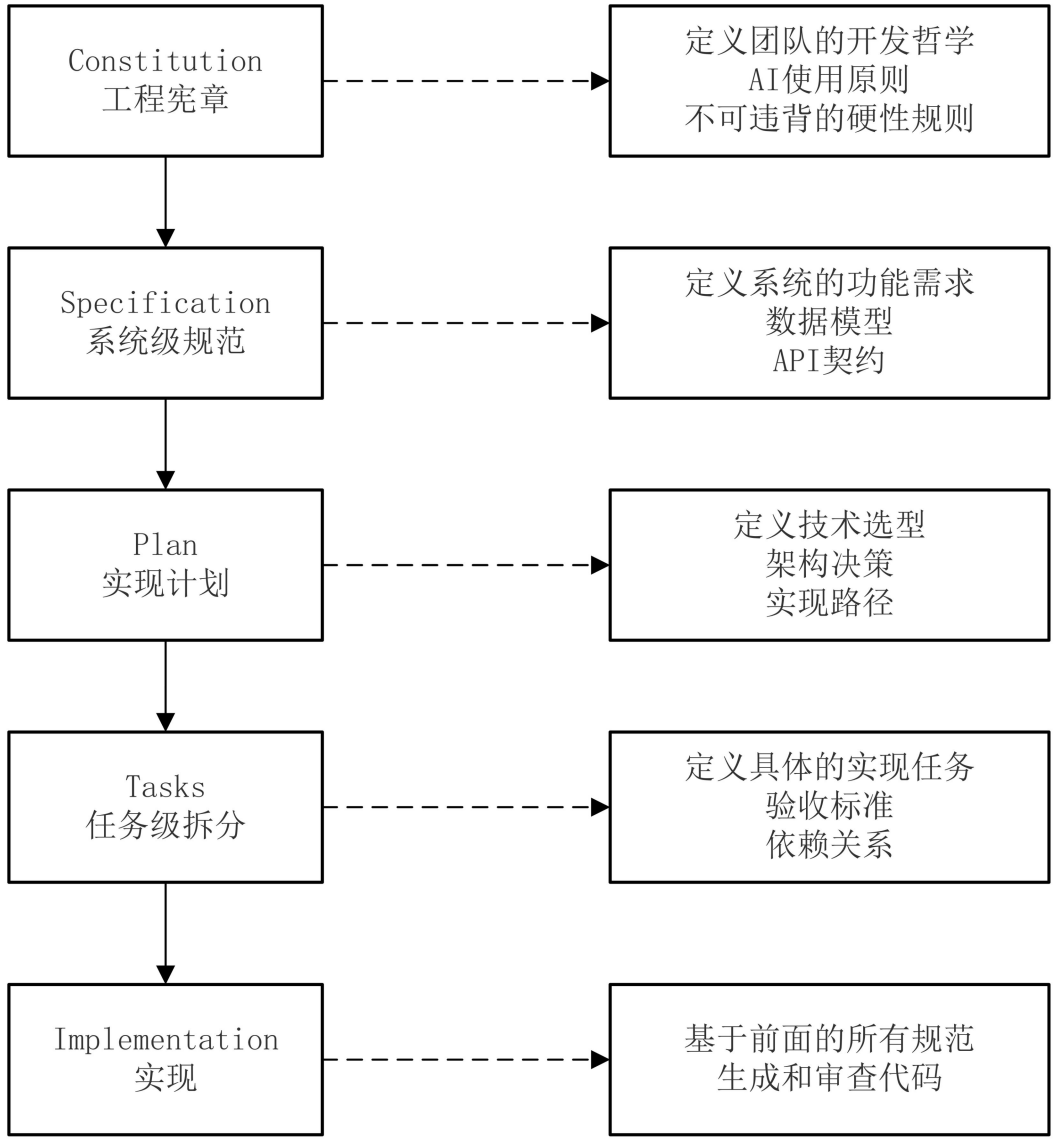
5.3.1 工具选型概述

下表列出了几个主要的规范驱动编程工具的对比

维度	Spec-Kit	OpenSpec	Amazon Bedrock Agent
维护方	GitHub官方	Fission-AI（开源社区）	AWS
GitHub星标数量	69.1k	23.7k	N/A（云服务）
技术栈	Python (uv)	TypeScript (npm)	Python/Boto3
规范组织	分散式 (多文件、多阶段)	统一式 (单文档为主)	API 内置 (云端管理)
规范严格度	高（五阶段门控）	中（四步循环）	低（内置框架）
与AI编辑器集成	需配置	原生支持TRAE等	仅支持AWS环境
学习曲线	陡峭	平缓	平缓
适用规模	大型团队、企业级	小团队、敏捷项目	云原生企业应用

5.3.2 Spec-Kit

Spec-Kit是GitHub官方推出的规范驱动开发框架，定位为企业级严格规范管理。它的核心特点是将规范过程分解为五个递进阶段，每个阶段有明确的产出物和门控检查。Spec-Kit的五层规范链路设计，从顶层工程宪章到底层代码实现，体现了严格的分层架构思维。即使目前无需使用如此重量级的框架，理解其分层思想也有助于读者更好地组织规范文档。五层规范链路是Spec-Kit的核心框架，如右图所示。



5.3.2 Spec-Kit

Constitution（工程宪章）是整个框架的顶层约束，定义团队在使用AI编程工具时必须遵守的硬性规则。例如：

```
# Constitution（工程宪章）

## AI使用原则

### 必须
- 所有AI生成代码必须经过代码审查才能合并
- 安全相关功能（认证、支付、数据加密）必须由人类工程师实现核心逻辑
- 规范文档必须与代码同步更新

### 禁止
- 禁止使用AI生成涉及用户隐私数据的处理逻辑
- 禁止使用AI修改数据库迁移文件
- 禁止在未创建规范文档的情况下直接让AI修改核心业务逻辑

## 代码质量标准

### AI生成代码的最低要求
- 所有函数必须包含中文docstring
- 所有公共API必须有单元测试覆盖
- 代码必须通过团队统一的lint检查

## 规范优先级
- Constitution > Specification > Plan > Tasks
- 下层规范不得与上层规范冲突
```

5.3.2 Spec-Kit

Specification（系统级规范）定义系统的功能需求和数据模型。与Constitution不同，Specification关注的是"系统做什么"，而非"团队如何工作"。例如：

```
# Specification（系统级规范）

## 功能范围

### 核心功能
- 用户注册与登录（邮箱 + 密码）
- 用户资料管理
- 订单创建与查询

### 排除功能（本章不涉及）
- 支付网关集成
- 物流跟踪
- 促销和优惠券
```

5.3.2 Spec-Kit

```
## 数据模型

核心数据模型包括用户和订单两个实体，字段定义如下：

**User实体字段定义**

| 字段 | 类型 | 约束 | 说明 |
|------|------|------|------|
| id | UUID | 主键 | 用户唯一标识 |
| email | VARCHAR(255) | 唯一，非空 | 邮箱地址 |
| hashed_password | VARCHAR(255) | 非空 | bcrypt 哈希密码 |
| created_at | TIMESTAMP | 非空，默认 NOW | 注册时间 |

### Order实体

**Order实体字段定义**

| 字段 | 类型 | 约束 | 说明 |
|------|------|------|------|
| id | UUID | 主键 | 订单唯一标识 |
| user_id | UUID | 外键，非空 | 下单用户 |
| total_amount | DECIMAL(10,2) | 非空 | 订单总金额 |
| status | ENUM | 默认 PENDING | 订单状态 |
| created_at | TIMESTAMP | 非空，默认 NOW | 创建时间 |
```

5.3.2 Spec-Kit

Spec-Kit的适用场景主要包括：

- **大型团队协作，需要明确的规范层级和门控检查。**
- **金融、医疗等对合规性要求高的行业。**
- **需要严格架构约束的复杂系统。**
- **已有成熟工程实践的团队，需要将AI编程纳入现有质量保障体系。**

5.3.2 Spec-Kit

Spec-Kit的局限性主要包括：

01 学习曲线较陡，团队需要投入较多时间学习五阶段框架。

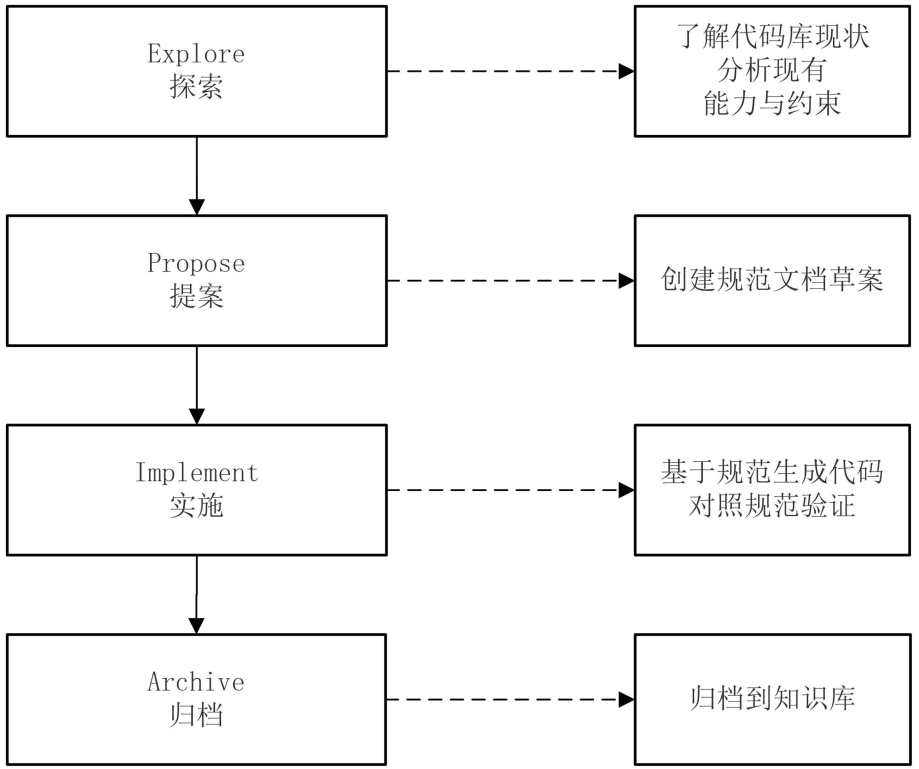
02 规范文档数量多，维护成本较高。

03 对于快速迭代的中小型项目，可能显得过于笨重。

5.3.3 OpenSpec

OpenSpec是Fission-AI推出的轻量级规范驱动框架，定位为快速迭代的灵活规范管理。相比Spec-Kit的严格多阶段流程，OpenSpec更强调简洁和快速上手。

OpenSpec是本书推荐的首选入门工具，它的四步工作流简洁直观，学习成本低，与TRAE等AI编辑器的集成原生顺畅。对于个人开发者和小团队而言，OpenSpec是实践规范驱动编程的合适起点。四步工作流是OpenSpec的核心，如图所示。



5.3.3 OpenSpec

OpenSpec的四步工作流具体如下：

Explore（探索）	Propose（提案）	Implement（实施）	Archive（归档）
全面调研梳理代码库整体现状，拆解项目现有技术能力、功能架构与业务逻辑，排查各类开发约束、潜在问题与技术边界，为后续开发工作筑牢前置基础	当有新功能需求时，在openspec/changes/目录下创建新的规范文档草案。这个阶段是快速且非正式的——先用简洁的语言描述要做什么，不需要追求完美。	规范确认后，AI基于规范生成代码。每次代码变更都与规范进行对比验证，确保一致性。这个阶段强调"规范即真理"——代码必须符合规范，任何偏差都需要被记录和修正	功能完成并通过验收后，将规范文档和相关知识归档到openspec/archive/目录。归档不是丢弃，而是将知识整理成可检索的形式，供日后参考

