阿里云

AI 原生应用架构白皮书

AI NATIVE APPLICATION ARCHITECTURE WHITE PAPER

2025

参编人员

按章节顺序排序：

作者：彦林、亦盏、穆飞、麻芃、望宸、杨涛、小取、昔比、陆龟、席翁、翼严、濯光、十眠、如漫、青瑭、佳皓、文昀、严研、叶仔、梧同、聪言、如葜、计缘、澄潭、洵沐、不铭、不瞋、世如、柳下、千风、梅茜、音速、砥行、曼红、墨陋、姬风、涯海、望陶、觉澄、义泊、卓昂、西杰、马云雷、谷奈、黑屏、君扬、虚正、末那、鸣智、渭龙、浴血、赢止、汐雨、良玖、灵闻

设计：小取、师文涛、李志

校对：亦盏、望宸、望陶、世如、黑屏

推荐序

《AI 原生应用架构白皮书》获得了来自 AI 科研、AI 运用、AI 供应和 AI 媒体&社区等领域的企业或个人的推荐（以下按领域、以及推荐人的姓氏首字母排序），我们在此表示衷心的感谢！

当下，人工智能正在成为推动社会进步与产业升级的核心驱动力。在这股浪潮中，AI 原生应用的兴起不仅仅是技术的演进，更是软件开发范式与产业逻辑的根本性重塑。《AI 原生应用架构白皮书》正是对这一趋势的系统性回应。它以全栈视角梳理了 AI 应用从开发、测试到部署与运维的完整生命周期，既深入揭示了模型、数据与场景的核心价值，也全面呈现了云智一体、上下文工程、多智能体协作等关键技术路径的落地实践。本书结合行业一线经验，提出了具有操作性的架构思路与工程方法，为开发者和企业提供了一套系统化的参考框架。本书不仅是了解 AI 原生应用的窗口，更是勾勒了未来技术与产业深度融合路径的蓝图。

——中国科学院软件研究所研究员、博士生导师，黄涛

作为长期致力于人工智能与软件工程研究与实践的学者，我深切体会到技术范式迁移对产业与社会的深远影响。本白皮书以系统化的视角，对 AI 原生应用的全生命周期进行了全面剖析，不仅揭示了大模型驱动下应用架构的核心特征，还提出了涵盖框架设计、上下文工程、系统集成、评估方法、安全机制、运行时管理与可观测性等关键方法论。其价值不仅在于为企业实践提供可操作的路径指引，更在于促进学界与业界形成共识，加速 AI 从技术突破走向规模化应用与产业落地。对于研究者而言，本白皮书是洞察新范式的重要窗口；对于开发者与企业而言，它是探索未来实践的行动指南。我诚挚推荐，并期待它能为 AI 原生应用的体系化发展贡献持久力量。

——浙江大学软件学院副教授，博士生导师，倪超

我们正处在一个激动人心的时代。AI 正在引发一场技能的大规模通胀，“技术品味”却成为了越来越稀缺的通货。

在阿里云，我们自身的数字化实践覆盖了翻译、智能外呼、合同风险审核等众多领域。在交付实践中，我们提炼出了企业 AI 数字化实现“Result as a Service (RaaS)”的方法论（RIDE）——Reorganize (重组组织与生产关系)、Identify (识别业务痛点与AI机会)、Define (定义指标与运营体系)、和Execute (推进数据建设与工程落地)。

RIDE 的每一步都至关重要，我们看到太多企业在识别了机会后，却在工程落地时步履维艰。我认为，当前企业 AI 转型的主要矛盾，已经演变为企业 CEO 或业务部门追求社交媒体上‘炸裂’效果”，与“IT 部门 AI 交付能力发展不均衡、不充分”之间的矛盾。AI 应用如何从原型走向生产？如何应对幻觉、延迟、安全与成本等一系列系统性挑战？

我欣喜地看到，《AI 原生应用架构白皮书》提供了翔实且体系化的工程蓝图。它以“云智一体”为基石，系统性地回应了企业在大模型落地过程中最关切的几大问题：输出效果、性能、稳定性、安全与成本。尤其值得称道的是，它直面了 AI 应用“品味”的度量难题。本轮 AI 浪潮的关键区别在于评测没有标准答案，而这本白皮书对 AI 评估体系（第九章）的深入剖析，为我们如何量化“好坏”、定义反馈闭环、驱动数据飞轮提供了宝贵的指引。

