Code Agent 核心概念及工具箱

课程逐字稿（讲师讲解词）

第二章

共 61 页幻灯片

第 1 页

【幻灯片标题】课程封面

【逐字稿】

同学们好，欢迎来到第二章《Code Agent 核心概念及工具箱》。在上一章中，我们学习了 AI 编程的发展历程，了解了编程语言的三次革命和 AI 编程工具的三代演进。这一章我们将深入 Code Agent 的核心概念和它所使用的各种工具。如果说上一章是「道」——理解为什么 AI 编程可行，那么这一章就是「术」——掌握具体怎么用 AI 编程。Code Agent 是当前发展最快速、落地最广泛的 Agent 应用之一，理解它的核心概念和工具箱，是实践 AI 编程的基础。让我们开始吧！

第 2 页

【幻灯片标题】前言

【逐字稿】

本章主要讲述 Code Agent 的实用工具及技巧。我们会通过学习 Prompt、RAG、Agent 等相关技术，以及 Skills、Cli、MCP、插件等工具，了解从 Prompt 到 Context 再到 Harness Engineering 的发展趋势。这些概念可能看起来很多很杂，但它们有一个共同的主线——都是为了让大模型更好地完成任务而存在。大模型本身只是一个「大脑」，它需要眼睛（RAG）、双手（工具调用）、记忆（上下文管理）、规矩（Rule）才能成为真正有用的 Code Agent。本章的目标是帮大家掌握这些前置知识，为后续实践 AI Coding 开发夯实基础。内容比较多，请大家跟上节奏。

第 3 页

【幻灯片标题】课程目标

【逐字稿】

我们来看本章的学习目标。学完之后，大家将能够：第一，了解 Code Agent 的概念及架构——知道 Code Agent 由哪些组件构成，各组件如何协作。第二，掌握 Prompt 概念及要素——知道如何写好提示词来引导 AI。第三，了解 RAG 概念及要素——知道如何给 AI 喂外部知识。第四，掌握大模型的局限及外部连接工具——理解 Function Calling 和 MCP 如何扩展 AI 的能力边界。第五，掌握大模型的操作手册——理解 Skills 如何封装可复用的能力。第六，了解 Code Agent 的拓展概念——包括 Cli、Hook、Rule、上下文工程、驾驭工程等。这六个目标从核心到拓展，从基础到进阶，构成了 Code Agent 的完整知识图谱。

第 4 页

【幻灯片标题】目录

【逐字稿】

我们来看本章的目录。本章分为六个部分：第一部分是 Code Agent 概念及架构，从整体上理解 Code Agent 是什么。第二部分是 Prompt 提示词工程概览，学习如何与 AI 有效沟通。第三部分是 RAG 检索增强生成概览，学习如何给 AI 外挂知识库。第四部分是 MCP 与 Function Calling 概览，学习如何扩展 AI 的能力边界。第五部分是 Skills 概览，学习如何封装可复用的 AI 能力。第六部分是 Code Agent 拓展介绍，了解更高级的概念。这六部分从核心概念到工具使用到工程化拓展，层层递进。我们逐一展开。

第 5 页

【幻灯片标题】Code Agent 核心概念

【逐字稿】

我们先来理解 Code Agent 的核心概念。Code Agent 作为发展最快速、落地最广泛的 Agent 应用之一，它充分利用并优化了大模型应用相关技术，从实验室概念逐步成为程序员们手中的日常生产力。Code Agent 的核心组件有哪些呢？大语言模型——负责理解和生成的 AI 大脑。执行环境——代码运行沙盒。工具调用——包括 Function Calling、MCP、Cli 等，让 AI 能操作外部系统。Hook 脚本——在特定时机自动执行的插件。SubAgent——子智能体，处理子任务。上下文管理——管理对话和项目上下文。记忆系统 Memory——短期和长期记忆。Rule 规则——约束 AI 行为。Skill 列表——可复用的能力单元。这些组件协同工作，使 Code Agent 能够自主完成从需求理解到代码生成到测试验证的全流程。后续我们会逐一深入每个组件。

第 6 页

【幻灯片标题】大模型能力需要"激活"

【逐字稿】

我们来看一个关键认知——大模型的能力需要「激活」。大模型本身有强大的能力，但它不会主动做事，需要人类的有效引导和赋能。什么是提示词（Prompt）？简单说就是从需求到传递到输出的过程。你有一个需求，通过 Prompt 把需求传递给大模型，大模型输出结果。核心是三步：明确需求→传递意图→引导输出。举个通俗的例子：大模型就像一个能力超强但被动的员工，它什么都会做，但不会主动做。你需要明确告诉它要做什么（明确需求），用它能理解的方式表达（传递意图），并指定你想要的输出形式（引导输

出）。这就是 Prompt Engineering 的本质——让大模型从「有能力」到「能做事」。一个好的 Prompt 可以让普通的模型表现出色，一个差的 Prompt 可以让强大的模型表现糟糕。所以 Prompt Engineering 是使用 AI 编程的第一项核心技能。

第 7 页

【幻灯片标题】什么是检索增强生成（RAG）

【逐字稿】

接下来讲检索增强生成 RAG。RAG 全称 Retrieval-Augmented Generation，是一种结合了信息检索和文本生成的技术。它允许大语言模型通过从数据源检索信息来生成回答，从而提高回答的准确性和上下文相关性。为什么需要 RAG？因为大模型的知识来自训练数据，训练完成后知识就固定了。如果你问它最新的信息，或者你公司的内部文档，它不知道。RAG 的解决思路是：不重新训练模型，而是在生成回答前，先从外部知识库检索相关信息，把检索到的信息拼接到 Prompt 中，让模型基于这些信息来回答。这样模型就能回答它训练数据中没有的问题。打个比方：大模型像一个博学的教授，但他读过的书是有限的。RAG 就是在他回答问题前，先去图书馆查相关资料，然后基于查到的资料来回答。这样教授就能回答任何领域的问题了。

第 8 页

【幻灯片标题】什么是智能体（Agent）

【逐字稿】

我们来讲什么是智能体 Agent。Agent 是由 LLM 驱动的自主代理系统，是一种能够感知环境并根据感知结果采取行动的实体。Agent 和普通 LLM 对话的区别在于「自主性」—