5.3.3 OpenSpec

OpenSpec的项目结构非常简洁，具体如下：

```
your-project/
├── openspec/
│   ├── changes/                    # 活跃变更目录（正在进行的工作）
│   │   ├── user-auth/              # 用户认证功能规范
│   │   │   └── spec.md             # 规范文档
│   │   └── todo-list/              # 待办列表功能规范
│   │       ├── spec.md             # 规范文档
│   │       ├── plan.md             # 技术方案（可选）
│   │       └── tasks.md            # 任务清单（可选）
│   ├── archive/                    # 归档知识库（已完成的工作）
│   │   └── v1.0-features/
│   │       └── spec.md             # 历史规范
│   └── config.yaml                 # 项目配置（可选）
└── AGENTS.md                        # AI 上下文配置
```

5.3.3 OpenSpec

OpenSpec的规范文档格式强调实用性和可操作性，具体格式如下：

```
# 规范：[功能名称]

## 1. 概述
[功能要解决什么问题，简要描述]

## 2. 功能范围
### 2.1 必须实现
- [功能点 1]
- [功能点 2]

### 2.2 不包含
- [明确排除的功能]

## 3. 数据模型
### 3.1 [实体名称]
| 字段 | 类型 | 约束 | 说明 |
|------|------|------|------|
| ... | ... | ... | ... |
```

5.3.3 OpenSpec

```
## 4. API规范
### 4.1 [端点名称]
**方法**: GET/POST/PUT/DELETE
**路径**: /api/xxx

**请求体**（如有）：
{ ... }

**响应**:
- 成功（200/201）：`{ ... }`
- 错误（4xx/5xx）：`{ "detail": "..." }`

## 5. 验收标准
- [ ] [可验证的标准 1]
- [ ] [可验证的标准 2]

## 6. 技术约束
- [约束 1]
- [约束 2]

## 7. 边界条件
- [边界情况 1]
- [边界情况 2]
```

5.3.3 OpenSpec

OpenSpec的适用场景主要包括：

01

存量项目引入规范驱动编程（学习成本低，可渐进式采用）。

02

初创公司或小团队（灵活、快速，不需要严格的阶段门控）。

03

个人开发者建立规范写作习惯。

04

快速迭代的敏捷项目。

5.3.3 OpenSpec

OpenSpec的局限性主要包括：

局限性

1. 规范的质量很大程度上取决于团队成员对"好规范"的理解。
2. 没有强制性的阶段检查，可能出现"规范写了但没认真执行"的情况。
3. 对于需要严格合规管理的行业，可能不够正式。

5.3.4 在TRAE中安装与配置OpenSpec

■ 前置要求

使用OpenSpec之前，需要确认本地已安装Node.js 20.19.0或更高版本。读者可以在终端中执行以下命令检查版本：

```
node --version
```

如果没有安装，则可以从https://nodejs.org/zh-cn/download下载Node.js进行安装。假设读者根据Node.js的指引安装以后，无法在Git Bash中使用，则一般是环境变量的问题，可以在“~/.bashrc”文件中添加如下环境变量：

```
export PATH="$PATH:/c/Program Files/nodejs/"
```

5.3.4 在TRAE中安装与配置OpenSpec

■ **第一步：安装OpenSpec**

在TRAE界面的底部面板的“终端”面板中，如图所示，点击箭头所指向的下拉菜单按钮，在弹出的菜单中选择“Git Bash”，打开Git Bash终端。

5.3.4 在TRAE中安装与配置OpenSpec

■ 第一步：安装OpenSpec

OpenSpec以npm全局包的形式发布（npm是Node.js的包管理工具，Node.js安装成功以后，就自带了npm），可以在Git Bash终端中，在当前项目的根目录下执行以下命令完成OpenSpec的安装：

```
npm install -g @fission-ai/openspec@latest
```

安装完成后，执行以下命令验证安装是否成功：

```
openspec --version
```

若终端输出版本号（例如1.4.1），则安装成功。

如果安装完OpenSpec以后无法使用，大概率是环境配置的问题，可以在“~/.bashrc”文件中添加如下命令路径（需要将路径替换为读者计算机上的npm的真实路径），然后重新打开Git Bash：

```
export PATH="$PATH:/c/Users/linziyu/AppData/Roaming/npm"
```

5.3.4 在TRAE中安装与配置OpenSpec

■ **第一步：安装OpenSpec**

如果读者自己无法找到npm全局安装路径，则可以直接询问TRAE，让TRAE帮你找到真实路径，具体方法是，在TRAE的智能体交互界面的对话框中输入如下提示词并发送：

```
帮我找到我这台机器的npm全局安装包的路径
```

5.3.4 在TRAE中安装与配置OpenSpec

■ 第二步：初始化项目

在TRAE的终端中，进入项目根目录，执行如下初始化命令：

```
openspec init
```

“openspec init”默认以交互模式运行。首先，会出现提示“Press Enter to select tools...”，需要按回车键进入工具选择界面。然后，会出现提示让你选择工具（如下页图所示），也就是你要把OpenSec集成到哪个工具中，因为我们需要在TRAE中使用OpenSpec，所以，需要选择TRAE，具体方法是，使用键盘上的下箭头，把光标移动到“TRAE”这个选项上，然后，按空格键选中TRAE工具，这时，TRAE这个选项后面会出现“(Selected)”，如果再次按空格键，就会取消选中状态，这里我们选中TRAE工具，最后，按回车键确认，就完成了OpenSpec的初始化工作。

5.3.4 在TRAE中安装与配置OpenSpec

- **第二步：初始化项目**

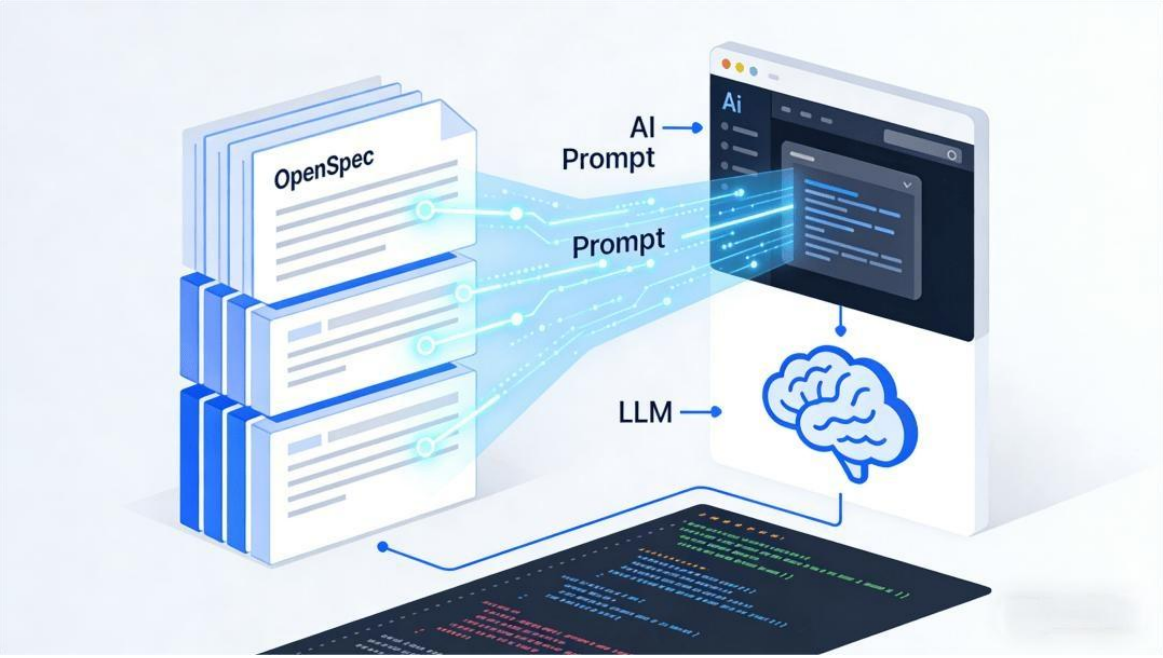
```
[redacted] % openspec init
? Select tools to set up (28 available)
  Selected: Trae
  Search: [type to filter]
  ↑↓ navigate • Space toggle • Backspace remove • Enter confirm
    ○ Factory Droid
    ○ Gemini CLI
    ○ GitHub Copilot
    ○ iFlow
    ○ Junie
    ○ Kilo Code
    ○ Kiro
    ○ OpenCode
    ○ Pi
    ○ Qoder
    ○ Lingma
    ○ Qwen Code
    ○ RooCode
  › ◉ Trae (selected)
    ○ Windsurf
  (2/2)
```

5.3.4 在TRAE中安装与配置OpenSpec

■ **第二步：初始化项目**

需要补充说明的是，如果需要在脚本或CI（Continuous Integration：持续集成）环境中以非交互模式运行（本书不使用非交互模式），可使用--tools参数，具体命令如下：

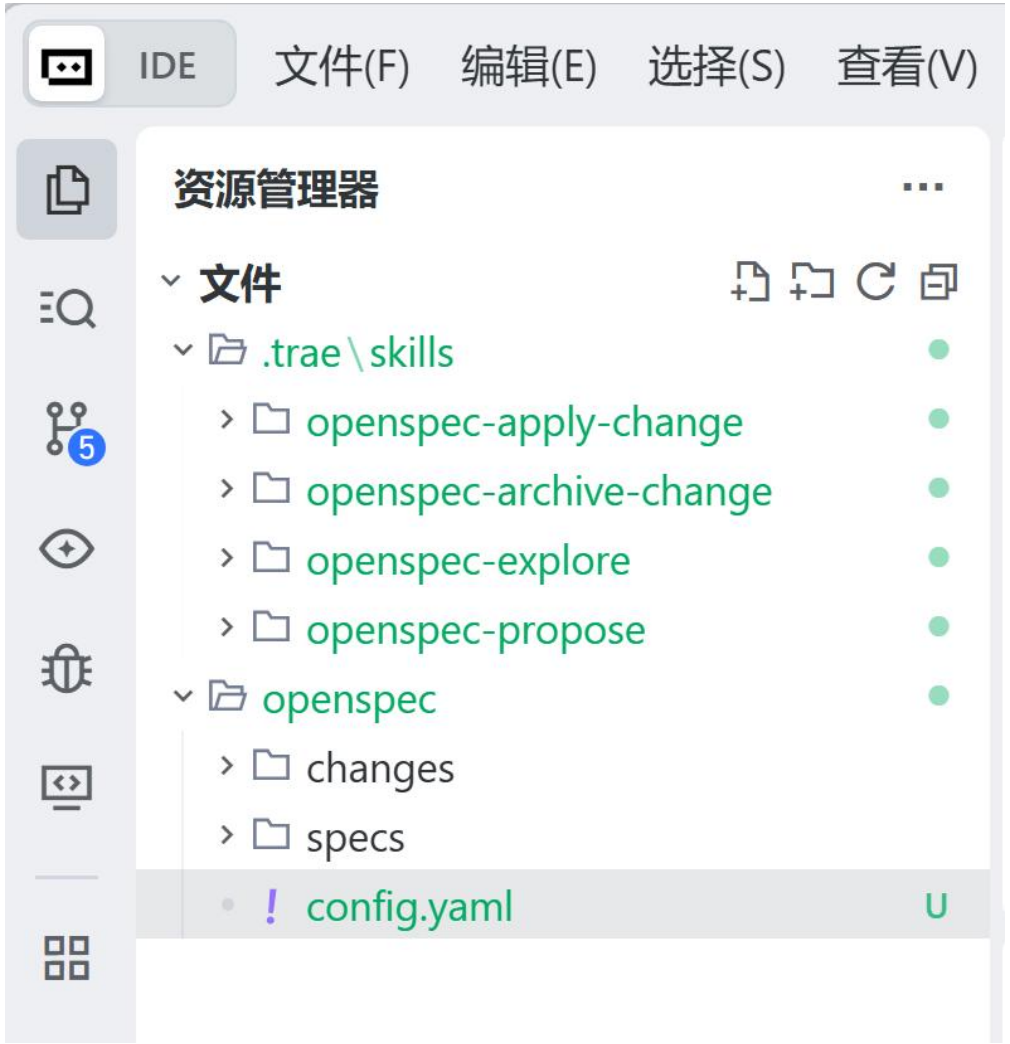
```
# 仅为 TRAE 配置
openspec init --tools trae
```