然而，一本优秀的技术白皮书，其价值远不止于工程。我常说“书同文，车同轨”，成功的 AI 转型必须重建组织内部的 AI 语境与认知基础，正如我要求全员通过阿里云的 AI 大模型 ACA/ACP 认证。这本白皮书以其系统性与全面性，为开发者、架构师、乃至业务决策者提供了一套共同的语言框架，有助于打破认知壁垒，实现高效协同。我诚挚地将其推荐给每一位走在云智一体道路上的开拓者。

——阿里云 CIO，蒋林泉

AI 重塑万物的时代已悄然而至，AI 原生应用正深刻变革软件开发与运维范式，其全生命周期的架构设计、技术选型、工程实践及运维优化均面临独特挑战。《AI 原生应用架构白皮书》应运而生，该书系统阐述了从设计到运维各环节的关键要素，兼具理论深度与实践洞见。它内容精炼、脉络清晰，是开发者、架构师、运维工程师及技术决策者理解并构建下一代AI应用的宝贵指南。

——夸克工程技术负责人，万明成

我们正身处 AI 技术革命的浪潮中，大模型的能力持续提升，不但能够解决越来越难的数学问题，而且能够写出越来越复杂的代码。如何将大模型应用到更多的领域，创造更多的 AI 应用来改变我们的生活，给老百姓带来更多的福祉，是我们正在面对的另一个机遇和挑战。将 AI 应用的过程中，我们需要更多同路人一起探索，一起分享。在这个时间点，我们迎来了《AI 原生应用架构白皮书》发布，这本书为 AI 应用探索做了阶段性总结。路漫漫，其修远兮，让我们一起加油，一起去用 AI 创造更美好的未来。

——爱橙科技智能引擎算法平台负责人，王家忙

大模型正催生一个全新的行业应用的开发范式，如何从 AI Plus 到 Native AI，不止是一个产品和业务问题，从数据处理、模型集成与迭代到工程运维，是每个开发者需要面向下一代的生成式应用深刻思考并拥抱变化的命题。《AI 原生应用架构白皮书》给原生应用的技术架构设计、运维和迭代提供了一个系统、通用、可操作可实践的框架层面的指导，希望有更多的开发者能看到其中优秀的方法论和案例。

——阿里健康 CTO，王祥志

云智一体时代已至，AI 原生应用正在重塑软件开发的底层逻辑。从传统的“构成式”架构向“生成式”架构转变，从基于确定性逻辑的编程转向自然语言交互，这场技术革命对每一位工程师都意味着机遇与挑战并存。

做为 AI Coding 领域的从业者，我深刻感受到 AI 正在改变代码的创作方式 —— 从被动的工具调用转向主动的智能协作。这本《AI 原生应用架构白皮书》恰逢其时，系统梳理了从架构设计到运维优化的完整 DevOps 生命周期，为工程师们提供了宝贵的实践指南。

特别值得称道的是，本书不仅涵盖了 AI 网关、运行时、可观测等技术组件，更重要的是提供了"云智一体"视角下的架构思维框架。在 AI 应用开发面临黑盒特性、幻觉问题等挑战时，这样的系统性指导显得尤为珍贵。期望这本白皮书能够成为每一位 AI 时代工程师的案头必备，助力大家在这场技术变革中抓住先机，构建真正具有竞争力的 AI 原生应用。

——Qoder 资深技术专家， 谢吉宝

AI 原生应用架构白皮书

AI 技术浪潮奔涌向前，大模型正从技术突破迈向产业深耕。《AI 原生应用架构白皮书》紧扣这一时代命题，系统拆解 AI 原生应用从开发到运维的全生命周期：从模型、数据与场景的核心要素，到 DevOps 全流程的落地实践；从智能体开发范式、上下文工程优化，到网关集成、安全防护与效能评估，为读者呈现了一套清晰的方法论与工具箱。无论是探索 AI 落地的企业，还是深耕技术的开发者，均可从中获取关键洞察，助力 AI 真正成为驱动产业升级的核心引擎。诚荐一读！