—你给它一个目标，它能自主规划步骤、调用工具、执行操作。Agent 有三个核心系统：第一，规划系统——包括子目标分解和结果反馈改进。比如你让 Agent「开发一个网站」，它会自主分解为设计、前端、后端、测试等子任务，逐个完成，遇到问题会根据反馈调整。第二，记忆系统——包括短期记忆（当前对话上下文）和长期记忆（外部数据库存储的历史信息）。记忆让 Agent 能在多轮交互中保持连贯。第三，工具使用——调用外部 API、算法、工具等。工具让 Agent 能突破对话框的限制，操作文件系统、调用 API、执行代码。这三个系统合在一起，使 Agent 从「被动回答」进化为「主动执行」，这就是 Code Agent 能自主完成开发任务的技术基础。

第 9 页

【幻灯片标题】什么是上下文（Context）

【逐字稿】

我们来讲什么是上下文 Context。上下文指模型在生成响应时输入的信息总和，包括当前 Prompt、历史对话、RAG 检索到的知识、Agent 提供的系统指令等。你可以把上下文理解为模型在某一刻「看到的所有信息」。上下文管理的目标是什么？在有限的 Token 内，给模型高质量信息。这里有一个关键约束——模型的上下文窗口是有限的，比如 4K、32K、128K 个 Token。信息太少模型理解不充分，信息太多又超出窗口限制。所以上下文管理的本质是信息压缩和筛选——在有限空间内，给模型最相关、最准确、最完整的信息。在 Code Agent 中，上下文管理尤其复杂，因为要同时管理代码文件、对话历史、工具定义、项目规则等多种信息源。后续我们会专门讲上下文工程 Context Engineering。

第 10 页

【幻灯片标题】目录

【逐字稿】

我们进入第二部分——Prompt 提示词工程概览。前面讲了 Code Agent 的核心概念，现在我们深入第一个核心技能——如何写好提示词。Prompt Engineering 是与 AI 沟通的基础，掌握它，你才能有效地让 AI 理解你的意图并生成高质量输出。这部分会讲清楚：什么是提示词工程、为什么需要它、提示词的组成要素、设计技巧、实用框架以及调优流程。

第 11 页

【幻灯片标题】什么是提示词工程

【逐字稿】

什么是提示词工程 Prompt Engineering？它包含三个层面的内容。要素层面：指令——告诉 AI 做什么；输入数据——给 AI 处理的数据；输出指示——指定输出格式；上下文——提供背景信息。技巧层面：目标——明确任务目标；任务步骤——详细说明步骤；约束——划定职责范围；背景知识——补充领域信息；角色——设定 AI 身份；技能——指定 AI 能力；输入格式和输出格式——规范数据格式；示例——提供参考样例。框架层面：有多种场景化最佳实践框架，如 COAST、APE、CARE、BROKE、ROSES 等。这些框架是前人总结的提示词模板，在不同场景下选用不同框架可以事半功倍。提示词工程看起来简单——不就是写几句话嘛，但要做到精通需要大量实践。同一个任务，不同的 Prompt 可能得到天差地别的结果。

第 12 页

【幻灯片标题】为什么需要提示词工程

【逐字稿】

为什么需要提示词工程？核心原因有三点。第一，模型生成结果的优劣取决于模型能力和提示词质量两个因素——模型能力是固定的，你能控制的就是提示词质量。第二，使用不同的 prompt 会获得不同的结果——同一个模型，好的 prompt 可能得到 90 分的结果，差的 prompt 可能只有 30 分。第三，提示词工程相比于其他模型能力提升手段（如微调、重新训练）更简单经济——写好 prompt 几分钟就行，微调模型需要大量数据和算力。举个实际例子：你让 AI 写一个排序算法，如果说「写个排序」，AI 可能给你一个简单的冒泡排序；如果说「写一个时间复杂度 O(n log n)的排序算法，处理大数据量，要求稳定排序，加注释」，AI 就会给你一个优化过的归并排序。同样的模型，不同的 prompt，结果质量完全不同。所以提示词工程是使用 AI 编程时投入产出比最高的技能。

第 13 页

【幻灯片标题】提示词的组成要素

【逐字稿】

我们来看提示词的组成要素——具体应该写什么。一个完整的提示词包含四个要素：指令（做什么）+ 上下文（为什么）+ 输入数据（基于什么）+ 输出指示（做成什么样）。用一个生活化的例子来理解：你让 AI 帮你写检讨书，你需要告诉它：写什么内容的检讨（指令）——比如迟到；给谁看（上下文）——比如给班主任；基于什么事实（输入数据）——比如周三早自习迟到 10 分钟；写成什么样（输出指示）——比如 800 字以上，态度诚恳，分段写。四要素缺一不可：缺指令 AI 不知道做什么；缺上下文 AI 不了解背景；缺输入数据 AI 没有处理素材；缺输出指示 AI 不知道你要什么格式。在实际使用中，最常犯

的错误是只给指令不给其他三要素，导致 AI 输出不符合预期。记住这个公式：要素 = 做什么 + 为什么要做 + 基于什么做 + 做成什么样。

第 14 页

【幻灯片标题】提示词的设计技巧

【逐字稿】

我们来看提示词的设计技巧——具体怎么写好。有六个核心技巧。第一，目标——明确 LLM 要完成的任务目标，越具体越好。不要说「写代码」，要说「写一个 Python 函数，输入列表返回排序后的列表」。第二，任务步骤——完成任务所需的详细步骤说明。比如「先分析需求，再设计数据结构，最后实现逻辑」。第三，约束——明确大模型的职责范围。比如「只处理正整数，负数报错」。第四，角色——让大模型扮演特定角色。比如「你是一个资深 Python 工程师」。第五，示例——提供遵守规范和格式的高质量示例。AI 看到示例就能模仿。第六，输出格式——指定输出格式，如表格、JSON、Markdown 等。这六个技巧不需要每次都用全部，但用得越多，输出质量越高。核心原则是：你给 AI 的信息越充分、越明确，AI 给你的结果就越好。