5.3.4 在TRAE中安装与配置OpenSpec

■ **第二步：初始化项目**

完成了OpenSpec的初始化工作以后，初始化程序会生成如图所示的目录结构。



5.3.4 在TRAE中安装与配置OpenSpec

■ **第3步：理解核心目录结构**

初始化完成后，项目根目录下的openspec目录即成为项目规范的统一管理中心。下表说明了各文件或目录的作用。

文件/目录	用途	说明
config.yaml	项目配置文件	声明技术栈、架构约束、规范写作规则，内容会被注入每次 AI 请求。备注：该文件和第4章介绍的项目说明书（AGENTS.md）部分重复，一般建议将项目的背景、技术栈、架构约束声明在项目说明书中，因为项目说明书更加通用，应用场景更加广泛
changes/	活跃变更目录	存放正在开发的功能规范，每个功能一个子目录
specs/	主规范目录	存放已完成并归档的规范，是项目的"事实来源"
.TRAE/skills/	AI Skills目录	AI自动检测并加载，驱动规范、驱动工作流

5.3.4 在TRAE中安装与配置OpenSpec

■ 第4步：配置config.yaml

config.yaml是项目配置文件，其context字段的内容会被自动注入到每一次AI规划请求中，确保AI始终了解项目的技术背景。下面是一个典型的配置示例：

```
schema: spec-driven

context: |
  Tech stack: Python 3.12, FastAPI, SQLite, SQLAlchemy
  Testing: pytest
  Architecture: Three-layer (routers → services → repositories)
  We use conventional commits

rules:
  proposal:
    - Always include "## Why" and "## What Changes" sections
  specs:
    - Use Given/When/Then format for scenarios
  tasks:
    - Break tasks into max 2-hour chunks
```

5.3.4 在TRAE中安装与配置OpenSpec

■ 第5步：理解规范文档格式

OpenSpec对规范文档格式有严格要求，了解这些格式是正确编写规范的前提。

每个变更目录下的proposal.md必须包含## Why和## What Changes两个段落，否则openspec validate会报错，示例如下：

```
# Proposal: 猜数字游戏
## Why
### Background
[背景描述]
### Problem Statement
[问题陈述]
## What Changes
### New Capabilities
- 游戏核心逻辑
- 数据存储
## Capabilities
### New Capabilities
- `game-core`: 游戏核心逻辑
- `game-storage`: localStorage 存储管理
```

5.3.4 在TRAE中安装与配置OpenSpec

■ 第5步：理解规范文档格式

能力规范文件“specs/[capability]/spec.md”，必须使用“变更类型+Requirement+Scenario”格式，示例如下：

```
## ADDED Requirements

### Requirement: 随机数生成

系统 SHALL 在每局游戏开始时生成 1~100 之间的随机整数。

**Priority**: P0 (Critical)

**Rationale**: 随机数是猜数字游戏的核心。

#### Scenario: 游戏开始时生成目标数字

Given 玩家点击"开始游戏"
When 系统初始化新一局游戏
Then 系统生成一个 1~100 之间的随机整数作为目标数字
And 玩家不可见该数字
```

5.3.4 在TRAE中安装与配置OpenSpec

■ **第5步：理解规范文档格式**

格式要点如表所示。

元素	格式	示例
变更类型	## ADDED/MODIFIED/REMOVED Requirements	## ADDED Requirements
需求标题	### Requirement: <标题>	### Requirement: 输入验证
场景标题	#### Scenario: <标题>	#### Scenario: 输入超出范围
场景内容	Gherkin格式	Given/When/Then

5.3.4 在TRAE中安装与配置OpenSpec

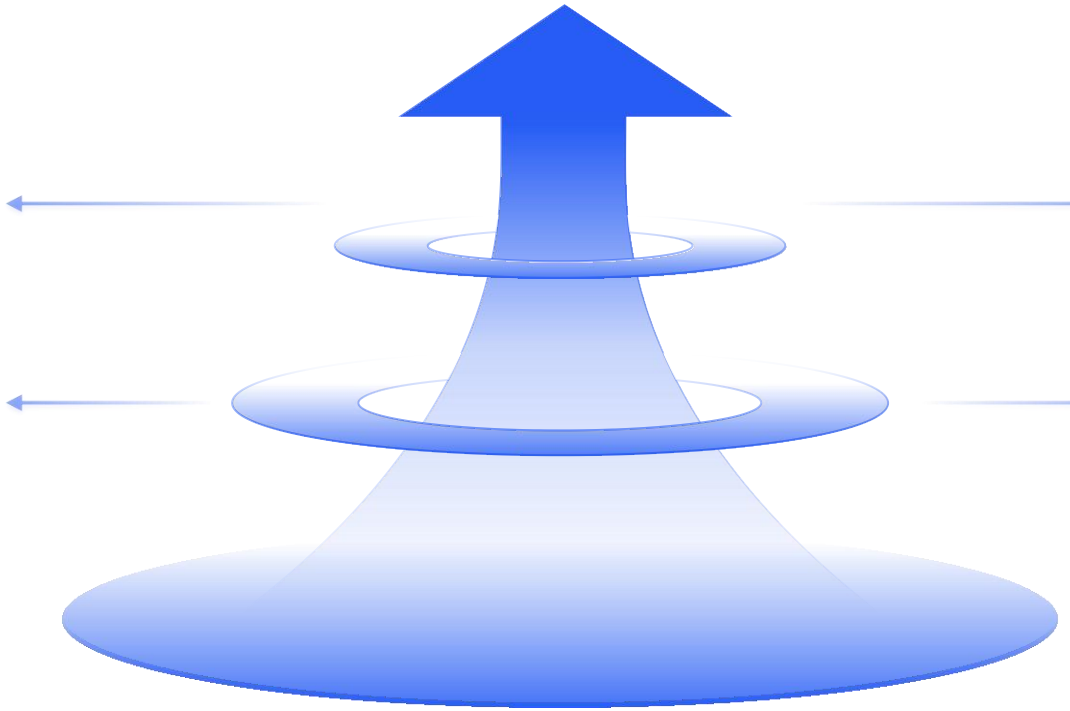
Gherkin是一种结构化自然语言格式，用于编写可被AI解析、可直接转化为测试用例的场景描述，核心是通过“Given（给定）、When（当）、Then（那么）”三段式清晰定义前置条件、触发动作和预期结果，确保验收标准可验证、无歧义。Gherkin格式不依赖编程语言，是规范文档场景描述的标准写法，其基础结构如下：

Given

描述场景执行前的初始状态、前置条件或数据准备，如“Given用户已登录系统”；

When

描述触发场景的具体动作或用户操作，如“When用户提交猜数字请求”；



Then

描述动作执行后预期的结果、系统反馈或状态变化，如“Then系统返回‘太低’的提示”；

And

用于补充多个Given、When或Then条件，简化重复表述，如“And剩余猜测次数更新为6”。

采用Gherkin格式编写场景，既能让人类开发者快速理解业务逻辑，又能让AI 精准解析验收标准，避免模糊描述导致的语义漂移，是OpenSpec规范文档场景部分的强制要求。

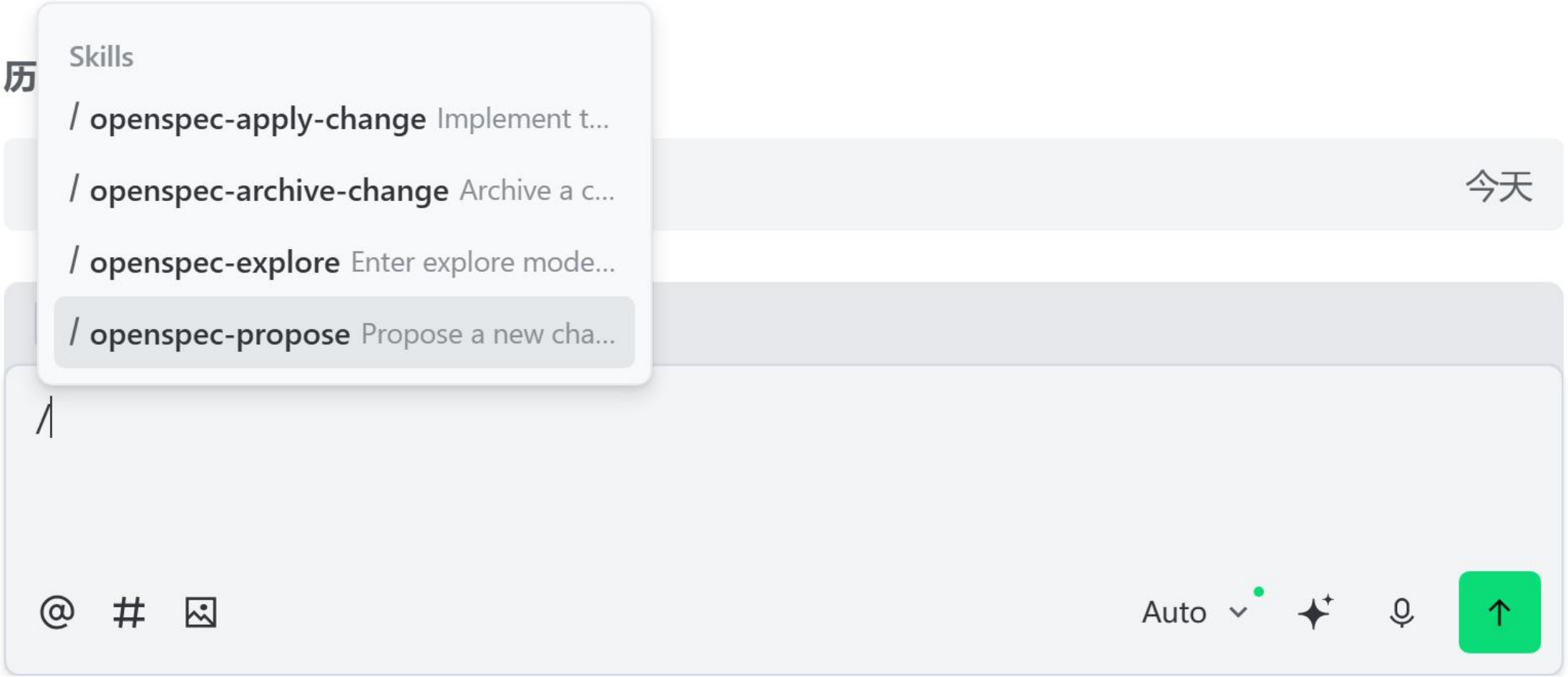
5.3.4 在TRAE中安装与配置OpenSpec

下表给出了OpenSpec的常用命令说明，一般情况下，我们都不需要直接使用这些命令，TRAE会自动调用这里的命令完成工作，但是，了解这些命令，有助于故障诊断和理解OpenSpec的原理。

命令	说明
openspec init --tools TRAE	初始化项目并配置 TRAE
openspec validate <name>	验证变更规范文档格式
openspec list --changes	列出所有活跃变更
openspec status --change <name>	查看变更文档完成进度
openspec archive <name>	归档已完成的变更
openspec --version	查看版本号

