第 25 课：MCP 基础，重点版

一、MCP 是什么？

MCP 全称是 **Model Context Protocol**。

你可以先记一句面试表达：MCP 是一种连接大模型应用和外部工具、数据源、工作流的开放协议。它把“模型应用怎么发现工具、读取资源、调用外部系统”标准化了。

官方文档把 MCP 描述为连接 AI 应用和外部系统的开源标准，例如连接本地文件、数据库、搜索引擎、计算器和专业 prompt 工作流等。它常被类比成 AI 应用里的“USB-C 接口”：不是每个应用都为每个工具单独写一套适配，而是大家按统一协议连接。

用你的项目语言说：

没有 MCP：

ChatGPT / Claude / Cursor / 你的 Agent 想调用工单系统、知识库、数据库，
每个客户端都要单独写适配代码。

有 MCP：

你把“查政策”“建工单”“查工单”“读知识库资源”封装成 MCP Server，
不同 MCP Client 都可以通过统一协议发现和调用这些能力。

二、为什么需要 MCP？

你已经学过 Tool Calling。那为什么还要 MCP？

因为 **Tool Calling 解决的是模型输出工具调用格式的问题**，而 MCP 更偏向解决：

工具在哪里？
工具怎么被发现？
工具参数 schema 怎么暴露？
工具执行结果怎么返回？
资源怎么读取？
多个客户端怎么复用同一批工具？

你可以这样理解：

Tool Calling：模型说“我要调用 create_ticket 工具，参数是 xxx”
MCP：外部系统用标准协议告诉客户端“我有哪些工具、资源、prompt，你可以怎么调用”

所以 MCP 不是替代 Tool Calling，而是让 Tool Calling 背后的工具接入更标准。

三、MCP 的核心架构：Host / Client / Server

MCP 是典型的客户端-服务端架构。

最核心是三个角色：

Host：用户真正交互的 AI 应用
Client：Host 里面负责连接某个 MCP Server 的协议组件
Server：暴露工具、资源、Prompt 的外部能力服务

官方文档说明，Host 是用户实际使用的应用，比如 Claude、IDE 或聊天应用；Client 由 Host 创建，用来和某一个 MCP Server 建立直接通信；一个 Client 通常负责连接一个 Server。

你可以这样画：

用户
↓
Host：ChatGPT / Claude / Cursor / 你的 Agent UI
↓
MCP Client：负责协议通信
↓
MCP Server：知识库服务 / 工单服务 / GitHub 服务 / 文件系统服务
↓
真实系统：数据库 / API / 文件 / 搜索引擎 / Qdrant

放到你的项目里：

Host：

你的 Policy KB Assistant 前端 / Agent 应用

MCP Client：
Agent 里负责连接 MCP Server 的组件

MCP Server：
policy-kb-mcp-server

Server 暴露能力：
- search_policy
- answer_with_citation
- create_ticket
- get_ticket
- add_ticket_comment
- cancel_ticket

但是注意：**MCP Server 不等于让模型直接操作数据库。**
你原来的安全边界仍然成立：

LLM 只能提出调用意图
MCP Server 接收调用
后端 service 做权限、参数、状态、确认、审计
最后才真正执行

四、MCP Server 能暴露什么？

MCP Server 主要暴露三类能力：

Tools
Resources
Prompts

官方规范也把服务端能力分成这三类：Resources 提供上下文和数据，Prompts 提供模板化消息和工作流，Tools 提供模型可以执行的函数。

1. Tools：可执行动作

Tools 是最像你已经学过的 Tool Calling 的部分。
官方文档说，MCP Tools 让服务器暴露可被语言模型调用的函数，用来查询数据库、调用 API 或执行计算；每个 tool 有名称和描述参数 schema 的元数据。
你的项目里，Tools 可以是：

search_policy(query, user_id, tenant_id)
create_ticket(title, category, priority, description)
get_ticket(ticket_id)
add_ticket_comment(ticket_id, content)
urge_ticket(ticket_id)
cancel_ticket(ticket_id, confirm_token)

面试表达：在我的项目里，MCP Tool 可以对应后端已经存在的业务服务方法，但我不会让 MCP Tool 直接写数据库，而是复用 service 层，继续做 schema 校验、权限校验、资源状态校验、确认机制和审计日志。