第 15 页

【幻灯片标题】简单好用的提示词框架

【逐字稿】

我们来学一个简单好用的提示词框架。四步法：设立角色→描述问题→确定目标→增补要求。我们对比看一个例子。非专业 Prompt：「我想写一个大模型的培训材料，能否给我做一个提纲？」——这个 prompt 没有角色、没有背景、没有格式要求，AI 只能给你

一个通用的提纲。专业 Prompt：「你是一位专业的学习与发展架构师（设立角色），我需要为新入职的程序员编写一份大模型应用培训材料（描述问题），目标是让学员在 3 天内掌握大模型基础概念和实操技能（确定目标），请按天划分章节，每天包含理论部分和实操练习，输出 Markdown 格式（增补要求）」。对比一下，专业 Prompt 生成的提纲会精确到每天讲什么、做什么练习，直接可用。这就是提示词框架的价值——它帮你系统化地思考，不遗漏关键信息。

第 16 页

【幻灯片标题】提示词工程调优流程

【逐字稿】

提示词工程有一个调优流程，不是一次就能写出完美 prompt 的。标准流程是：准备→Prompt 自由创作→选取「最优」Prompt→二次调优→效果验证→交付。具体来说：先准备任务描述和测试用例，然后尝试写多个不同风格的 Prompt，测试哪个效果最好，选出最优的那个，再针对性调优细节，验证效果是否稳定，最后交付使用。两个常见误区要避免：第一，过度指令和模糊指令——既给了太多约束又不够明确，自相矛盾。第二，期待一次性给出完美结果——Prompt 调优是迭代的，需要多轮测试改进。在实际 AI 编程中，你可能需要 3-5 轮调优才能得到一个稳定的 Prompt。这是正常投入，不要急躁。

第 17 页

【幻灯片标题】目录

【逐字稿】

我们进入第三部分——RAG 检索增强生成概览。前面讲了 Prompt，现在讲 RAG。如果说 Prompt 是教你怎么跟 AI 说话，那 RAG 就是教你怎么给 AI 喂知识。RAG 是 Code Agent 的「眼睛」，让 AI 能看到训练数据之外的信息。这部分会讲清楚：什么是 RAG、为什么需要它、它的核心组件、以及 Code Agent 中的检索技术。

第 18 页

【幻灯片标题】什么是 RAG

【逐字稿】

我们来正式定义 RAG。RAG 全称 Retrieval-Augmented Generation，检索增强生成，结合了信息检索和文本生成技术。它使大型语言模型通过从数据源检索信息来生成回答，提高回答的准确性和上下文相关性。RAG 的工作原理分三步：第一步，用户提问。第二步，系统从外部知识库中检索与问题相关的信息。第三步，把检索到的信息和原始问题一起发给大模型，模型基于这些信息生成回答。举个例子：你问 AI「华为云 ECS 的最新定价是多少？」如果没有 RAG，AI 要么说不知道，要么编一个错误答案（幻觉）。有了 RAG，系统先从华为云定价文档中检索相关信息，然后把检索到的定价数据和你的问题一起发给 AI，AI 基于真实的定价数据来回答，准确无误。RAG 的本质是：不修改模型本身，而是通过外挂知识库来增强模型的知识范围。

第 19 页

【幻灯片标题】为什么需要 RAG

【逐字稿】

为什么需要 RAG？大模型面临几个挑战：第一，缺乏领域内知识——大模型的训练数据是通用的，不了解你公司的业务。第二，知识更新节奏慢——模型训练完成后知识就固定了，世界在变它不知道。第三，训练成本高昂——重新训练一次大模型需要数百万美元。第四，幻觉问题——不知道的时候会编答案。第五，安全合规问题——某些知识不能放进训练数据。RAG 工程怎么解决这些问题？第一，外挂领域知识库——把领域知识放在外部库中，按需检索。第二，持续更新外挂知识库——知识库可以随时更新，不需要重新训练模型。第三，注明知识来源——回答时标注信息来源，可溯源可审计。第四，对知识文档设置权限——不同用户检索不同权限的文档，保证安全。RAG 的这些特性使它成为企业落地 AI 的首选方案——用最小的成本解决大模型的知识局限问题。

第 20 页

【幻灯片标题】RAG 核心组件介绍

【逐字稿】

我们来看 RAG 的核心组件和完整流程。共六步：第一步，文档预处理——把原始文档解析、清洗、切分成小块。第二步，向量化——把文本块转成向量表示。第三步，Query 重写——把用户的原始问题改写更适合检索的形式。第四步，检索与重排序——从向量库中检索相关文档，然后重排序选出最相关的。第五步，增强提示——把检索到的文档拼接到 Prompt 中。第六步，大模型生成回应——模型基于增强后的 Prompt 生成回答。这六步构成了 RAG 的完整链路，每一步都有技术细节。接下来几张幻灯片我们会逐步展开每一步的具体内容。

第 21 页

【幻灯片标题】文档预处理

【逐字稿】

先看第一步——文档预处理。原始文档格式多样，有 PDF、Word、HTML、Markdown 等。如果直接切碎，可能导致语义断裂——一句话被切成两半，检索时匹配不上。文档预处理分三步：第一步，文档解析与内容提取——把各种格式统一提取为纯文本。第二步，文本清洗——去除无关字符、格式标记、噪音数据。第三步，语义切块（切片）和元数据注入——按语义边界切分，每个切片附上来源、页码、标题等元数据。语义切块是关键——不能按固定字数切，要按段落、章节、语义边界切，保证每个切片语义完整。比如一个代码函数不能被切成两半。文档预处理的质量直接决定 RAG 的检索质量——垃圾进垃圾出，预处理做不好，后面的步骤再优化也没用。

第 22 页

【幻灯片标题】向量化

【逐字稿】

第二步——向量化。向量化 Embedding 是将离散数据转换为连续向量的技术。它的作用是通过向量空间中的距离度量语义相似度，从而让机器能够理解和检索内容的含义。通俗地说：把每段文本变成一个高维向量（比如 768 维的一个数字数组），语义相近的文本在向量空间中距离也近。这样当你提问时，把问题也变成向量，然后在向量空间中找最近的几个文档向量，就是语义最相关的文档。向量化是 RAG 的核心技术——它让「语义搜索」成为可能。传统的关键词搜索只能匹配字面相同的词，向量搜索能匹配语义相同但字面不同的内容。比如搜索「如何部署应用」，向量搜索能找到「怎样上线服务」的文档，虽然字面不同但语义相同。