——MCP.so，《这就是 MCP》作者，艾逗笔

大模型正引领一场深刻的技术革命，AI应用替代传统软件已成大势所趋，一个全新的软件“大航海时代”已然开启。大模型以其近乎人类的对话式交互，将吸引远超以往的开发者和创造者投身其中。在这波澜壮阔的浪潮中，这本白皮书系统性地拆解了 AI 原生应用的技术与架构，它将如同一座灯塔，为所有已经或即将启航的航海家们指明方向，照亮前行的道路。

——MuleRun CTO， 束骏亮

通用人工智能时代，应用架构的范式正在被重塑。“云智一体”不仅是技术理念，更是产业未来的基石。此白皮书首次系统性地定义了 AI 原生应用架构，其深度与广度令人叹服。它为所有架构师和技术决策者，提供了全新的航海图。

——Cherry Studio CEO，王新铭

大模型降低了 AI 原生应用的开发门槛，却也带来了架构、成本与运维方面的全新挑战。《AI原生应用架构白皮书》系统拆解了 AI 应用全生命周期的核心要素，从上下文工程、Agent 开发、工具集成、安全治理到成本优化，直击开发痛点与运维难点，为开发者高效构建可靠、落地的 AI 原生应用提供了清晰的“导航图”。硅基流动通过构建高效、易用的 MaaS 平台让 AI 应用落地更简单，与白皮书理念高度契合。我们期待业内携手探索，让智能真正流动起来。

——硅基流动联合创始人，杨攀

作为 AI 开源社区"通往 AGI 之路"发起人，我荣幸推荐这本《AI 原生应用架构白皮书》。

当 ChatGPT 开启 AGI 时代，AI 原生应用架构范式仍待明确。这本书以"云智一体"为底座，围绕全生命周期，从架构设计到工程实践，涵盖智能体开发、上下文工程等前沿议题，辅以携程等实践案例，为开发者提供从0到1的"施工图"。愿它成为AI浪潮中的"航海图"，助力创新应用落地。

——通往 AGI 之路发起人，🌈AJ

AI 当下其实可以说还是在非常早期，处于百废待兴的状态，各种基础设施和基本定义都还不健全，这个时候有一本讲解 AI 应用开发范式的书籍是非常有价值的。大家可以看到本书覆盖的范围非常广，基本涵盖了各个流程的基础知识，所以你如果是一个对 AI 行业和开发感兴趣的人，本书会非常适合你。

——42章经创始人，曲凯

《AI 原生应用架构白皮书》以清晰的架构视角和工程实践，为每一位 AI 开发者提供了系统化的成长路径。该白皮书从开发、调试到部署运维，全程贴合实际学习与构建需求，帮助开发者降低认知门槛、提升实践效率。我们始终相信，以学习者为中心的技术普惠，是推动 AI 真正落地的关键。

——Datawhale

魔搭 ModelScope 作为开源社区，见证了开源模型从'可用'到'好用'的跃迁。我们深知，AI原生应用的突破，不仅依赖单点模型能力，更需架构层面的协同进化。本白皮书从真实落地场景出发，整合模型服务、工具链协同、AI DevOps 等核心要素，为开源生态中的每一位创造者提供了可复用、可演进的AI原生架构体系。我们坚信：未来的 AI，生于开源，长于协同，成于架构。这本白皮书，正是这场进化的重要路标。

——魔搭社区（ModelScope）

前言

云智一体，碳硅共生

通用人工智能（AGI）已是确定的事情，我们正通往超级人工智能（ASI）。

过去三年，人工智能技术正以前所未有的速度渗透千行百业。国务院日前也印发了《关于深入实施“人工智能+”行动的意见》，从顶层设计的高度为人工智能技术落地提供了关键指引。这既展现出重塑生产力的巨大潜力，也孕育着重构生产关系的无限可能，为全球数字经济的智能化升级注入全新动能。

为持续突破 AI 性能边界，大模型厂商通过技术路径迭代不断拓展三个核心维度：在模型参数层面，从千亿参数，逐步演进至万亿级参数规模，实现对复杂知识的深度拟合；在训练数据层面，从数百 GB 文本，拓展至数百 TB 甚至 EB 级的多模态数据集，且数据质量与领域适配性持续提升；在算力支撑层面，核心训练算力需求呈现指数级增长，其规模每2年增长约10倍（即“黄氏定律”），为模型性能突破提供了有力保障。随着模型的推理能力和多模态能力显著提升，这些技术进步不仅为通用人工智能的实现减少了障碍，也为产业智能化升级提供了核心驱动力。