2. Resources：可读取上下文

Resources 不是“执行动作”，而是“提供上下文数据”。
官方文档说明，Resources 用来让服务器向客户端共享可作为上下文的数据，例如文件、数据库 schema 或特定应用信息；每个 Resource 通过 URI 唯一标识。
你的项目里，Resources 可以是：

policy://documents/hr-leave-policy
policy://documents/finance-reimbursement
schema://ticket

schema://audit-log
kb://tenant/{tenant_id}/available-documents

它们适合做：

读取制度文档
读取某类工单 schema
读取可用知识库列表
读取某个租户可访问的文档目录

面试表达：Tool 是“做事”，Resource 是“给上下文”。比如 create_ticket 是 Tool，ticket_schema 是 Resource；search_policy 是 Tool，某个制度文档内容可以作为 Resource 暴露。

3. Prompts：可复用提示模板

Prompts 是服务器提供的可复用 prompt 模板。

官方文档说明，MCP Prompts 允许服务器暴露结构化提示模板，客户端可以发现、获取并传入参数来定制这些 prompt。

你的项目里，Prompts 可以是：

prompt://policy_qa
prompt://ticket_triage
prompt://safe_cancel_ticket
prompt://rag_answer_with_citation

比如：

ticket_triage_prompt:
你是 IT 服务台助手，请根据用户描述判断是否需要创建工单，
如果缺少 category / priority / description，则追问缺失字段。

面试表达：Prompt 不是写死在某个客户端里，而是由 MCP Server 标准化暴露。这样不同 Agent 客户端都可以复用同一套企业工单处理模板。

五、MCP 和你已学内容的区别

1. MCP vs Tool Calling

Tool Calling：
模型输出结构化工具调用。

MCP：
外部系统用标准协议暴露工具、资源和 Prompt，让客户端可以发现和调用。

一句话：

Tool Calling 关注“模型怎么调用工具”，MCP 关注“工具怎么标准化接入模型应用”。

---

2. MCP vs LangChain

LangChain：
LLM 应用开发框架，帮你管理模型、prompt、retriever、chain、tool、agent。

MCP：
工具和资源接入协议，解决外部能力如何标准化暴露给 AI 应用。

一句话：

LangChain 更像应用开发框架，MCP 更像外部工具和数据源的连接协议。

---

3. MCP vs LangGraph

LangGraph：
负责 Agent 流程编排，比如 State、Node、Edge、Checkpoint、Human-in-the-loop。

MCP：
负责外部工具、资源、Prompt 的标准化暴露和调用。

一句话：

LangGraph 解决 Agent 内部流程怎么走，MCP 解决 Agent 外部工具怎么接。

放到你的项目：

LangGraph：

Memory → Planner → Validate → RAG → Draft → Confirm → Execute → Audit

MCP：

Execute Node 里面调用 search_policy / create_ticket / get_ticket 等外部工具

六、MCP 不是安全万能药

这一点很重要，因为你的项目亮点就是安全和审计。

MCP 让工具接入更标准，但不代表工具调用自动安全。官方安全文档也强调，MCP 带来强大的数据访问和代码执行能力，因此实现方必须认真处理安全和信任问题。

所以你要坚持原来的设计：

MCP 只解决工具接入标准化

安全仍然要靠业务系统实现

你的安全边界应该是：

1. 工具白名单
2. input schema 校验
3. 当前用户身份解析
4. tenant_id / user_id 权限过滤
5. 资源状态校验
6. 高风险动作二次确认
7. pending_action 恢复
8. audit_log 记录
9. trace_id 全链路追踪

面试表达：我不会因为用了 MCP 就让模型直接控制数据库。MCP Server 后面仍然是受控的 service 层，每个工具调用都要经过权限、参数、状态、确认和审计校验。

七、把 MCP 放进你的项目，怎么讲？

你可以这样讲：我的项目目前已经有 Agent ToolPlan 和后端 service 层。

如果引入 MCP，我会把知识库查询、工单创建、工单查询、评论、催办、取消等能力封装成 MCP Server 暴露出来。

这样不只是我的 Web Agent 能用，未来 Cursor、Claude Desktop、企业内部 Agent 平台也可以通过 MCP Client 接入同一套工具。

但 MCP Server 内部不会绕过原有业务逻辑，而是复用 service 层继续做权限校验、状态校验、高风险确认和审计日志。

八、本节你先记住 5 句话

1. MCP 是连接 AI 应用和外部工具、资源、Prompt 的开放协议。

2. Tool Calling 解决模型如何发起工具调用，MCP 解决工具如何标准化暴露给模型应用。

3. Host 是用户交互的 AI 应用，Client 负责协议连接，Server 暴露工具、资源和 Prompt。

4. Tools 是可执行动作，Resources 是可读取上下文，Prompts 是可复用提示模板。

5. MCP 不替代权限、确认和审计；企业项目里 MCP Server 后面仍然要接受控的 service 层。

第 26 课：MCP Server / Client / Tools / Resources / Prompts 项目化深入

一、先给你一句总表达

在我的项目里，MCP 主要承担工具执行层的标准化封装。LangGraph 负责 Agent 流程编排和工具调用计划，MCP Server 对外暴露知识库问答、工单创建、草稿续办、确认动作、工单工具规划和工单详情查询等工具；MCP Wrapper 则统一做工具名映射、参数过滤、身份防伪、幂等、并发控制、错误归一化和统一结果封装。最终所有工具仍然回到后端 service 层执行，不会让模型直接操作数据库。