第 23 页

【幻灯片标题】主流开源向量化模型

【逐字稿】

我们来看主流的开源向量化模型。第一个，bge-m3——多功能合一，多语言领先，最大支持 8192 个 tokens，适合多语言场景。第二个，m3e-base——中文任务性能强，1.1 亿参数轻量，适合纯中文场景。第三个，bge-large-zh-v1.5——中文 C-MTEB 榜单第一，中文检索效果最好。第四个，paraphrase-multilingual-MiniLM-L12-v2——极速轻量，支持 50+语言，适合对速度要求高的场景。选择向量化模型时需要权衡：中文场景优先选 bge 或 m3e；多语言场景选 bge-m3；速度优先选 MiniLM。向量化模型的选择直接影响 RAG 的检索质量，建议根据实际场景测试对比后选择。

第 24 页

【幻灯片标题】检索与重排序

【逐字稿】

第三步和第四步——检索与重排序。这个过程跟图书管理员的工作类似：理解用户含义、找到相关内容、给用户推荐最合适的内容。在检索时需要权衡几个因素：文档相关性——检索结果跟问题的相关程度；返回数量——返回太少可能遗漏，太多可能噪音；排名最高文档——最相关的应该排在最前；监控与实验——持续监控检索质量，通过实验优化参数。检索是 RAG 的核心环节——检索质量直接决定最终回答质量。如果检索不到相关文档，大模型就没有可参考的信息；如果检索到太多不相关文档，会干扰大模型的判断。所以检索参数需要反复调优。

第 25 页

【幻灯片标题】检索与重排序（续）

【逐字稿】

我们继续看检索与重排序的具体技术。有四种检索方式：第一，关键词搜索——对精确字词检索，保证关键字敏感度。比如搜「K8s」能精确匹配含「K8s」的文档。第二，语义搜索——查找相似含义文档，即使没有匹配词语。比如搜「容器编排」能找到讲「Kubernetes」的文档。第三，元数据过滤——根据设定条件排除文档。比如只搜某个日期范围内的、某个部门的文档。第四，重排序——根据混合检索文档进行重排。多种检索方式各有优劣，实际中通常混合使用——先用多种方式各自检索一批候选，然后用重排序模型统一打分排序，选出最相关的。这种混合检索加重排序的方案是目前 RAG 的最佳实践。

第 26 页

【幻灯片标题】Code Agent 中的检索技术

【逐字稿】

我们来看 Code Agent 中特有的检索技术。有五种检索方式：第一，词法/grep——关键词匹配，使用 BM25 算法，是稀疏检索器。适合精确匹配变量名、函数名。第二，向量检索——语义搜索，使用嵌入模型加向量数据库，是稠密检索器。适合查找语义相关的代码。第三，结构化/图谱检索——知识图谱增强，抽取代码结构关系。比如类继承关系、函数调用链。第四，符号定位——快速索引，类似 IDE 的定义跳转。直接跳到变量或函数的定义位置。第五，混合检索——多路检索融合，兼顾精确和语义。Code Agent 的检索比

通用 RAG 更复杂——因为代码有结构、有依赖、有类型信息，需要多种检索方式协同才能准确定位相关代码。这也是为什么 Code Agent 的检索能力是核心竞争力之一。

第 27 页

【幻灯片标题】目录

【逐字稿】

我们进入第四部分——MCP 与 Function Calling 概览。前面讲了 Prompt 和 RAG，现在讲工具调用。大模型本身被关在对话框里，不能操作文件、不能调用 API、不能执行代码。工具调用就是给大模型装上「双手」，让它能突破对话框的限制。这部分会讲清楚 Function Calling 和 MCP 两种核心工具调用技术。

第 28 页

【幻灯片标题】大模型的局限性及工具调用

【逐字稿】

我们先理解大模型的局限性及工具调用的必要性。大语言模型无法突破对话框的限制进行自主行动——它只能接收文本输入、输出文本，不能直接操作文件系统、不能调用外部 API、不能执行代码。但实际开发中，我们需要 AI 能读文件、写文件、查数据库、调 API。工具调用就是扩展大模型能力的关键技术。工具调用技术的演进路径是：手动 API 集成→Function Calling→MCP→CLI。最早是手动集成——人写代码调 API，AI 只给建议。然后是 Function Calling——AI 能自动调预定义的函数。再到 MCP——标准化协议，统一连接各种工具。最后是 CLI——AI 直接执行命令行操作。每一步演进都让 AI 的自主能力更强。接下来我们逐一讲解。

第 29 页

【幻灯片标题】什么是 Function Calling

【逐字稿】

什么是 Function Calling？Function Calling 指大模型调用预定义函数的能力，允许模型生成请求参数并由外部系统执行具体操作，将结果反馈给模型。完整流程是：用户提问→模型组合工具→解析意图生成参数→调用外部函数→返回结果→生成自然语言回复。举个具体例子：用户问「今天北京天气怎么样？」模型知道有天气查询函数，生成调用参数 {city: '北京', date: 'today'}，外部系统调用天气 API 获取结果，把结果返回给模型，模型用自然语言回复「北京今天晴，25 度」。Function Calling 的关键是：模型不直接执行函数，而是生成调用参数，由外部系统执行。这样保证了安全性——模型不能随意执行危险操作。Function Calling 是工具调用最基础的形式，简单有效，但每个工具需要单独定义和适配。

第 30 页

【幻灯片标题】什么是 MCP

【逐字稿】

什么是 MCP？MCP 全称 Model Context Protocol，模型上下文协议，是 Anthropic 公司在 2024 年底主导发布的开放通用协议标准。MCP 使 LLM 能标准化接入外部数据源与工具。如果说 Function Calling 是「每个工具单独写适配」，MCP 就是「写一次适配，所有模型通用」。MCP 定义了一套标准协议——任何工具只要按 MCP 协议实现，就能被任何支持 MCP 的模型直接使用，不需要单独适配。这大大降低了工具集成的重复劳动。