随着大模型厂商将训练与使用成本压缩至原有水平的几十分之一，AI 应用开始跨越效果与成本的平衡临界点。与此同时，AI 原生应用开发范式逐步形成雏形，从模型调用到场景适配的开发逻辑日渐清晰，为 AI 应用的深度探索奠定了坚实基础。自此，AI 正式进入规模化应用的爆发阶段。数据显示，过去16个月内全球对 AI Agent（智能体）的关注热度增长达1088%，AI 办公助手、数字员工、智能客服等应用如雨后春笋般涌现。这

其中，以 Agentic AI 为核心的技术路径逐渐成为主流，其通过自主规划、任务拆解与动态交互能力，推动 AI 从工具化应用向自主化服务演进，加速实现对数字世界的智能重塑与高效接管。

随着大模型与感知、控制技术的深度融合，具身智能正从实验室走向产业实践。从工厂的智能协作机器人到家庭服务终端，其发展依托于数字空间的智能能力向物理世界的延伸。Physical AI 作为这一进程的前沿方向，正推动 AI 从数据驱动的数字决策，逐步拓展至对实体环境的感知、规划与执行，进而实现对物理世界的智能化赋能与协同。

可见，大模型已完成从技术突破到产业应用的关键跨越，AI 正深度融入并重塑数字世界，并持续向物理世界延伸，最终推动人类生产生活方式的根本性变革。

在这一进程中，云计算以“云智一体”的形态，成为连接数字与物理世界的核心底座。极致弹性的算力资源、秒级伸缩的推理服务、跨“云–边–端”的统一调度框架，以及面向 AI DevOps 的全生命周期工具链，使得应用的训练、推理和运维像水电一样随取随用；云原生安全、成本治理与多租户隔离，为企业级 AI 应用提供了可信赖的运行环境；开放的模型即服务（MaaS）生态，让任何组织都能以最低门槛接入前沿智能。云不再只是简单的资源池化，而是与智能算法融为一体，成为 AI 能力不可替代的技术平台，让智能在数字世界和物理世界之间自由流动，实现真正的“碳硅共生”。

为什么写这本白皮书

AI 应用已初步跨越规模化落地的门槛，但 AI 原生应用的架构范式仍待明确。不同于传统软件开发通过编程与算法构建的确定性逻辑，AI 时代的应用构建以面对自然语言编程、上下文工程为核心特征，将复杂业务逻辑与决策过程下沉至模型推理环节，从而实现业务的智能化自适应。

构建有竞争力的 AI 应用，需聚焦三个核心要素：模型、数据与场景。对多数企业而言，AI 应用的本质是通过深度挖掘私域数据，解决核心场景的效率瓶颈，进而实现快速创新与增长突破。私域数据的价值挖掘至关重要，而高质量数据的构建需依托系统化能力。AI 应用架构的核心，正在于为客户打造完整的数据飞轮，实现私域数据的持续沉淀、行业数据的动态演进、评估数据的量化闭环与反馈数据的迭代循环，最终提供稳定可靠的 AI 服务。

然而，AI 应用开发过程中仍面临诸多挑战，例如开发阶段强依赖模型黑盒特性，导致结果可控性不足、幻觉问题频发，从原型验证（PoC）到生产部署往往需要数月调优，核心痛点集中在调试效率与业务适配；上线后则面临推理延迟、稳定性波动、问题排查困难、安全风险凸显、输出不可靠及成本过高等问题，折射出企业级 AI 应用在稳定性、性能、安全与成本控制上的系统性挑战。这些问题的解决，亟需"云智一体"的全栈能力支撑，包括模型训练、应用构建和逻辑编排、实时推理加速，以及基于 AIOps 的智能运维体系等。针对此，本白皮书将围绕 AI 原生应用的 DevOps 全生命周期，从架构设计、技术选型、工程实践到运维优化，对概念和重难点进行系统的拆解，并尝试提供一些解题思路。

这本白皮书主要讲什么

本白皮书的出发点是帮助读者系统、全面地了解 AI 原生应用的开发、测试、上线和运维的完整应用生命周期，并辅以实操展示关键功能的实现。本书分为 11 章，每一章节的核心内容如下。