二、你项目里的 MCP 分成 3 层

你现在不是只有一个 MCP Server，而是可以拆成三层理解。

1. MCP Server：对外暴露工具入口

你的项目里有两个入口：

- src/mcp_stdio_server.py
- src/mcp_http_server.py

它们暴露的工具包括：

- ask_policy
- create_ticket
- continue_ticket_draft
- confirm_action
- ticket_tool_planner
- get_ticket_detail

可以这么理解：

MCP Server 是给外部 MCP Client 或 MCP Host 调用的工具入口。

比如 Claude Desktop、Cursor、企业内部 Agent 平台，未来都可以通过 MCP 协议调用这些工具。

这里最重要的是：**MCP Server 不是业务逻辑本身，它只是入口。**

真正业务逻辑还是在你的 services 里。

2. MCP Wrapper：统一工具执行契约

核心在：

- src/mcp_wrapper/registry.py
- src/mcp_wrapper/executor.py

这一层才是你项目最有价值的地方。

它做了几件事：

- 工具名映射
- 参数过滤
- 禁止伪造 actor / user_id / role / session
- 幂等控制
- 并发闸门
- 错误归一化
- 统一 ToolCallResult 返回

所以 MCP Wrapper 的本质是：

把内部业务能力包装成一套安全、统一、可治理的工具调用协议。

这句话非常重要。

它比单纯“我会写 MCP Server”更有项目含金量。

3. Services：真正执行业务

最后，MCP Wrapper 会通过 registry 分发到真实业务函数：

- ask_policy -> _handle_kb_intent
- create_ticket -> _handle_create_ticket_intent
- continue_ticket_draft -> _resume_ticket_draft_workflow
- ticket_tool_planner -> _handle_ticket_tool_route_with_planner
- confirm_action -> _handle_confirmed_pending_action

get_ticket_detail -> invoke_ticket_tool

也就是说：

MCP Server
↓
MCP Wrapper
↓
Registry
↓
Services
↓
PostgreSQL / Qdrant / Redis

所以你项目里 MCP 的安全边界是清楚的：

MCP 不直接碰数据库，MCP 最终还是调用受控 service 层。

三、你的项目是“双轨制”

这一点必须讲清楚。

你项目里不是所有 LangGraph 分支都走 MCP。

当前是两条路径。

路径一：规则 / Legacy 分支，不走 MCP

比如：

nodes_ask.py
nodes_ticket_create.py
nodes_ticket_tool.py
nodes_confirm.py

这些节点大致是：

LangGraph node
↓
adapters
↓
services

也就是：

Agent 流程节点直接调用业务 service

这条路径适合规则路由、已有稳定流程、低改造成本的场景。

路径二：LLM Global Planner 分支，走 MCP

这条路径是：

prepare
↓
load_memory
↓
resolve_references
↓
global_plan
↓
global_validate
↓
execute_mcp_tool
↓
finalize

关键点是：

global_plan 负责规划

global_validate 负责校验并生成统一 tool_request
execute_mcp_tool 负责执行工具

如果开启远程 MCP:

LangGraph
↓
call_remote_tool
↓
MCP HTTP Server
↓
MCP Wrapper
↓
Services

如果没有开启远程 MCP:

LangGraph
↓
invoke_tool
↓
MCP Wrapper
↓
Services

所以你可以这样总结：我的项目目前是双轨制。规则路由分支直接走 adapters 到 services；LLM Global Planner 分支会先生成统一 tool_request，再通过 execute_mcp_tool 进入 MCP wrapper。远程模式下会通过 MCP HTTP server 调用，本地模式下则直接调用 wrapper。这样既保留了原有稳定流程，也为未来工具远程化和多客户端接入提供了扩展能力。

四、MCP 使用流程，简化版

面试里不要陷入具体参数，可以讲这条主链路。

本地 MCP wrapper 模式

用户输入
↓
LangGraph 规划工具调用
↓
global_validate 校验工具名和参数
↓
execute_mcp_tool
↓
invoke_tool
↓
MCP Wrapper 统一治理
↓
Registry 分发
↓
Services 执行业务
↓
统一 ToolCallResult
↓
写回 LangGraph state
↓
finalize 输出给用户