MCP 的愿景是成为 AI 工具调用的「USB 接口」——一个标准接口连接所有设备。目前 MCP 已经获得广泛生态支持，大量工具和服务提供了 MCP Server 实现。

第 31 页

【幻灯片标题】为什么需要 MCP

【逐字稿】

为什么需要 MCP？核心原因是 Function Calling 的痛点。Function Calling 面向大量工具及模型应用场景时，需要逐一适配——每个工具为每个模型写一套适配代码，造成大量重复劳动。比如你有 10 个工具、5 个模型，就需要写 50 套适配代码。MCP 为大模型与外部数据和工具的无缝集成提供了标准化通信协议。有了 MCP，10 个工具各写一个 MCP Server，5 个模型都支持 MCP 协议，总共只需要 10 套代码，不需要为每个模型单独适配。这大幅降低了集成成本和开发工作量。MCP 的意义不仅是技术层面的简化，更是生态层面的统一——它让工具开发者和模型开发者可以各自独立发展，通过标准协议对接，促进整个 AI 工具生态繁荣。

第 32 页

【幻灯片标题】MCP 内部架构

【逐字稿】

我们来看 MCP 的内部架构。MCP 遵循 CS 架构，即客户端-服务器架构。三个角色：第一，主机 MCP Hosts——发起连接的一方，通常是 AI 编程工具，如 CodeArts、Claude Code 等。第二，客户端 MCP Clients——与服务器 1:1 连接的中间层，每个客户端连接一个服务器。第三，服务器 MCP Servers——提供上下文工具和提示信息的一方，比如数据

库 MCP Server、文件系统 MCP Server、API MCP Server 等。工作流程是：Host 需要某个工具能力时，通过 Client 向对应的 Server 发请求，Server 执行操作后返回结果。这种架构的好处是解耦——工具开发者只需要实现 MCP Server，不需要关心哪个模型在用它；模型开发者只需要支持 MCP 协议，不需要关心有哪些工具可用。

第 33 页

【幻灯片标题】MCP 与大模型交互流程

【逐字稿】

我们来看 MCP 与大模型交互的完整流程。共九步：用户 Query→系统把 Query 和可用 MCP Server 列表一起发给 LLM→LLM 做工具使用决策（需要用哪个工具）→系统提供工具调用说明→LLM 输出函数调用参数→系统发起调用请求→路由分发到对应的 MCP Server→Server 执行并返回结果→LLM 基于结果生成最终响应。这个过程对用户是透明的——用户只看到自己问了问题，AI 给了回答，中间的工具调用是自动完成的。比如你问「帮我查看数据库里的用户表结构」，AI 自动通过 MCP 调用数据库工具，获取表结构，然后告诉你结果。你不需要知道 MCP 的存在，但 MCP 在背后默默工作，让 AI 能操作外部系统。

第 34 页

【幻灯片标题】MCP 连接及协同外部服务的典型工作流

【逐字稿】

我们来看 MCP 连接及协同外部服务的典型工作流。共九步：初始化客户端→发送初始化请求→服务器响应→发送确认通知→开始消息交换→请求服务→调用工具→返回结果

→传递响应。这个过程建立了客户端和服务器之间的稳定连接通道。一旦连接建立，后续的工具调用就可以在这个通道上反复进行，不需要每次重新连接。这就像建立了一个专线——一次拨通，后续随时通话。MCP 的连接管理是自动的，用户和开发者通常不需要关心这些细节，但理解这个流程有助于排查问题和优化性能。

第 35 页

【幻灯片标题】MCP 与 Function Calling 对比

【逐字稿】

我们来对比 MCP 和 Function Calling，帮助大家选择。从四个维度来看：第一，架构层级——FC 轻量无部署成本，直接在代码里定义函数；MCP 引入中间层更规范，需要部署 MCP Server。第二，Token 开销——少量工具时 FC 更低，因为不需要协议开销；大批量工具时 MCP 更节约，因为按需加载。第三，多模型适配——单模型用 FC 更快，不需要协议转换；多模型用 MCP 减少重复开发，一次实现多模型通用。第四，适用场景——小项目优选 FC，简单直接；中大型工程优选 MCP，规范可扩展。总结：如果你的项目只用一个模型、几个工具，用 FC 就够了；如果需要多模型、大量工具、团队协作，MCP 是更好的选择。两者不是替代关系，而是不同规模下的最优选择。

第 36 页

【幻灯片标题】MCP 市场

【逐字稿】

我们来看 MCP 的市场生态。海外主要有两个平台：mcp.so 是全球存量最全的 MCP 市场，各种工具的 MCP Server 都能找到；smithery.ai 支持远程托管和本地部署，使用更方

便。国内也有两个平台：mcp-cn.com 聚焦国内 SaaS 服务的 MCP 适配，比如钉钉、飞书等；mcpmarket.cn 是中文 MCP 聚合市场，国内服务覆盖更广。MCP 市场的繁荣说明这个协议正在成为行业标准。在实际使用中，你可以直接从市场下载需要的 MCP Server，配置后即可使用，不需要自己开发。这大大降低了 MCP 的使用门槛。

第 37 页

【幻灯片标题】码道中的 MCP 服务

【逐字稿】

我们来看码道平台中的 MCP 服务。有四个特点：第一，开放易用——体验版即开放 MCP 能力，每月 500 万 Tokens 免费额度，门槛很低。第二，灵活扩展——支持本地和远程双模式，内置 MCP 市场，可以直接安装使用。第三，智能协同——多工具自动编排，一个提示词就能完成完整链路。比如你说「查数据库然后生成报表发邮件」，AI 自动编排数据库 MCP、报表 MCP、邮件 MCP 完成全流程。第四，标准统一——基于标准 MCP 开放协议，非厂商锁定，你可以用任何兼容 MCP 的工具。码道平台的 MCP 能力是实践 AI 编程的重要基础设施，建议大家课后亲自体验。