第 1 章： AI 原生应用及其架构

第 2 章： AI 原生应用的关键要素

第 3 章： AI 应用开发框架

第 4 章： 上下文工程

第 5 章： AI 工具

致谢

我们曾经参与编写过《云原生应用架构白皮书2022》、《云原生应用架构白皮书2023》、《Nacos架构与原理》、《微服务治理技术白皮书》等电子书，也参与编写过《面向 LLM 应用的可观测性能力要求》、《人工智能云 AI 网关能力要求》等标准，具有一定的编写经验，但是在发起这本《AI 原生应用架构白皮书》之初，便深刻感受到 AI 时代，产品创新的速度之快、架构的复杂度之高、学科的交叉度之广，都远超以往，技术成熟度也仍处于行业发展初期。单个团队已经很难系统、全面的去解构 AI 原生应用架构。因此，我们邀请了阿里云内的上下游兄弟团队、阿里巴巴爱橙科技等联合编写该白皮书，在此表示诚挚的谢意。编写过程中，我们借助 AI 提升了内容的结构化程度和完整度，并逐字进行了人工校对。

我们期望以抛砖引玉的姿态，为 AI 原生应用的标准化、体系化发展提供参考框架，并计划不定期对白皮书进行更新，持续呈现 AI 原生应用架构的前沿思考。我们非常欢迎业内各方一道，无论您是学者、开发者、或是企业等，参与进来共同更新白皮书，共同定义行业共识、破解技术瓶颈，加速推动 AI 从概念走向产业、从潜力转化为价值，让 AI 真正成为驱动全球产业升级与社会进步的核心力量。

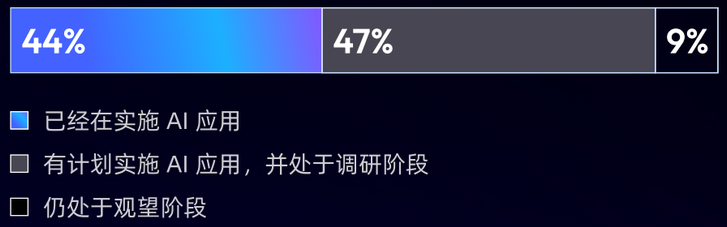
若白皮书能对各个人学习和企业落地 AI 原生应用架构起到一点点的促进作用，将是我们莫大的荣幸。

谨以此书，献给参与 AI 建设的所有同行者们。

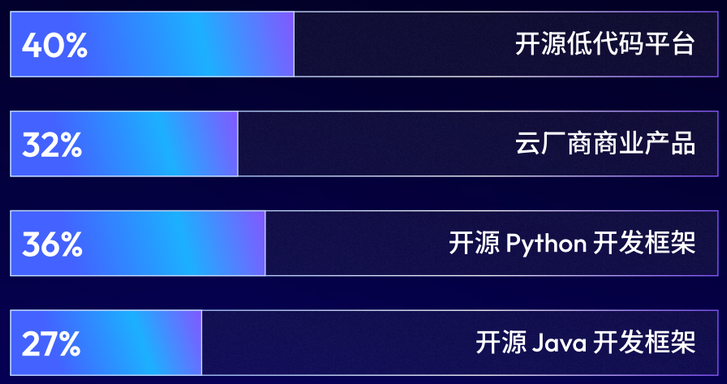
数据洞见

今年，我们在杭州、上海、北京、深圳、广州举办了6场 AI 原生为主题的线下开发者沙龙，报名总人数达1382人，我们借此机会调研了国内 AI 原生应用的落地进程，不同城市的调研结果略有差异，但是整体均差不大，因此我们汇总在一起，提供一个全局的概览，供大家参考。（除了第一题是单选，其他均是不定项选择题）