5.3.4 在TRAE中安装与配置OpenSpec

在TRAE中，更推荐使用斜杠命令驱动工作流，比直接调用CLI更加便捷，如图所示。



5.3.5 工具选型建议

基于前面的分析，下表给出了针对不同场景的规范驱动编程工具选型建议

团队规模	项目类型	推荐工具	理由
个人开发者	探索性项目、个人工具	OpenSpec	学习成本低，灵活快捷
2～5 人小团队	快速迭代产品	OpenSpec	轻量级，无需严格流程，适合敏捷开发
5～15 人中型团队	生产级应用	OpenSpec + 自定义门控	在 OpenSpec 基础上增加必要的检查点
15 人以上大型团队	企业级系统	Spec-Kit	严格的多阶段流程适合大规模协作
任何规模	AWS 云原生项目	Amazon Bedrock Agent	与 AWS 服务天然集成，适合云应用

Part 04

OpenSpec实践

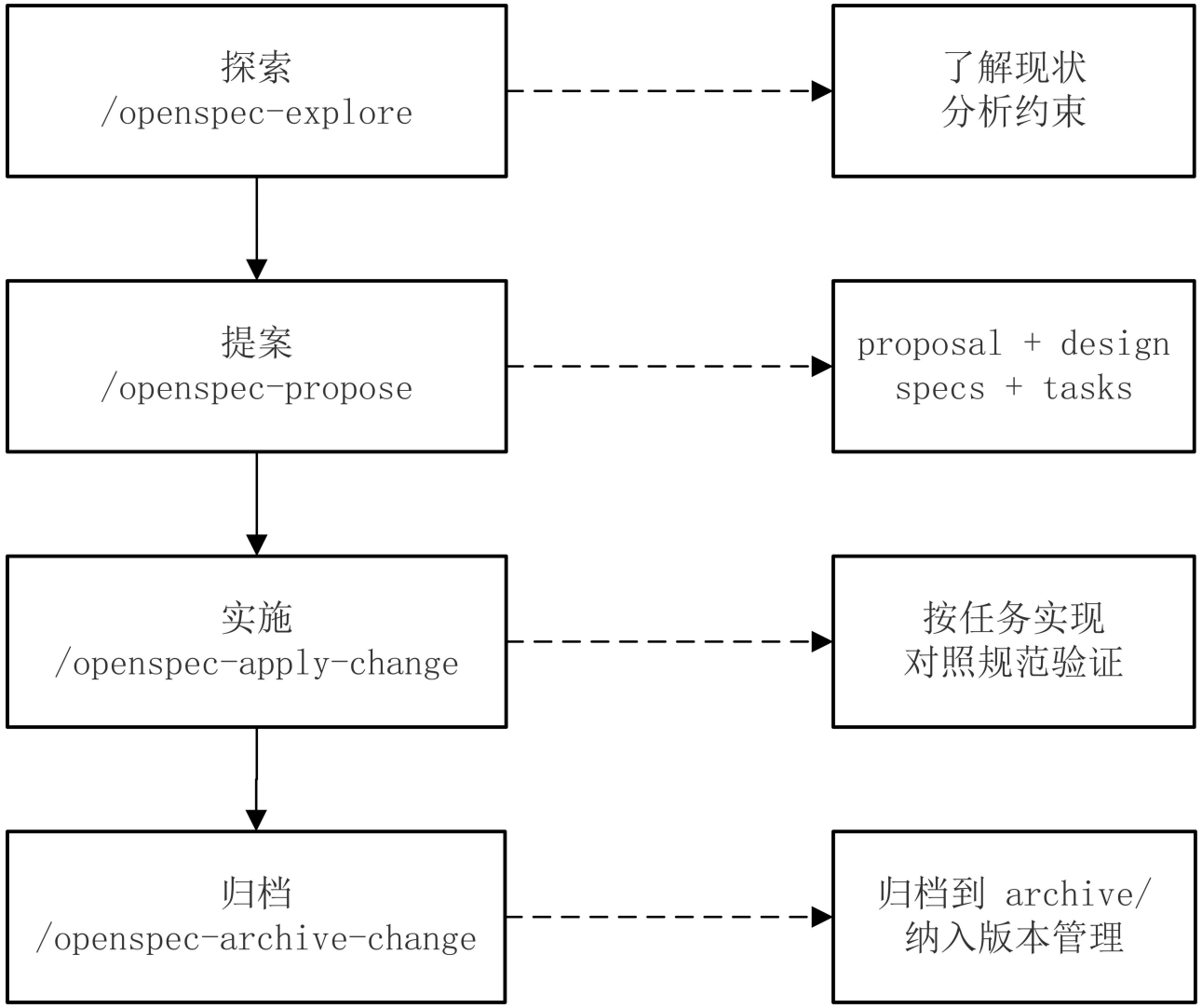
5.4 OpenSpec实践

OpenSpec是Fission-AI推出的轻量级规范驱动编程框架，也是本书推荐的首选入门工具。它的核心工作流分为四个阶段，每个阶段对应一条OpenSpec斜杠命令，具体如表所示。

阶段	命令	作用
探索	/openspec-explore	了解代码库现状，分析现有能力与约束
提案	/openspec-propose	创建结构化规范文档（proposal + design + specs + tasks）
实施	/openspec-apply-change	按照 tasks.md 逐步实现代码，严格对照规范验证
归档	/openspec-archive-change	将已完成的变更归档，形成可追溯的知识库

5.4 OpenSpec实践

OpenSpec的整体流程如图所示。



5.4 OpenSpec实践

前置准备

第一步：探索（/openspec-explore）

第二步：提案（/openspec-propose）

第三步：实施（/openspec-apply-change）

第四步：归档（/openspec-archive-change）

功能迭代：增加暗黑模式

其他操作

5.4.1 前置准备

在开始实践之前，请确认已按照5.3.5节完成OpenSpec的安装。在TRAE中新建一个项目，项目名称为guess-number-game，然后，在guess-number-game项目的根目录下，执行命令“openspec init”，完成OpenSpec的初始化。

OpenSpec的初始化工作完成以后，会在项目的根目录下出现openspec子目录。打开openspec目录下的config.yaml文件进行编辑，清空里面的内容，然后，写入猜数字游戏项目的如下技术栈信息：

```
schema: spec-driven

context: |
  Tech stack: HTML5, CSS3, Vanilla JavaScript
  Storage: localStorage
  No backend, pure frontend application

rules:
  proposal:
    - Always include "## Why" and "## What Changes" sections
  specs:
    - Use Given/When/Then format for scenarios
```

5.4.1 前置准备

需要说明的是，config.yaml的内容应根据实际项目调整，以便OpenSpec为AI提供准确的上下文。例如，当前项目是纯前端项目，因此，技术栈是“HTML5+CSS3+Vanilla JavaScript”，如果是Python后台项目，则技术栈应声明Python的版本和使用的框架，如“Python 3.12+FastAPI+SQLite”。

配置完成后，在后续步骤中可以选择以下两种方式驱动工作流：

方式一 OpenSpec斜杠命令 （推荐）	在TRAE的智能体交互界面的对话框中输入 /openspec-explore、/openspec-propose 、 /openspec-apply-change 、 /openspec-archive-change等命令，自动触发对应的技能工作流。
方式二 手动输入指令	在AI对话中手动输入"请使用OpenSpec的xxx Skill"指令，灵活控制每一步的输入。

5.4.2 第一步：探索（/openspec-explore）

探索阶段的目的是让AI深入了解代码库现状，在不写任何代码的情况下，理清现有功能、技术约束和已有规范，为后续的提案阶段奠定基础。

是否需要探索，可参考下表的判断标准。

场景	是否需要探索	说明
全新项目，从零开始	通常可跳过	代码库为空，没有需要了解的现状
在已有功能基础上迭代	必须进行	需要了解现有实现，避免冲突
接手他人代码进行修改	强烈建议	理解已有约定和技术选型
排查Bug或优化性能	建议进行	需要先定位问题所在

对于猜数字游戏案例而言，由于这是一个全新项目，代码库为空，这里可以跳过探索阶段，直接进入提案阶段。

5.4.3 第二步：提案（/openspec-propose）

提案阶段是OpenSpec工作流的核心。执行/openspec-propose命令后，输入你的项目需求，AI就会生成完整的规范文档集合，具体包括：

design.md

描述怎么做（技术方案、架构决策）。

specs/<能力名>/spec.md：

每个能力的详细规范（数据模型、场景、验收标准）。

tasks.md

将实现过程拆解为可执行的任务列表。

proposal.md

描述为什么做（背景、目标）和做什么（功能范围、影响的能力）。

这些文档共同构成了"规范即真理"的事实来源，后续的代码实现必须严格对照这些文档。

5.4.3 第二步：提案（/openspec-propose）

现在我们为一个网页应用实现"猜数字游戏"功能。在TRAE的Agent模式中，在智能体交互界面的对话框中输入以下指令，观察AI的输出：

```
/openspec-propose请为"猜数字游戏"创建规范文档。

需求描述：
实现一个猜数字游戏网页应用。系统在1~100之间随机选择一个整数作为目标数字。
玩家有7次机会猜测，每次猜测后系统给出反馈（太高、太低或正确）。
游戏结束后自动保存记录到 localStorage，支持查看历史记录和统计数据。
```

5.4.3 第二步：提案（/openspec-propose）

AI执行/openspec-propose后，会在openspec/changes目录下创建一个名为“guess-number-game”的子目录，并在这个子目录下创建多个文件。此外，需要特别注意的是，这个子目录名称就是变更的任务ID，在后续的流程中，将通过这个任务ID来推进流程。

执行完/openspec-propose以后，AI创建的目录和文件如下：

```
openspec/changes/guess-number-game/
├── proposal.md              # 提议文档
├── design.md                # 技术设计文档
├── tasks.md                 # 任务清单
└── specs/
    ├── game-core/           # 游戏核心逻辑规格
    ├── game-state/          # 游戏状态管理规格
    ├── score-storage/       # 存储功能规格
    ├── history-view/        # 历史记录视图规格
    └── statistics/          # 统计数据规格
```

5.4.3 第二步：提案（/openspec-propose）

AI还会简单列出各个文件的作用，便于用户审查。例如，在这个项目中，AI列出的审查信息如下：

```
## 文档内容概览

1. proposal.md

- 阐述了游戏的动机和目标
- 定义了5个核心能力：game-core、game-state、score-storage、history-view、statistics

2. design.md

- 技术栈选择：纯 HTML/CSS/JavaScript + TailwindCSS + Lucide Icons
- 文件结构设计和游戏状态管理方案
- localStorage 数据结构设计
- 风险评估和缓解策略
```

5.4.3 第二步：提案（/openspec-propose）

3. specs/

- game-core：随机数生成、猜测验证、反馈提示、输入验证
- game-state：游戏状态追踪、持久化、重置
- score-storage：记录保存、历史读取、清除功能
- history-view：记录展示、筛选、详情查看
- statistics：胜率、平均猜测次数、最佳游戏、总数统计

4. tasks.md

- 包含9个任务组，共36个可追踪任务
- 从项目结构创建到测试验证的完整流程

5.4.3 第二步：提案（/openspec-propose）

AI生成的文档有严格的顺序，总是先生成proposal.md，然后是design.md，接下来是各个规范文档（specs目录下的各个文件），最后是tasks.md文档。

例如，在当前这个示例中，AI输出的proposal.md如下：

```
## Why

猜数字游戏是一个经典的互动游戏，通过网页应用形式实现可以提供便捷的娱乐体验。该游戏能够锻炼玩家的逻辑推理能力，同时记录功能可以让玩家追踪自己的游戏历史和表现。

## What Changes

- 创建猜数字游戏网页应用，支持1~100范围内的随机数字生成
- 实现7次猜测机会限制，每次猜测后给出"太高"、"太低"或"正确"的反馈
- 游戏结束后自动保存记录到 localStorage
- 提供历史记录查看功能
- 提供游戏统计数据展示（胜率、平均猜测次数等）
```