第 38 页

【幻灯片标题】目录

【逐字稿】

我们进入第五部分——Skills 概览。前面讲了 Prompt、RAG、MCP，现在讲 Skills。如果说 MCP 是给 AI 连接外部工具，Skills 就是给 AI 封装可复用的业务能力。一个 Skill

就像一个标准化的「工作技能」，AI 学会了就能反复使用。这部分会讲清楚：什么是 Skill、为什么用 Skill、怎么用 Skill、以及 Skill 和 MCP 的区别。

第 39 页

【幻灯片标题】什么是 Skill

【逐字稿】

什么是 Skill？Skill 是一个被标准化封装的能力单元，能像函数一样反复使用。一个 Skill 就是一个文件夹，通过按需、渐进式加载，实现高效且可复用的经验传递。一个 Skill 文件夹包含：SKILL.md——核心说明文件，描述这个 Skill 做什么、怎么用；scripts 目录——存放脚本文件；references 目录——存放参考资料；assets 目录——存放资源文件。Skill 的核心价值是「经验复用」——你把一次解决问题的经验封装成 Skill，下次遇到类似问题 AI 就能直接用，不需要从头来。比如你写了一个「代码审查」的 Skill，封装了审查标准、流程、输出格式，以后每次代码审查都自动按这个标准来。Skill 让 AI 的能力可以积累和传承，这是团队协作的重要基础。

第 40 页

【幻灯片标题】为什么要使用 Skill

【逐字稿】

为什么要使用 Skill？核心优势是 Skill 采取渐进式加载的三层架构。初始时仅加载元数据层及正文层中的 Name、Description 字段，减少 Token 消耗及上下文窗口。什么意思呢？AI 一开始只看 Skill 的名字和描述，判断当前任务需不需要这个 Skill。如果需要，才加载完整的 Skill 内容。如果不需要，就跳过不加载。这种按需加载机制大幅减少了 Token

消耗——假设你有 100 个 Skill，如果全量加载，每个 Skill 1000 个 Token，就是 10 万个 Token，上下文窗口可能就满了。但用渐进式加载，初始只加载 100 个 Skill 的名字和描述，可能只占 5000 个 Token，只有真正需要的 1-2 个 Skill 才全量加载。这让 AI 可以管理大量 Skill 而不会撑爆上下文。

第 41 页

【幻灯片标题】SKILL.md 的组成结构

【逐字稿】

我们来看 SKILL.md 的组成结构。SKILL.md 是每个 Skill 的核心文件，包含三部分：第一，name——技能标识符，命名原则是动词加名词，比如「review-code」「deploy-app」。第二，description——Agent 是否激活 Skill 判断的唯一依据。AI 看到任务描述后，跟所有 Skill 的 description 比对，决定激活哪个。所以 description 要写得准确、有区分度。第三，执行指令——Skill 激活后具体执行剧本，包含步骤说明、分析框架、输出格式要求。比如一个代码审查 Skill 的执行指令可能是：第一步检查代码规范，第二步检查安全漏洞，第三步检查性能问题，第四步输出审查报告。SKILL.md 的写法本质上是 Prompt Engineering——用结构化的方式告诉 AI 如何完成特定任务。

第 42 页

【幻灯片标题】Skill 与 MCP 的对比

【逐字稿】

我们来对比 Skill 和 MCP，帮助大家理解两者的定位差异。MCP 负责打通外部资源的连接通路——它解决的是「怎么连」的问题，比如连数据库、连 API、连文件系统。

MCP 是全量预加载上下文——连接后所有工具定义都加载进来。Skill 负责按需封装业务调用逻辑——它解决的是「怎么做」的问题，比如怎么做代码审查、怎么做部署。Skill 是惰性按需加载——用到时才加载，大幅压缩 Token 开销。简单说：MCP 是「管道」，负责连接；Skill 是「流程」，负责编排。MCP 告诉 AI「有哪些工具可用」，Skill 告诉 AI「按什么步骤使用这些工具」。两者互补——MCP 提供能力，Skill 提供方法。在实际项目中通常两者结合使用。

第 43 页

【幻灯片标题】什么时候使用 Skills

【逐字稿】

什么时候使用 Skills，什么时候使用 MCP？选择原则是：当需要降低 Token 消耗、面向非开发人员落地能力、按需动态调用工具、轻量化封装业务逻辑时，优先选用 Skills。反之，底层链路对接、标准协议互联场景用 MCP。具体来说：如果你要封装一个业务流程（如代码审查流程、部署流程），用 Skill——因为它可以定义步骤和规范。如果你要连接一个外部服务（如数据库、API），用 MCP——因为它提供标准化的连接协议。如果你要降低 Token 消耗，用 Skill——因为它是按需加载。如果你要多模型复用，用 MCP——因为它是标准协议。实际中很多场景两者结合——用 MCP 连接外部服务，用 Skill 编排业务流程。

第 44 页

【幻灯片标题】Skills 市场

【逐字稿】

我们来看 Skills 的市场生态。海外有两个主要平台：awesome-claude-skills 是领域权威精选，质量高但数量少；skillsmp.com 是全球海量开源 Skill，覆盖各种场景。国内也有两个：awesome-agent-skills 中文版，适合国内开发者；skillhub.cn 是中文整理全品类，分类清晰。Skills 市场正在快速发展，越来越多的最佳实践被封装成 Skill 分享出来。你可以直接从市场下载需要的 Skill，也可以把自己的经验封装成 Skill 分享给社区。这种经验共享机制让 AI 编程的能力可以积累和传承，是团队和社区提升 AI 编程效率的重要途径。

第 45 页

【幻灯片标题】常用 Skill 推荐

【逐字稿】

我们来看一些常用的 Skill 推荐。华为云 Skill——官方 Agent 技能平台，针对华为云场景优化。skill-creator——帮助安装 skill 的 skill，元技能，用它可以快速创建新 Skill。find-skill——一句话找到想要的 skill，在大量 Skill 中快速定位需要的。Anthropic 开源了 17 款 Skills——包括 office 四件套（Word、Excel、PPT、PDF 处理）和 MCP Builder 等。这些 Skill 可以直接安装使用，覆盖了常见的工作场景。建议大家从这些常用 Skill 开始体验，感受 Skill 如何提升 AI 编程效率。安装一个 Skill 后，AI 在遇到相关任务时会自动激活，不需要你手动指定。