实施进程



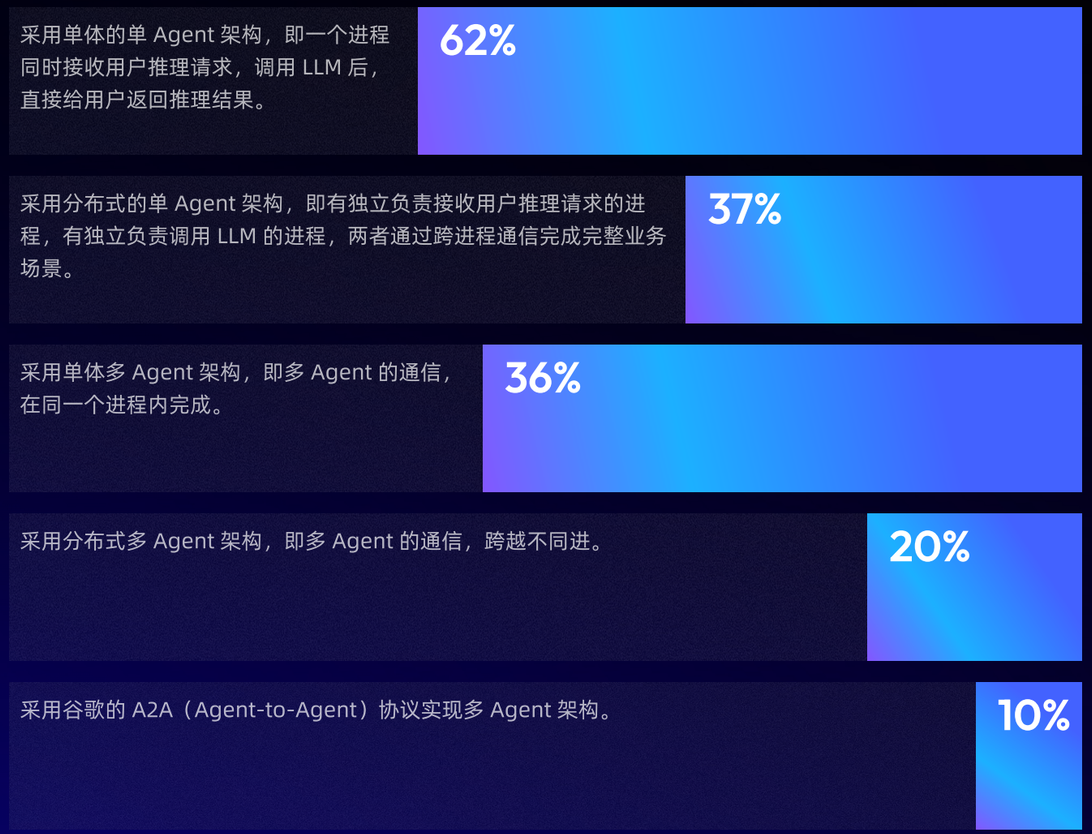
开发框架或平台



落地场景

提升员工体验	重塑客户互动	重塑业务流程	推动创新突破
46%	62%	56%	37%
如会议纪要自动生成、邮件处理和回复、文档起草和编辑、代码开发辅助等。	如智能客服聊天机器人、智能玩具、智能终端、个性化产品推荐、多语言翻译服务、客户情感分析等。	如发票自动处理、合同分析和审查、数据分析和报告、供应链优化、预测性维护等。	如药物研发加速、新材料发现、教育个性化、内容创作、科学实验分析等。

架构或通信协议



Tool（包含 MCP）的采用情况

主要挑战

长会话状态管理

68%

难以保障长耗时、多轮次对话的连续性，面临会话上下文丢失、推理任务中断导致资源浪费等挑战。

算力资源调度

52%

在高并发场景下，前端请求流量突发性强，后端算力资源有限，需实现精细化的负载均衡和优先级控制。

异步通信需求

43%

多个 AI Agent 或工作流节点间需高效协作，但同步调用易引发级联阻塞，需引入稳定可靠的异步通知机制。

数据处理链路

49%

RAG 多源数据接入、向量数据库入库，以及实时推理与异步推理等场景，需要构建高效的数据处理链路。

AI 网关诉求

可观测痛点

CONTENTS

目录

CONTENTS

目录

CONTENTS

目录

AI 原生应用及其架构

AI Native Application and Architecture

CHAPTER

01

大模型技术发展回顾和产业价值
AI 时代应用架构的演进
AI 原生应用及其架构的定义
AI 原生应用架构成熟度

02

AI Native Application Components

03

AI Development Frameworks

04

Context Engineering

1.1 大模型技术发展回顾和产业价值

AI 原生应用的发展，得益于底层大模型技术的突破。而大模型技术也悄然改变着应用架构的演进方向。因此，在介绍 AI 原生应用及其架构之前，我们先来简单回顾大模型的发展历程和带来的产业价值。