5.4.3 第二步：提案（/openspec-propose）

```
## Capabilities

### New Capabilities
- `game-core`: 游戏核心逻辑，包括随机数生成、猜测验证和反馈提示
- `game-state`: 游戏状态管理，包括剩余猜测次数、当前目标数字等
- `score-storage`: localStorage 存储管理，自动保存游戏记录
- `history-view`: 历史记录查看功能，展示过往游戏记录
- `statistics`: 统计数据计算和展示

### Modified Capabilities
<!-- 无修改现有能力 -->

## Impact

- 新增 HTML/CSS/JS 文件实现游戏界面和逻辑
- 使用 localStorage 进行数据持久化
- 无需后端依赖，纯前端实现
```

5.4.3 第二步：提案（/openspec-propose）

对比原始需求描述和AI输出的规范文档，可以看到提案阶段带来的三个关键变化：

模糊变清晰

细化了用户的各项需求，并补充了一些细节，便于用户修改和后续流程实现。

隐性变显性

原始需求没有提到的"重复猜测检查"、"太高/太低反馈格式"、"游戏结束后的目标数字揭示"等细节，在规范文档中被明确列出。

不可验证变可验证

原始需求中的"游戏记录"变成了具体的数据结构设计（历史记录、统计数据），每项功能都有明确的验收标准。

5.4.3 第二步：提案（/openspec-propose）

AI在生成proposal.md文档以后，接下来输出和实现细节相关的技术方案文档，即design.md（只截取部分内容）

```
## Context

本项目旨在实现一个猜数字游戏网页应用。根据 proposal，游戏核心功能包括：
- 在1~100范围内随机生成目标数字
- 玩家有7次猜测机会
- 每次猜测后给出"太高"、"太低"或"正确"的反馈
- 自动保存游戏记录到 localStorage
- 支持查看历史记录和统计数据

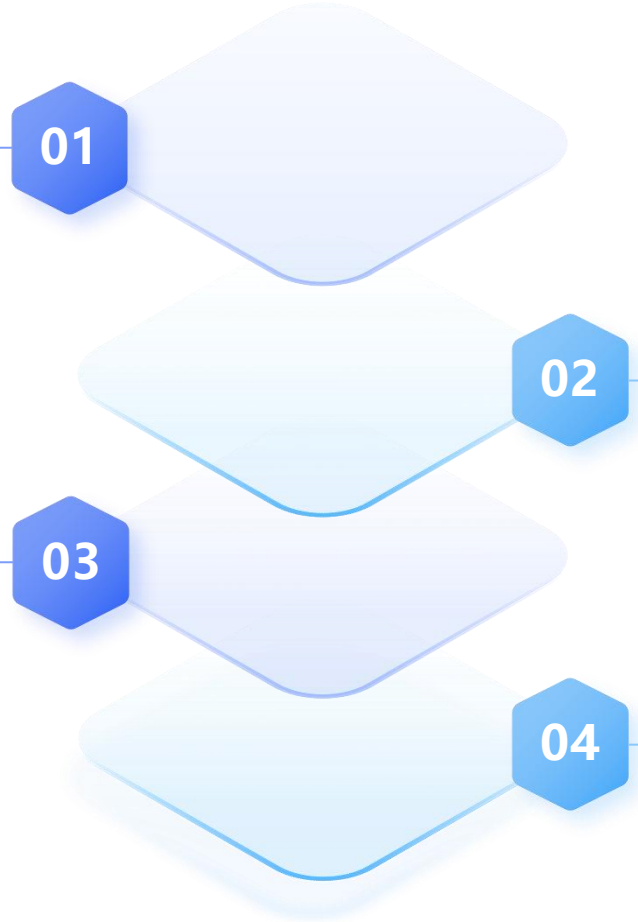
当前状态：项目为空项目，需要从零开始构建前端应用。

## Goals / Non-Goals

**Goals:**
- 实现完整的猜数字游戏逻辑
- 提供良好的用户界面体验
- 实现 localStorage 数据持久化
- 展示历史记录和统计数据
```

5.4.3 第二步：提案（/openspec-propose）

design.md 将 proposal.md 中的"做什么"转化为"怎么做"，它确保了以下事项：



01 技术选型有依据

每个技术选择都有明确的理由，而不是"因为AI默认用了这个"。

02 架构设计有规划

在写代码之前就明确了目录结构、模块划分和数据流。

03 关键决策有记录

游戏核心算法（二分查找策略）、localStorage 存储策略、响应式设计方案等关键决策被显式记录，供后续开发者参考。

04 验收标准可追溯

每条验收标准都有对应的实现方式和验证方法。

5.4.3 第二步：提案（/openspec-propose）

接下来AI将会生成规范（specs），规范会有多个文件，位于specs目录下，使用的是5.3.5小节介绍的Gherkin 格式。规范详细定义了边界条件、异常处理，以及用户交互的细节。比如，game-core目录下的规范文件内容如下所示（注意，你机器里的内容可能与下面内容不同，AI每次执行以后生成的内容也不会相同）：

```
# game-core Specification

## Purpose
TBD - created by archiving change guess-number-game. Update Purpose after archive.
## Requirements
### Requirement: Generate random target number
The system SHALL generate a random integer between 1 and 100 (inclusive) as the target number when starting a new game.

#### Scenario: Generate target on new game
- **WHEN** user starts a new game
- **THEN** system generates a random integer between 1 and 100
- **AND** the generated number is stored as the target number
```

5.4.3 第二步：提案（/openspec-propose）

提案阶段的最后一步，AI会根据技术方案生成任务列表tasks.md，具体内容如下：（只截图部分内容）

```
## 1. 创建项目基础文件

- [ ] 1.1 创建 index.html 主页面
- [ ] 1.2 创建 styles.css 样式文件
- [ ] 1.3 创建 script.js 游戏逻辑文件

## 2. 实现游戏核心逻辑

- [ ] 2.1 实现随机数生成函数 (1-100)
- [ ] 2.2 实现猜测验证函数 (范围检查、格式检查、重复检查)
- [ ] 2.3 实现猜测比较逻辑 (太高/太低/正确)
- [ ] 2.4 实现游戏状态管理 (目标数字、猜测次数、已猜数字列表)
```

5.4.3 第二步：提案（/openspec-propose）

可以看到，我们只是使用/openspec-propose提出了一个需求，AI就为我们创建了这么多文件。这些文件共同构成了"猜数字游戏"功能的完整事实来源。后续代码必须严格对照这些文档，这正是规范驱动编程的铁律二"规范即真理"的体现。

对于AI自动生成的每个specs文件（比如design.md、proposal.md、tasks.md），在每个文件中，都要点击“保留 Ctrl+Enter”按钮，接受对文件内容的修改，并保存该文件。

5.4.4 第三步：实施（/openspec-apply-change）

提案与规范确认无误后，即可进入实施阶段，开始编写代码。

在TRAE中选择“Agent模式”，在智能体交互界面的对话框中输入如下实施命令，即可启动编码流程：

```
/openspec-apply-change guess-number-game
```

若当前仅有一个活跃变更，/openspec-apply-change命令也可以不用带上guess-number-game，可直接简写为：

```
/openspec-apply-change
```

5.4.4 第三步：实施（/openspec-apply-change）

执行后，AI将自动加载当前变更目录（如openspec/changes/2026-06-08-guess-number-game/）下的所有规范文档，包括 proposal.md、design.md、specs/ 及 tasks.md，并严格按照tasks.md中的任务顺序，逐项生成代码。每完成一项任务，AI会自动将任务状态从未完成标记为完成。

和/openspec-propose阶段一样，AI执行完/openspec-apply-change以后，也会生成概要信息，告诉用户它做了什么，生成了哪些文件。在本例中，AI创建了下面这些文件：

```
├── index.html              # 主页面
├── css/
│   └── styles.css          # 样式文件
├── js/
│   ├── game.js             # 游戏核心逻辑
│   ├── storage.js          # 存储管理
│   └── ui.js               # UI 交互逻辑
└── openspec/changes/guess-number-game/
    ├── proposal.md         # 提议文档
    ├── design.md           # 技术设计文档
    ├── tasks.md            # 任务清单（已全部完成）
    └── specs/              # 详细规格文档
```

5.4.4 第三步：实施（/openspec-apply-change）

在本例中，最重要的业务逻辑都位于game.js中。代码生成后，需要逐条对照规范文档中的验收标准进行验证。下表列出了针对核心验收标准的测试验证要点。

验收标准	验证方法
随机数生成	多次新游戏，确认目标数字在 1~100 之间且随机
输入验证	测试 0、101、非数字、空输入、重复数字
反馈显示	猜大、猜小、猜对分别验证提示文字
次数限制	连续猜错7次后检查游戏是否结束
历史记录	刷新页面后检查记录是否仍在
统计数据	验证胜率计算公式是否正确

5.4.4 第三步：实施（/openspec-apply-change）

在Windows系统的文件管理器中，找到项目目录guess-number-game，进入这个目录后，可以看到一个网页文件index.html，用浏览器打开这个网页文件，就可以看到猜数字游戏的界面了。猜数字游戏完成后的界面效果如图所示（你生成的界面可能和这个不同）。

5.4.4 第三步：实施（/openspec-apply-change）

可以测试猜数字游戏是否可以正常工作，如果无法正常工作，比如，点击按钮没有反应，可以使用AI自动排查错误并修正代码，方法是，在TRAE的智能体交互界面的对话框中输入如下提示词并提交：

```
我用浏览器打开了index.html，生成了游戏界面，但是，我点击“猜”按钮没有任何反应，点击“新游戏”按钮也没有反应，帮我解决
```

TRAE就会自动帮你排查错误并修正代码，修正结束以后，需要你在被修改过的代码文件中点击“Ctrl+Enter”组合键接受修改操作，然后保存代码文件。再次用浏览器打开index.html，测试游戏是否可以正常工作。如果仍然无法正常工作，可以再次在TRAE的智能体交互界面的对话框中输入提示词“问题还是没有解决，你再处理一下”并提交。如果还是无法解决，就再次输入提示词“问题还是没有解决，你再处理一下”并提交，直到问题解决，游戏可以正常工作。

5.4.4 第三步：实施（/openspec-apply-change）

实施（/openspec-apply-change）阶段确保生成的代码与规范严格对齐，具体表现在以下几个方面：

游戏逻辑与规范一致

随机数范围、猜测次数限制、反馈规则与规范文档完全一致。

存储机制与规范一致

localStorage键名、数据结构与规范文档完全一致。

技术约束被遵守

纯前端实现，无后端依赖，响应式设计。

验收标准可验证

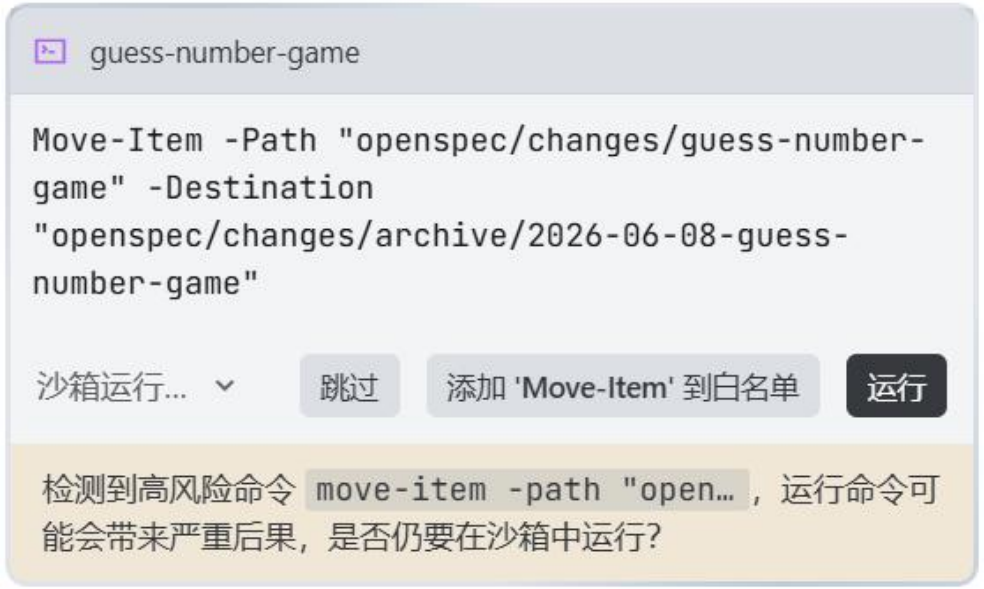
每条验收标准都有对应的代码实现和验证方法。

5.4.5 第四步：归档（/openspec-archive-change）

功能实现并通过验收后，执行/openspec-archive-change将当前变更从 openspec/changes/移动到 openspec/changes/archive/，完成归档，也就是在TRAE的智能体交互界面的对话框中输入如下指令：

```
# 在 TRAE 中输入
/openspec-archive-change guess-number-game
```