第 46 页

【幻灯片标题】码道中的 Skill

【逐字稿】

我们来看码道平台中的 Skill 能力。有三个特点：第一，开放可扩展的 Skill 市场——可以安装社区 Skill，也可以自己开发 Skill。第二，项目级与个人级 Skill 的分层管理——项目级 Skill 对整个项目生效，个人级 Skill 只对自己生效，方便团队协作。第三，全流程质量守护链——包含检视、风格、检查、自测试、修复五个协作 Skill，覆盖代码质量的全流程。这套质量守护链是码道平台的特色能力，让 AI 编程不只是生成代码，还能保证代码质量。建议大家在实际项目中体验这套质量守护链，感受 Skill 如何系统化地提升代码质量。

第 47 页

【幻灯片标题】目录

【逐字稿】

我们进入第六部分——Code Agent 拓展介绍。前面讲了核心概念和主要工具，现在讲一些更高级的拓展概念。这部分会介绍 Rule、Hook、Cli、上下文工程、驾驭工程、Agent Team、循环工程等概念。这些概念在 Code Agent 的工程化实践中非常重要，理解它们可以帮助你更好地驾驭 AI 编程工具。

第 48 页

【幻灯片标题】什么是 Rule

【逐字稿】

什么是 Rule？在 Code Agent 中，可以通过自定义规则 Rule 对智能体行为进行约束，使其更准确地理解并严格遵循相关规范。Rule 就像给 AI 制定的「工作守则」——告诉它什么该做、什么不该做、怎么做。比如你可以定义 Rule：所有 Python 代码必须用 Black 格

式化；不允许使用 eval 函数；函数必须有类型注解。AI 在生成代码时会自动遵守这些规则。Rule 的价值在于把团队的编码规范自动执行——不需要人来检查，AI 自己就按规范来。Rule 通常定义在项目配置文件中，如 AGENTS.md，对整个项目生效。

第 49 页

【幻灯片标题】什么是 Hook

【逐字稿】

什么是 Hook？Hooks 钩子是 Agent 工程化中实现可观测性与可干预性的基础插件机制，作用于单点，实现功能扩展。Hooks 的执行是确定性的——在特定时机一定会触发。比如你可以定义 Hook：在 AI 生成代码后自动运行 lint 检查；在提交代码前自动运行测试；在修改文件后自动更新依赖。Hook 跟 Rule 的区别是：Rule 是约束 AI 的行为——告诉 AI 不要做什么；Hook 是在 AI 行为前后插入自动操作——AI 做完一件事后自动触发另一件事。Hook 让 Code Agent 的工作流程可以自动化串联，减少人工干预。Hooks 的确定性保证了流程的可靠性——每次都会执行，不会遗漏。

第 50 页

【幻灯片标题】什么是 Cli

【逐字稿】

什么是 Cli？CLI 本质上是一套标准化命令接口，核心作用让 AI 可以直接执行操作。命令更适合 AI 使用——因为命令是结构化的、可组合的、可编程的。Cli 的优势有四点：第一，更低开发门槛——不需要写复杂代码，一条命令搞定。第二，更高开发效率——命令组合可以快速完成复杂任务。第三，更优性能成本——命令执行比 API 调用更轻量。第

四，更完善治理体系——命令有权限控制、日志记录、审计追踪。Cli 让 AI 可以直接操作文件系统、运行程序、管理项目——这是 Code Agent 能自主完成开发任务的重要基础。比如 AI 可以自己运行 npm install、npm test、git commit 等命令，不需要人来执行。

第 51 页

【幻灯片标题】什么是上下文工程

【逐字稿】

什么是上下文工程 Context Engineering？Context Engineering 是系统性工程，负责在 Agent 执行任务的每一步，为其智能地组装出最优的上下文组合。前面讲上下文 Context 时提到过，上下文管理的目标是在有限 Token 内给模型高质量信息。Context Engineering 把这个目标系统化——它不只是简单地拼接信息，而是智能地决定每一步该给模型看什么信息。面临的挑战有七个：上下文窗口限制——空间有限；信息过载——信息太多反而干扰；多轮对话状态漂移——对话越长越容易跑偏；外部知识实时性——知识库可能过时；系统指令稳定性——系统指令不能被覆盖；上下文冲突——不同来源信息可能矛盾；语义压缩——需要压缩信息但保留语义。Context Engineering 是 Code Agent 的核心竞争力——同样的模型，上下文工程做得好的 Code Agent 表现远超做得差的。

第 52 页

【幻灯片标题】上下文的组成及分类

【逐字稿】

我们来看上下文的组成及分类。上下文分为三类：第一，引导型——包括系统指令（定义 AI 角色和行为）、任务说明（当前要做什么）、示例演示（参考样例）、输出规

范（输出格式要求）。引导型上下文告诉 AI「该怎么做事」。第二，知识型——包括 RAG 检索结果、短期记忆（当前对话历史）、长期记忆（数据库中的历史信息）、状态记录（当前项目状态）。知识型上下文告诉 AI「相关信息是什么」。第三，操作型——包括工具定义（有哪些工具可用）、工具调用记录（之前调过什么工具）。操作型上下文告诉 AI「能做什么操作」。这三类上下文合在一起，构成了 AI 在某一刻看到的完整信息世界。Context Engineering 就是智能地管理这三类信息的组合。

第 53 页

【幻灯片标题】码道中的上下文管理

【逐字稿】

我们来看码道平台中的上下文管理。有四个能力：第一，上下文引用——可以指定文件、目录、终端输出、Git 记录作为上下文。比如你可以让 AI 参考某个文件的内容来生成代码。第二，上下文精细化管理——提前加载缓存提升速度，压缩过大上下文节省 Token，按重要程度采用不同压缩策略。第三，上下文注入——自动读取 AGENTS.md 项目配置，注入项目规则和个人规则。第四，对话历史记录与记忆管理——保存对话历史，支持回溯和引用。这些能力让码道平台能高效管理上下文，在有限的 Token 窗口内给 AI 最优质的信息。这也是为什么同样的模型，在码道平台上的表现往往优于裸模型——因为上下文工程做得好。