1.1.1 大模型发展回顾与展望

2022年11月，ChatGPT 的横空出世成为人工智能浪潮的标志性事件，不仅让全球用户首次直观感受到生成式 AI 的潜力，更让业界看到了通用人工智能（AGI）的曙光。这一突破的核心驱动力来自深度学习技术的成熟，即神经网络通过对海量数据的持续学习，能够自主提炼知识规律并生成类人化内容。在此背景下，模型即服务（MaaS）模式快速兴起，企业无需自建复杂模型，只需通过 API 接口即可便捷获取大模型能力。这一阶段，AI 从实验室走向商业场景，逐渐成为各行业数字化转型的核心驱动力。

2024年，大模型技术再次迎来质的飞跃：OpenAI 推出了 o1 模型，其强推理能力标志着大模型在逻辑思维、复杂任务规划等领域的突破；而 4o 模型的发布，则实现了文本、语音、视觉的全模态融合，为跨模态交互和场景化应用奠定了技术基础。

2025年成为 AI 应用落地的关键转折点：随着多家大模型厂商宣布支持模型上下文协议（MCP）以及 Google 推出 A2A 架构，具备自主决策能力的 Agent 迎来大规模落地，这标志着 AI 从单点工具向系统级生产力工具的转变。正如阿里巴巴集团 CEO、阿里云智能集团董事长兼 CEO 吴泳铭在2024年云栖大会上所言："AI 的最大想象力是接管数字世界，改变物理世界"。

展望未来，随着脑机接口、自演化人工智能、量子计算等技术的突破和应用，AI 和人类将成为共同进化的伙伴，走向深度共生，并重构整个社会文明的范式。这种转变不仅影响技术层面，更将深刻改变人类社会结构、经济形态和生活方式。

1.1.2 大模型的五大产业价值

大模型的技术突破不仅重塑了 AI 的能力边界，更通过深度渗透各行业，催生了全新的产业价值

体系。其通过重构研发、生产、服务、决策等核心环节，催生出五大关键价值，成为驱动千行百业智能化转型的核心引擎：

一是效率新工具。
生成式 AI 可自动化生成高质量内容与数据，大幅降低企业在文档处理、产品设计、生产运营等环节的成本及人力依赖，减少重复性工作消耗，为企业运营效率提升提供支撑。

二是服务新体验。
依托大模型的推理与交互能力，可从服务的精准度、个性化适配、定制化水平及交互人性化等维度，打破传统标准化服务局限，重塑用户体验链路，让用户获得更优质的服务感受。

三是产品新形态。
大模型以生成式能力革新内容创作，降低绘画、写作等领域创作门槛；让硬件设备对图像、语音等有更精准的感知和理解，推动产品交互模式发生质的飞跃，催生新形态产品。

四是决策新助手。
AI 融合数据驱动、实时优化等能力，重构企业从战略到运营的决策链条，将传统经验驱动升级为“数据+算法+领域知识”的复合智能，提升决策的科学性与效率。

五是科研新模式。
AI 凭借智能计算平台、数据处理及大模型算法能力，加速科学发现、优化实验设计、解决复杂科研问题，为科研人员提供全新工具与方法，为科研注入活力、提升效能。

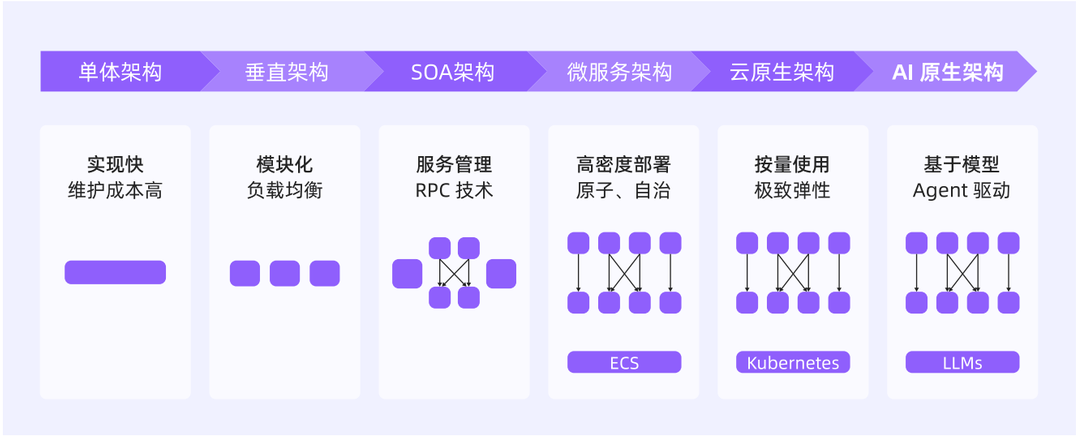
综上可见，大模型的产业价值不仅体现在技术层面的颠覆，更在于其对生产关系、组织形态和社会结构的深层重构。未来，随着模型能力的持续进化，AI 原生应用将进一步打破行业边界，推动社会文明迈入“碳硅共生”的新纪元。