在归档命令的执行过程中，可能会出现如图所示的询问界面，可以点击“运行”即可（其他地方遇到类似询问也可以这样处理）。



5.4.5 第四步：归档（/openspec-archive-change）

归档不仅仅是"移动文件"，它标志着该功能的规范文档已进入只读状态，除非有新的变更提案，否则不再修改。归档后的文件会纳入版本管理（git），成为项目历史的一部分。请按照5.3.5节的操作，在TRAE中打开一个Git Bash终端，在Git Bash终端中手动执行下面的命令将规范纳入到仓库中，和代码一起同源管理：

```
git add openspec/changes/archive/2026-05-11-guess-number-game/
git commit -m "feat: archive guess-number-game change"
```

归档赋予了规范文档三个重要属性：

可追溯性

每个功能的完整需求、设计和实现决策都有据可查。

可复用性

遇到类似功能时，直接参考已归档的规范，无需从头设计。

可审计性

规范文档纳入版本管理，团队可以看到每个变更的完整历史。

5.4.6 功能迭代：增加暗黑模式

■ 探索：了解现有实现

在TRAE的智能体交互界面的对话框中，通过以下提示词即可触发探索，TRAE会探索当前的项目目录、功能，然后给出实现建议。

```
/openspec-explore 为猜数字游戏增加暗黑模式
```

AI会基于对现有代码的分析，给出相应的设计方案，例如，本案例中，AI给出了下列建议：（只截取部分内容）

```
### 当前代码结构分析
```
├── index.html   # 主页面（无主题切换按钮）
├── css/styles.css   # 样式文件（纯亮色主题）
├── js/game.js   # 游戏核心逻辑
├── js/storage.js   # 存储管理
└── js/ui.js   # UI 交互逻辑
```

### 暗黑模式实现方案
```

5.4.6 功能迭代：增加暗黑模式

■ 提案：定义暗黑模式规范

在TRAE的智能体交互界面的对话框中使用/openspec-propose指令进行提案，提案会自动使用探索阶段的方案：

```
/openspec-propose 为猜数字游戏增加暗黑模式
```

1. TRAE执行完提案以后，会在openspec/changes目录下创建一个名为add-dark-mode的子目录，其中，add-dark-mode就是本次变更的任务ID。

2. 和前面的流程一样，提案会创建相应的proposal.md、design.md、规范文档和tasks.md，这里不再赘述。

5.4.6 功能迭代：增加暗黑模式

■ 归档：形成知识库

在TRAE的智能体交互界面的对话框中输入如下指令实现归档：

```
# 在 TRAE 中输入
/openspec-archive-change add-dark-mode
```

在TRAE中打开一个Git Bash终端，在Git Bash终端中手动执行下面的命令将规范纳入到仓库中，和代码一起同源管理：

```
git init
git add openspec/changes/archive/2026-06-08-add-dark-mode/
git commit -m "feat: archive add-dark-mode change"
```

5.4.6 功能迭代：增加暗黑模式

当完成了两个功能开发并归档后，项目目录结构如图所示。

5.4.6 功能迭代：增加暗黑模式

从上页图中可以看到，openspec/changes/archive/目录下保存了两个已完成变更的完整规范文档：

2026-06-08-guess-number-game/：猜数字游戏的原始实现

2026-06-08-add-dark-mode/：暗黑模式的功能迭代。

这个目录结构本身就是项目的历史记录。当团队新成员加入时，只需浏览 archive目录，就能了解项目的完整演进历史——每个功能是什么时候加的、为什么加、设计了什么、如何实现的,这些信息都在规范文档中，而不是散落在git提交历史和办公聊天工具中。

5.4.7 其他操作

除了四个核心步骤，OpenSpec还提供以下辅助操作：

规范审查

规范文档编写完成后，提交给团队成员审查，重点检查功能范围是否清晰、数据模型是否合理、验收标准是否可验证。

规范复用

遇到类似功能时，直接参考已归档的规范文档，修改功能范围和数据模型即可快速复用。

5.4.7 其他操作

此外，四个阶段可根据任务复杂度灵活裁剪，具体建议参见下表。

任务复杂度	推荐工作流
简单任务（修复 Bug、小改动）	直接 apply，可跳过 explore 和 propose
中等任务（新增单个功能）	propose → apply → archive
功能迭代（在已有功能基础上新增）	explore → propose → apply → archive
复杂任务（涉及多个能力、架构变更）	完整四阶段

Part 05

待办事项管理应用实践

5.5 待办事项管理应用实践

实践目标与要求

实践实施步骤

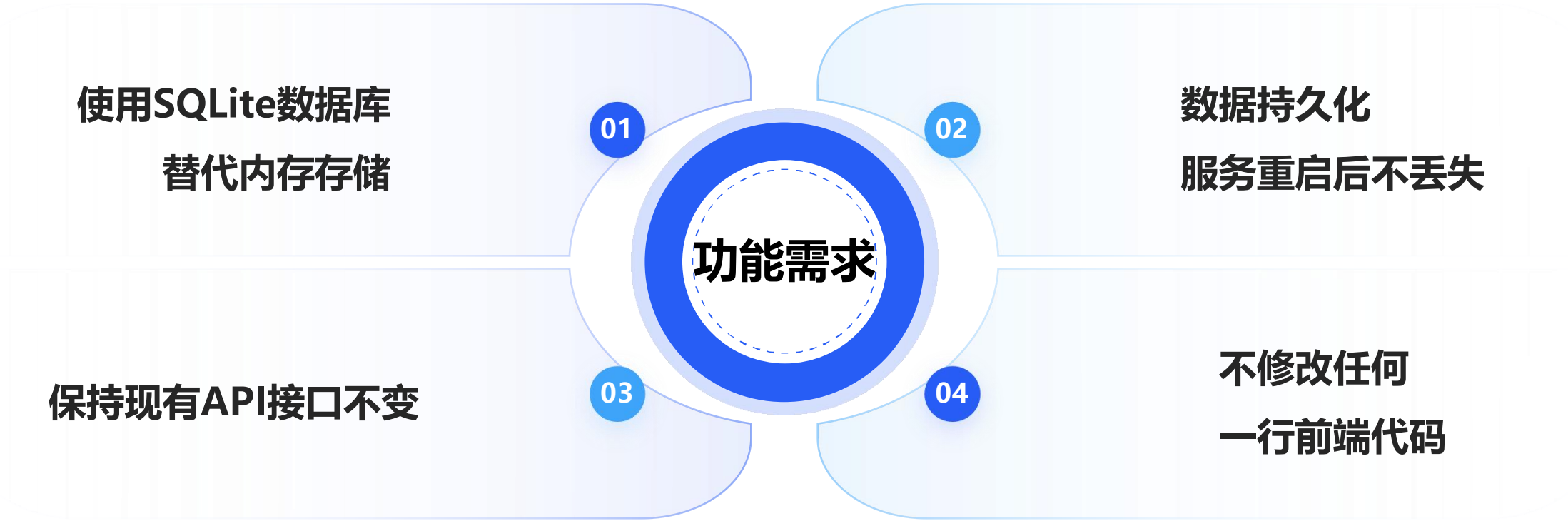
项目验收

5.5.1 实践目标与要求

第4章的4.7节实现了一个基于提示词编程的待办事项（TODO）应用，使用内存存储数据。本节要求读者使用规范驱动编程，对同一个TODO应用进行重构，即引入SQLite数据库实现数据持久化，同时不修改任何一行前端代码，前端保持与第4章完全一致。

在本次重构中，SQLite将替代内存存储，成为 TODO应用的持久化数据载体。

项目的功能需求如下：



5.5.2 实践实施步骤

■ **第一步：探索**

在TRAE的“AI侧栏”的智能体交互界面中，选择Agent模式，新建一个任务（或称为对话），在智能体交互界面的对话框中，输入以下指令：

```
请你严格按照openspec的四个步骤进行功能开发，无确认绝不主动推进、不超前写代码、不跳过任何一个步骤。

固定流程顺序，缺一不可，仅完成当前步骤，等待我确认后再进入下一环。

1. 探索阶段 /skill:openspec-explore 根据用户需求探索当前项目
2. 提案阶段 /skill:openspec-propose 根据探索阶段的讨论进行提案
3. 执行阶段 /skill:openspec-apply-change 根据提案阶段的产出实现代码
4. 归档阶段 /skill:openspec-archive-change 归档变更
```

5.5.2 实践实施步骤

■ 第一步：探索

1.输出首行标注当前所处步骤
2.每步结束统一话术：本步骤完成，请确认进入下一步
3.未收到确认指令，仅限完善当前内容，禁止输出后续内容与代码
4.严禁自主提速、合并步骤、提前完成开发

现在开始第1步，使用/openspec-expore 重构当前的待办事项应用，使用SQLite数据库存储数据，避免服务重启以后数据丢失。

AI会对当前项目进行分析，并给出重构方案，包括变更后的系统架构、需要修改的源码文件、采用的技术栈和可能存在的风险。

5.5.2 实践实施步骤

■ 第二步：提案

在同一个对话中，在智能体交互界面的对话框中，输入提示词“确认”或“下一步”并提交，将会进入到提案阶段。这个时候AI会在openspec/changes目录下创建一个名为”refactor-to-sqlite”的变更，并在该目录下创建proposal.md、design.md、spec和tasks.md文件。

在当前案例中，AI生成的proposal.md文档如下，包含了本次修改的背景和要求：

```
## Why
当前待办事项应用使用内存存储（Python list），服务重启后所有数据会丢失。这导致用户无法持久化保存他们的待办事项，严重影响用户体验。使用 SQLite 数据库替代内存存储可以实现数据持久化。

## What Changes
- 将内存存储 `_todos: list[dict]` 替换为 SQLite 数据库
- 使用 Python 内置 `sqlite3` 模块进行数据库操作
- 创建 `todos` 表，包含 id、title、is_completed、created_at 字段
- 修改 `main.py` 中的 API 路由实现，使用数据库操作替代内存操作
- 保持现有 API 接口不变，前端无需修改
```

5.5.2 实践实施步骤

■ 第二步：提案

```
## Capabilities

### New Capabilities
- `sqlite-persistence`: 实现 SQLite 数据库持久化存储，支持待办事项的增删改查操作

### Modified Capabilities
- `todo-api`: API 实现从内存改为数据库，但接口规范保持不变

## Impact

- `main.py`: 需要完全重写数据访问层，使用 sqlite3 替代内存 list
- `schemas.py`: 保持不变（Pydantic 模型）
- `templates/index.html`: 无需修改（API 接口兼容）
- 新增文件: `todo.db` (SQLite 数据库文件，运行时自动创建)
```

5.5.2 实践实施步骤

除了proposal.md文件以外，用户还需要检查design.md文档中的架构设计和技术选型是否符合要求，如不符合要求，则可以直接修改该文件。

在本例中，AI在design.md中给出了数据库的库表结构和技术栈选型的依据，如下所示：

```
## Context
当前TodoApp使用内存存储（Python list），数据在服务重启后会丢失。需要将数据持久化到SQLite数据库中。

**当前架构:**
```
Frontend (HTML) → FastAPI → Memory (list)
```

**目标架构:**
```
Frontend (HTML) → FastAPI → SQLite Database
```
```