第 54 页

【幻灯片标题】什么是驾驭工程

【逐字稿】

什么是驾驭工程 Harness Engineering？核心是系统性地解决「如何为 AI 构建一个能稳定运行、处理长周期复杂任务的可靠环境」。当任务复杂、周期长时，AI 可能在执行过程中跑偏、出错、陷入死循环。Harness Engineering 就是构建一个「框架」来驾驭 AI，保证它稳定可靠地运行。包含五个部分：引导层——通过 Prompt 和 Rule 引导 AI 行为；记忆与上下文管理——管理长周期任务的信息；工具与外部系统集成——提供可控的工具调用；安全护栏与沙箱——防止 AI 执行危险操作；反馈与验证闭环——自动验证 AI 的输出，发现问题自动修复。Harness Engineering 是 Code Agent 工程化的最高形态——它把 AI 从一个「对话工具」变成了一个「可靠的工程系统」。

第 55 页

【幻灯片标题】Agent Team 团队架构

【逐字稿】

我们来看 Agent Team 团队架构。Agent Team 是多个专业化 AI Agent 组成的协作型工作单元。有两种角色：Team Leader 是核心调度者，负责分解任务、分配工作、整合结果；Team Mate 是执行具体任务的专职 Agent，比如前端 Agent、后端 Agent、测试 Agent。Agent Team 跟单个 SubAgent 的区别在于：上下文管理——Team 中每个 Agent 有独立上下文，互不干扰；通信机制——Agent 之间可以通信协作；任务协调——Leader 统一调度，避免冲突；团队组建——可以根据任务动态组建团队；故障恢复——某个 Agent 失败时可以重新分配。Agent Team 让 Code Agent 能处理更复杂的任务——一个人做不到的，一个团队可以。这是 AI 编程工具向「AI 工程团队」演进的方向。

第 56 页

【幻灯片标题】什么是循环工程

【逐字稿】

什么是循环工程 Loop Engineering？核心是设计一套带上下文记忆、能自主持续闭环运行的自动化工作流。共五步：发现→分配→执行→验证→记录。系统自己发现任务（比如发现代码有 bug），分派给合适的 Agent 执行，执行后验证结果，记录过程供后续参考，然后继续发现下一个任务。循环工程让 AI 系统能「自运转」——不需要人持续盯着，系统自己发现问题、解决问题、优化自己。这跟传统的一次性执行不同——循环工程是持续运行的，像一个永不疲倦的工人。当然，循环工程对系统稳定性要求很高——每一步都必须可靠，否则循环会中断。这也是为什么前面讲的 Harness Engineering 很重要——它是循环工程的基础。

第 57 页

【幻灯片标题】核心概念对比

【逐字稿】

我们来做一个核心概念对比，用比喻帮助大家记忆。Prompt 是「客户需求订单」——灵活直接但精度要求高，写得好 AI 做得好，写得差 AI 做得差。RAG 是「工厂档案室」——抑制幻觉但依赖知识库质量，档案室里没有的找不到。Agent 是「工厂领班」——自主规划但推理要求高，能力强但需要好的指令。Context 是「需求沟通细节」——场景连贯但易信息过载，信息太多太少都不行。这四个概念是 Code Agent 的基础——Prompt 定义需求，RAG 提供知识，Agent 执行任务，Context 管理信息。理解了这四个比喻，就理解了 Code Agent 的核心架构。

第 58 页

【幻灯片标题】工具使用概念对比

【逐字稿】

我们来看工具使用概念的对比。Function Calling 是「调用车间专业设备」——拓展能力边界但参数要求严，每个工具需要精确定义。MCP 是「工厂总管线」——统一调度但配置复杂，一次配置多模型通用。Skill 是「工人熟练手艺」——专项能力强但调用门槛高，需要写好 SKILL.md。Rule 是「工厂规章制度」——约束精准但灵活性差，定义了就不能变。CLI 是「车间控制台」——操作高效但有使用门槛，直接执行命令。Hook 是「设备感应触发器」——自动化触发但排查难度高，出问题时不好调试。这六种工具各有定位，在实际 Code Agent 中组合使用——MCP 连接外部，Skill 封装流程，Rule 约束行为，CLI 执行操作，Hook 自动触发，Function Calling 做基础工具调用。

第 59 页

【幻灯片标题】工具理念对比

【逐字稿】

我们来看工具理念的对比，这是更高层面的概念对比。Prompt Engineering 是「订单优化专员」——门槛低但依赖优化经验，人人能用但用好需要练习。Context Engineering 是「工厂资料整理员」——解决长度限制但易丢失核心信息，需要智能筛选。Loop Engineering 是「闭环流水线设计」——自主迭代但 Token 消耗大，系统自运转但成本高。Harness Engineering 是「工厂整体基建和运维」——系统稳定但工程搭建复杂，一次搭建长期受益。这四种工程理念代表了 Code Agent 的不同成熟阶段——从 Prompt 到 Context 到

Loop 到 Harness，越来越高级，越来越复杂。在实际使用中，从 Prompt 开始，逐步引入更高级的工程理念，循序渐进。

第 60 页

【幻灯片标题】学习推荐

【逐字稿】

给大家一些学习推荐资源。华为云官网是 https://www.huaweicloud.com/，可以了解华为云的 Code Agent 产品和服务。华为云学堂是 https://edu.huaweicloud.com/，有丰富的 AI 编程课程。建议大家课后访问这些平台，动手实践本课程讲的内容。本章讲了很多概念，光听容易忘，最好边听边用——打开一个 AI 编程工具，试试写不同的 Prompt，体验 RAG 和 MCP 的效果，安装一个 Skill 用用看。概念只有在实践中才能真正理解。后续课程会有更多实操环节，带大家深入每个技术细节。好，本章内容就到这里，感谢大家的聆听！

第 61 页

【幻灯片标题】版权页

【逐字稿】

这是版权页。本资料版权归华为技术有限公司所有，保留所有权利。本资料是华为的保密信息，仅供华为授权的培训客户内部使用。请大家遵守保密规定，不要将课程资料外传。感谢大家的配合，我们下节课再见！