一句话解释：本地模式下，LangGraph 仍然负责流程，MCP Wrapper 负责把工具调用统一治理后分发到业务

service。

远程 MCP HTTP 模式

用户输入
↓
LangGraph 生成 tool_request
↓
execute_mcp_tool
↓
call_remote_tool
↓
MCP Client 连接 MCP HTTP Server
↓
MCP Server 接收 tool call
↓
MCP Wrapper 做统一治理（工具名校验，参数校验，Redis 幂等查询）
↓
Registry 分发到 Services
↓
Services 执行业务
↓
MCP Server 返回统一结果
↓
MCP Client 解析结果
↓
写回 LangGraph state
↓
finalize 输出

一句话解释：远程模式下，MCP 的价值是把工具执行从本地函数调用升级成标准 MCP 协议调用，使这些工具可以被外部 MCP Host 复用。

五、call_remote_tool 到底做什么？

你可以把 call_remote_tool(...) 理解成：它不是让模型直接调用工具，而是把 LangGraph 已经校验过的工具请求，通过 MCP 协议发给远程 MCP Server，然后接收统一结果。

简化成 7 步：

第一步：LangGraph 已经完成规划和校验

在进入 call_remote_tool(...) 之前，流程已经不是“模型随便说调用什么工具”。

前面已经经过：

global_plan
↓
global_validate

也就是说，系统已经得到一个比较规范的工具请求：

调用哪个工具
带什么业务参数
当前请求 id 是什么
当前模式是什么

面试表达：远程 MCP 调用不是直接相信模型输出，而是基于 LangGraph validate 后的 tool_request 执行。

第二步：检查远程 MCP 是否可用

call_remote_tool(...) 会先判断远程 MCP 是否处于熔断状态。

如果远程服务连续失败，系统就不会一直重试打爆远程服务，而是快速返回失败。

你可以简化说：

远程 MCP 调用前会做可用性和熔断检查，避免远程服务异常时拖垮 Agent 流程。
不用展开具体阈值。
第三步：把内部工具名映射成远程 MCP 工具名
你的项目内部可能叫：

kb_answer
lookup_ticket

远程 MCP Server 暴露的是：

ask_policy
get_ticket_detail

所以 client 侧会做一次工具名映射。
简化理解：

内部工具名
↓
远程 MCP Server 可识别的工具名

这一步的意义是：内部 Agent 的工具命名和外部 MCP Server 的工具命名可以解耦。
第四步：通过 MCP Client 调用 MCP HTTP Server
这一段就是标准远程 MCP 调用：

MCP Client
↓
initialize
↓
call_tool
↓
MCP HTTP Server

你不用在面试里讲具体协议对象，只要说：
远程模式下，execute_mcp_tool 会通过 MCP Client 连接 MCP HTTP Server，并调用对应的 MCP Tool。
第五步：MCP Server 收到请求后进入 Wrapper
MCP Server 收到工具调用后，不直接执行业务，而是进入：

invoke_tool

也就是 MCP Wrapper。
这一步非常关键。
因为 Wrapper 会做统一治理：

工具名规范化
参数过滤
禁止身份伪造
幂等控制
并发控制
错误归一化
统一结果格式

面试表达：
MCP Server 只是入口，真正的工具治理集中在 MCP Wrapper。这样本地调用和远程调用都可以复用同一套执行规则。
这句话很加分。
第六步：Registry 分发到 Services
Wrapper 治理完成后，通过 registry 找到对应业务函数。
例如：