5.5.2 实践实施步骤

Goals / Non-Goals

Goals:

- 使用SQLite数据库替代内存存储
- 实现数据持久化，服务重启后数据不丢失
- 保持API接口不变，前端无需修改
- 使用Python内置`sqlite3`模块，无需额外依赖

Non-Goals:

- 不引入ORM框架（如SQLAlchemy）
- 不修改前端代码
- 不添加新的API端点

5.5.2 实践实施步骤

```
## Decisions

### 1. 数据库选型：SQLite
- **理由**: Python内置支持，无需额外依赖，适合轻量级应用
- **替代方案**: SQLAlchemy - 更强大的 ORM，但增加复杂度和依赖

### 2. 数据库连接管理
- **方案**: 使用`sqlite3.connect()`建立连接，每次请求时打开/关闭连接
- **理由**: SQLite是文件数据库，连接开销低，适合简单场景
### 3. 表结构设计
```sql
CREATE TABLE todos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    is_completed BOOLEAN DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```
```

5.5.2 实践实施步骤

```
### 4. 数据访问层设计
- 在`main.py`中创建辅助函数:
  - `get_db_connection()`: 获取数据库连接
  - `init_db()`: 初始化数据库表（首次运行时自动创建）

## Risks / Trade-offs

| Risk | Mitigation |
|------|------------|
| 数据库文件权限问题 | 使用项目根目录作为数据库路径，确保有写入权限 |
| 并发访问问题 | SQLite 支持并发读，但写操作会锁定数据库 |
| 数据迁移问题 | 首次运行自动创建表，无需手动迁移 |
```

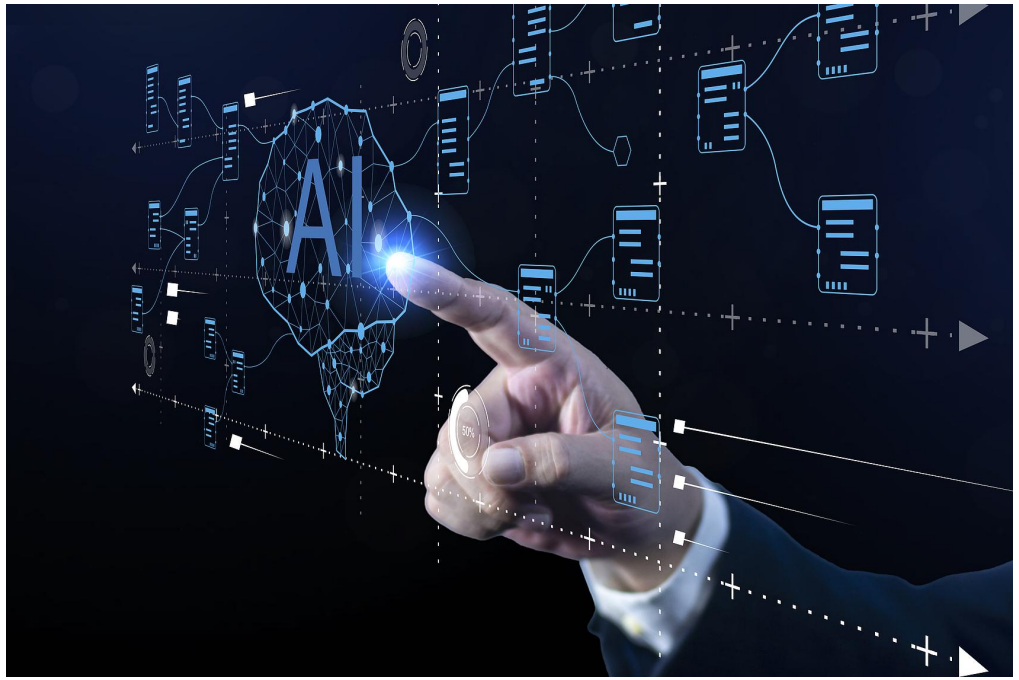
对于AI自动生成的每个specs文件（比如design.dm、proposal.md、tasks.md），在每个文件中，都要点击“保留 Ctrl+Enter”按钮，接受对文件内容的修改，并保存该文件。

5.5.2 实践实施步骤

■ 第三步：实施

因为在第一步中，要求AI严格按照OpenSpec的四步流程进行功能开发，因此，在实施阶段，可以在对话框中输入提示词“确认”或“下一步”并提交，让AI自动生成代码，或者，也可以在对话框中输入“/openspec-apply-change ToDoApp”并提交，让AI开始自动编码。

因为当前变更要求不能修改API，也不能修改前端，因此，主要的代码修改都集中到main.py文件中，增加了对数据库的访问。在这一步中，AI会读取tasks.md中的任务，并依次实现，然后标记认为状态为完成。



5.5.2 实践实施步骤

■ 第四步：归档

在同一个对话中，在智能体交互界面的对话框中，继续输入提示词“确认”或“下一步”并提交，将会进入归档流程。归档流程会将openspec/changes中的“refactor-to-sqlite”目录迁移到openspec/changes/archive目录下。接下来，用户可以使用git提交代码，完成整个功能的开发。在TRAE中打开一个Git Bash终端，在Git Bash终端中手动执行下面的命令将规范纳入到仓库中，和代码一起同源管理：

```
git add openspec/changes/archive/2026-06-08-migrate-to-sqlite/
git commit -m "feat: archive ToDo-App change"
```

5.5.3 项目验收

完成功能开发以后，有多种方式可以验证功能的正确性，可以直接告诉AI运行程序便于验证，也可以在TRAE界面的Git Bash终端中执行下面的命令启动程序：

```
uvicorn main:app --reload
```

程序启动以后，可以访问http://localhost:8000/看到前端页面，可以看到，前端和第4章的页面一模一样，这个时候可以在前端页面插入几条待办事项，便于从数据库内部验证功能正确性。

执行程序以后，会main.py所在目录下自动生成一个名为todo.db的SQLite 数据库文件。可以在Git Bash中登录数据库后台查看数据库的内容，如下所示：

```
# 进入项目目录，连接数据库
sqlite3 todo.db
```

5.5.3 项目验收

执行“.tables”命令，查询数据库表；执行“.schema todos”命令，验证表结构与规范一致：

```
-- 查看所有数据表
.tables
-- 输出：todos

-- 查看t表结构
.schema todos
CREATE TABLE todos (
        id INTEGER NOT NULL,
        title VARCHAR NOT NULL,
        is_completed BOOLEAN,
        created_at DATETIME NOT NULL,
        PRIMARY KEY (id)
);
```

5.5.3 项目验收

查询todos表数据，验证增删改操作后的持久化结果：