ask_policy
↓
_handle_kb_intent

create_ticket
↓
_handle_create_ticket_intent

confirm_action
↓
_handle_confirmed_pending_action

也就是说，最后还是：

MCP Tool
↓
业务 service
↓
数据库 / 向量库 / Redis

这说明你的 MCP 没有绕过业务层。

第七步：统一结果写回 LangGraph

业务执行完成后，会返回统一结果。

然后：

MCP Server 返回
↓
MCP Client 解析
↓
ToolCallResult
↓
state["execution"]
↓
finalize

最后 LangGraph 根据执行结果组织自然语言回复给用户。

六、你项目 MCP 的核心价值是什么？

不要只说“我用了 MCP”。

你要说出它解决了什么问题。

价值 1：工具执行标准化

原来不同节点可能直接调不同 service。

现在 Global Planner 分支可以统一成：

tool_request
↓
ToolCallRequest
↓
ToolCallResult

好处是：工具调用输入输出更统一，方便评测、审计、回归测试和远程化扩展。

价值 2：统一治理边界

Wrapper 统一处理：

工具白名单
参数过滤
身份防伪
幂等
并发
错误归一化

这比每个工具自己散落处理更稳定。

面试表达：

MCP Wrapper 相当于工具执行的治理层，保证不同工具调用都遵守同一套安全和工程约束。

价值 3：远程化能力

开启远程 MCP 后：

LangGraph
↓
MCP HTTP Server
↓
Services

未来外部工具也能接入：

Claude Desktop
Cursor
企业内部 Agent 平台
其他 Web Agent

这就是 MCP 的扩展价值。

面试表达：

通过 MCP HTTP Server，我的工具不再只服务于当前 Web Agent，而是可以作为标准 MCP 工具被其他 Agent Client 复用。

价值 4：保留原有业务安全

你最重要的项目亮点是：

权限校验
确认机制
pending_action
审计日志
trace_id

MCP 没有替代这些东西。

它只是把调用入口标准化。

面试表达：MCP 只解决工具接入和调用协议，不替代业务安全。我的实现里，所有 MCP 调用最终仍然经过 service 层，保留原来的权限、状态、确认和审计机制。

最终面试版表达

我的项目里 MCP 不是替代 LangGraph，而是作为工具执行层的标准化封装。LangGraph 的 Global Planner 分支会先根据用户输入生成工具调用计划，经过 global_validate 做工具名和参数校验后，进入 execute_mcp_tool。

如果是本地模式，就直接调用 MCP Wrapper 的 invoke_tool；如果开启远程模式，就通过 call_remote_tool 调用 MCP HTTP Server。无论本地还是远程，最终都会进入同一套 MCP Wrapper。Wrapper 会做工具名规范化、参数过滤、禁止伪造用户身份、幂等控制、并发控制、错误归一化和统一 ToolCallResult 封装，然后通过 registry 分发到真实 services。

所以 MCP 在这个项目里的价值，是把知识库问答、工单创建、草稿续办、确认动作、工单详情查询等内部业务能力包装成可治理、可远程调用、可被多客户端复用的工具接口，同时不会绕过原有的权限校验、状态校验、二次确认和审计日志。

九、你现在该怎么理解这几个组件的关系？

LangGraph

负责：

流程编排
状态管理
记忆加载
引用解析
Planner
Validate

Execute
Finalize
Human-in-the-loop
Checkpoint

一句话：LangGraph 决定 Agent 流程怎么走。

MCP Server

负责：

对外暴露工具
接收 MCP Client 调用
把调用转交给 MCP Wrapper

一句话：MCP Server 是标准化工具入口。

MCP Wrapper

负责：

工具执行治理
统一参数处理
统一安全检查
统一错误格式
统一返回结果

一句话：MCP Wrapper 是工具调用治理层。

Registry

负责：

工具名 -> 业务函数

一句话：Registry 是工具路由表。

Services

负责：

真正业务逻辑
数据库操作
向量库检索
工单状态变更
审计记录

一句话：Services 是真正执行业务的地方。

十、用一个“取消上一张工单”串起来

用户说：

取消上一张工单

真实流程可以简化为：

用户输入
↓
LangGraph 加载记忆
↓
resolve_references 解析“上一张工单”
↓
global_plan 判断是取消工单
↓
global_validate 校验工具和参数
↓
execute_mcp_tool
↓
本地 invoke_tool 或远程 call_remote_tool
↓

MCP Wrapper 过滤参数、防止身份伪造、处理幂等

↓

Registry 分发到 confirm_action / ticket_tool_planner 相关业务函数

↓

Service 检查工单是否存在、是否属于当前用户、当前状态是否允许取消

↓

如果需要确认，创建 pending_action

↓

用户确认后再次进入 confirm_action

↓

恢复 pending_action

↓

再次校验权限和状态

↓

执行取消

↓

写 audit_log

↓

统一结果返回 LangGraph

↓

finalize 回复用户

十一、本课你必须记住 6 句话

1. 我项目里的 **MCP 不是 Agent 规划框架**，而是**工具执行层的标准化封装**。
2. **LangGraph 负责流程编排，MCP Wrapper 负责工具调用治理。**
3. **MCP Server 对外暴露工具，但不直接处理业务，最终还是调用 service 层。**
4. **本地模式是 LangGraph → invoke_tool → MCP Wrapper → Services。**
5. **远程模式是 LangGraph → call_remote_tool → MCP HTTP Server → MCP Wrapper → Services。**
6. **MCP 的价值是工具标准化、远程化、多客户端复用，同时保留权限、确认、幂等和审计。**