```
-- 查询所有待办数据
select * from todos;
1|测试持久化|1|2026-05-25 10:25:24.819974
2|学习基于提示词的AI编程|0|2026-05-25 10:26:14.662469
3|学习基于规范驱动的AI编程|0|2026-05-25 10:26:26.532397
4|准备英语四级考试|0|2026-05-25 10:26:32.556445
```

Part 06

与提示词编程的对比

5.6 与提示词编程的对比

工作流程对比

适用场景对比

混合策略

5.6.1 工作流程对比

提示词编程和规范驱动编程的工作流程有本质区别，具体对比如表所示。

维度	提示词编程	规范驱动编程
起点	自然语言描述需求	结构化规范文档
过程	多轮对话迭代	四步流程（需求→方案→拆解→实现）
输出	可运行代码	规范文档 + 可运行代码
验证	人工检查运行结果	对照验收标准逐条验证
可追溯性	低（对话历史难以检索）	高（规范文档是事实来源）
协作性	弱（依赖个人经验）	强（规范文档是团队共识）

5.6.2 适用场景对比

两种方法各有擅长的场景。提示词编程更适合以下场景：

01 快速原型验证

需要快速验证一个想法是否可行时，提示词编程可以在几分钟内生成可运行的原型。

02 一次性脚本

数据处理脚本、自动化任务等不需要长期维护的代码。

03 个人项目

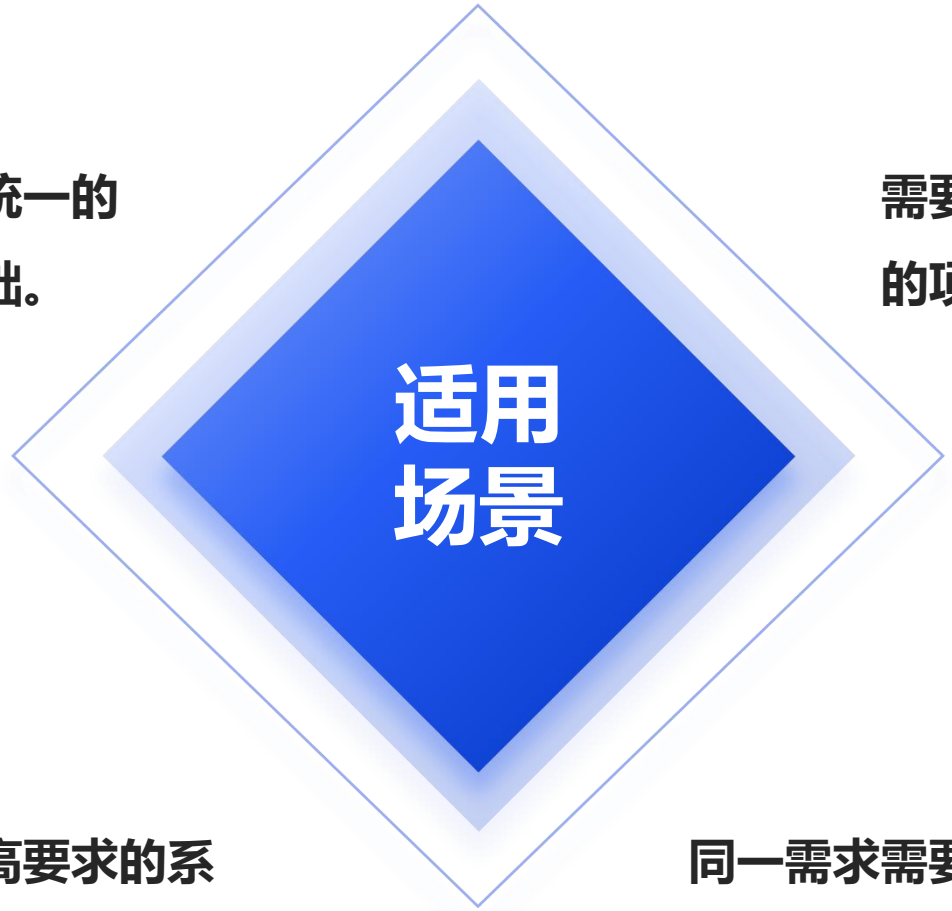
只有自己使用的工具，不需要与团队协作。

04 探索性开发

不确定技术方案时，通过多轮对话探索可能的实现路径。

5.6.2 适用场景对比

规范驱动编程更适合以下场景：



团队协作项目

多人参与的项目需要统一的规范文档作为协作基础。

长期维护项目

需要持续迭代和多人接手的项目，规范文档是可传承的工程资产。

关键业务系统

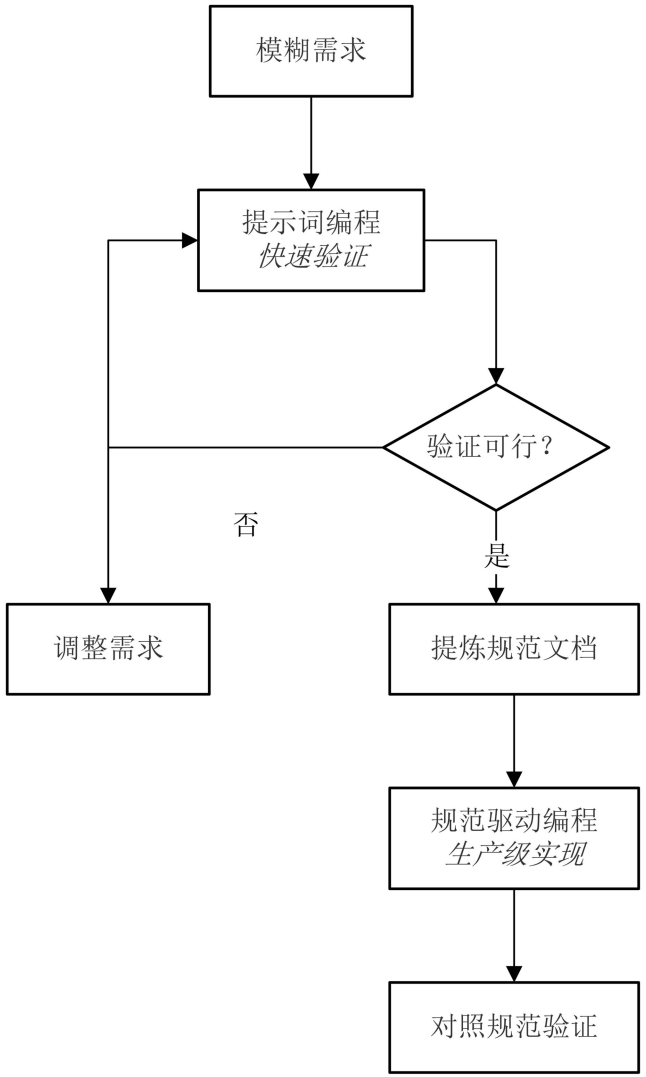
对正确性和可靠性有高要求的系统，需要明确的验收标准。

多语言实现

同一需求需要用不同语言实现时，规范文档是共同语言。

5.6.3 混合策略

在实际项目中，两种方法并非二选一，而是可以结合使用。推荐采用"先验证、后规范"的混合策略，具体流程如图所示。



5.6.3 混合策略

混合策略的执行步骤如下：

用提示词编程快速验证

将模糊需求直接交给 AI，快速生成原型代码，验证技术方案的可行性。

提炼关键决策为规范

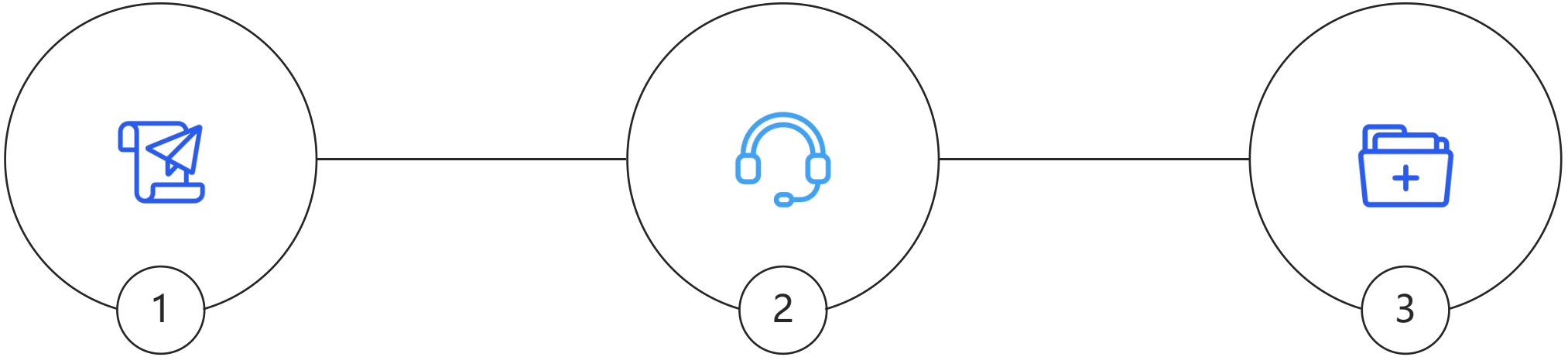
验证可行后，将原型中的关键决策（数据模型、接口契约、边界条件）提炼为规范文档。

用规范驱动编程重新实现

基于规范文档，让 AI 重新生成生产级代码，确保代码满足所有验收标准。

5.6.3 混合策略

这里给出一个混合策略示例。以5.5节的TODO应用为例，假设团队不确定"用 SQLite替换内存存储"是否会影响前端兼容性，则可以使用如下混合策略：



第一步

开发者用提示词编程快速生成一个“SQLite + FastAPI”的原型，手动测试确认前端可以正常调用；

第二步

将验证通过的技术方案（SQLite文件路径、SQLAlchemy模型定义、API路径保持 /api/todos）提炼为规范文档；

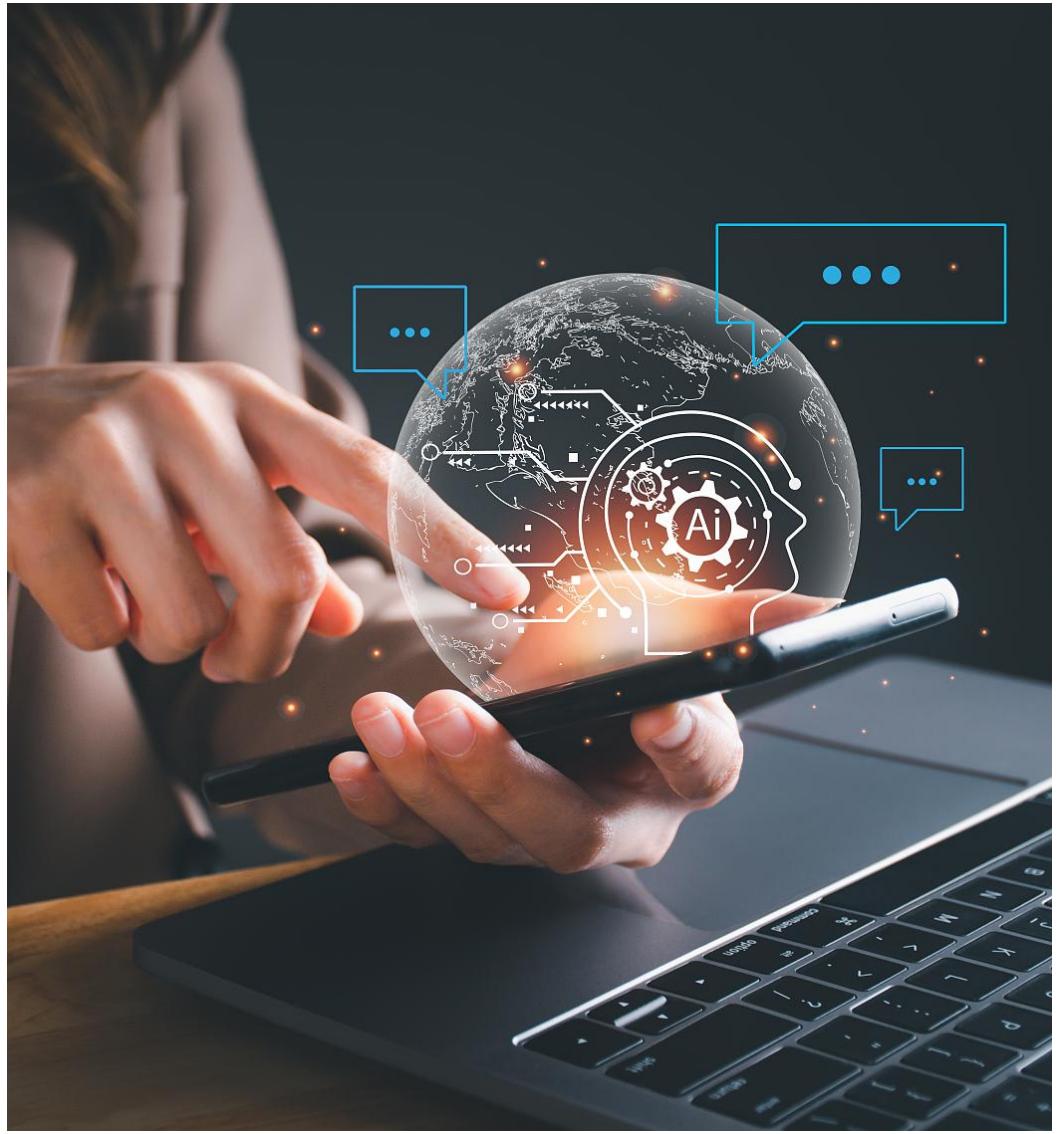
第三步

基于规范文档让AI重新生成完整的生产级代码，包括错误处理、边界条件检查和数据库迁移脚本。

5.6.3 混合策略

这种策略的优势在于：提示词编程阶段解决了"能不能做"的问题，规范驱动编程阶段解决了"做得好不好"的问题。两者结合，既保证了开发效率，又保障了代码质量。

混合策略的关键转折点是"验证可行"。一旦确认技术方案可行，就应立即切换到规范驱动编程模式，避免在提示词编程中过度迭代，因为提示词编程的迭代成本会随着代码复杂度的增加而急剧上升



5.7 本章小结

本章系统介绍规范驱动编程（Spec-Driven Development，SDD）的理念、工具链与实践方法。

SDD的核心思想是“无规范不写代码”，即在让AI编写代码前，先通过结构化规范明确“做什么、怎么做、有何约束”，并以规范作为开发起点和验收依据。

其三条铁律包括：无规范不写代码、规范即真理、逆向同步；当代码与规范不一致时，应以规范为准，发现Bug时也应先修正规范，再修正代码。

本章还介绍了OpenSpec的探索、提案、实施和归档四个步骤，并通过“猜数字游戏”和“待办事项管理”案例展示落地流程。

SDD与提示词编程并非替代关系，而是互补关系：提示词编程适合快速验证想法，SDD适合保障团队协作和交付质量。规范文档不仅是AI输入，更是团队可传承的工程资产。

谢谢！

林子雨　副教授　厦门大学

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn

个人网页：http://dblab.xmu.edu.cn/post/linziyu

数据库实验室网站：http://dblab.xmu.edu.cn

- 林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据与人工智能教师，入选“2021年高校计算机专业优秀教师奖励计划”。现为厦门大学计算机科学与技术系副教授，厦门大学信息学院实验教学中心主任，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会执行委员，中国计算机学会信息系统专业委员会执行委员。
- 国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度、2020年度、2023年度和2026年度厦门大学教学类奖教金获得者，荣获2024年福建省高等教育教学成果奖特等奖（个人排名第七）、2022年福建省高等教育教学成果奖特等奖（个人排名第一）、2018年福建省高等教育教学成果奖二等奖（个人排名第一）、2018年国家精品在线开放课程、2021年国家级线上一流本科课程、2020年国家级线上一流本科课程。编著出版了20余本大数据与人工智能系列教材，被国内1000多所高校采用，3本教材入选教育部“十四五”普通高等教育本科国家级规划教材；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过400万次，累计访问量超过3200万次。大数据系列MOOC课程入选“2023年教育部国家智慧教育公共服务平台应用典型案例”。2025年发布4个大模型科普报告，全网累计浏览量超过1000万，并应邀为高校、政府和企业做170余场大模型讲座。
- 主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题、1项2016年教育部产学协作育人项目、1项2024年教育部产学协作育人项目。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：林子雨人工智能通识系列教材

厦门大学林子雨人工智能通识系列教材

被众多高校采用，满足不同高校差异化教学需求

了解全部教材信息：http://dblab.xmu.edu.cn/post/ai-book/

附录E：林子雨Python系列教材

了解全部教材信息：https://dblab.xmu.edu.cn/post/python-books/

谢谢！

林子雨　副教授

厦门大学