ALIBABA CLOUD

1.2 AI 时代应用架构的演进

1.2.1 IT 应用架构的演进脉络

从计算机诞生至今，IT 应用架构的演进始终遵循“业务痛点→技术突破→架构升级”的逻辑，每一步都源于业务对稳定性、可维护性和协作效率的提升追求：



- **单体架构：**早期业务场景简单，单体架构以一站式开发快速落地，但随着功能叠加，代码耦合导致“修改一处，影响全局”，维护成本陡增，成为业务创新的枷锁。
- **垂直架构：**当业务线分化，垂直架构通过模块化拆分实现负载均衡，缓解了单一应用的膨胀问题，但模块间协作仍依赖硬编码，跨域交互效率低下。
- **面向服务架构（SOA）：**企业级系统互联需求爆发，SOA 以服务化技术实现功能解耦与复用，但集中式服务治理的复杂度，仍制约着响应速度。
- **微服务架构：**互联网流量井喷，微服务将业务拆解为原子级自治单元，支持独立部署与弹性扩展，但细粒度服务带来的运维压力，倒逼技术进一步突破。
- **云原生架构：**Kubernetes等技术通过容器化、集群化管理，解决了微服务的运维难题，实现按量使用、秒级弹性的极致资源调度。至此，云不再只是资源池，而是默认的运行环境。

可以看到，每一次架构的升级，都是在满足业务规模更大、需求变化更快、资源成本更低情况下的诉求，先用拆分降低复杂度问题，或用平台化屏蔽复杂度问题。

1.2.2 云原生应用架构向 AI 原生应用架构的跃迁

过去十年，云原生（Cloud Native）重塑了应用架构的基石，它强调以容器、微服务为代表的基础设施能力，确保应用能够在云环境下具备敏捷性、可扩展性和可观测性。今天，AI 成为新的需求放大器，给应用提出了智能优先的命题，促使全行业迈向 AI 原生。如果说，云原生解决的是如何高效地运行，那么 AI 原生是在此基础上解决如何智能地运行。

在大语言模型（LLM）出现之前，AI 以功能模块形态嵌入系统，包括图像识别、推荐算法、风控模型等，它们依赖监督学习和既定规则，边界清晰，职责单一，不会去改变系统的核心架构。

LLM 的诞生打破了这一边界。LLM 具备通用理解、推理和生成能力，并能通过函数调用、外部工具联动和知识库，形成可扩展的 Agent 体系。由此，AI 由嵌入功能跃升成为应用的底座。

因此，一种全新的应用范式，AI 原生应用（AI Native Application）应运而生，其运行逻辑不再完全由工程师编写的代码所决定，而是由大模型进行自主判断、行动和生成，并具备以下3个特征：

- 以 LLM 为核心，用自然语言统一交互协议；
- 以多模态感知扩展输入边界，以 Agent 框架编排工具链；
- 以数据飞轮驱动模型持续进化，实现系统的自我优化。

当我们说 AI 原生应用的时候，并非抛弃云原生应用。相反，它建立在云原生的基础之上，依然会广泛使用容器化、容器编排和微服务等技术，来确保 AI 原生应用能够实现弹性、可靠、高效地部署和运维。广义上讲，无论是云原生应用，还是 AI 原生应用，都越来越依赖云这一基础设施，基于云来构建应用，两者都是云原生的应用。

应用架构是指导如何系统性地构建应用。在云原生应用架构中，我们讨论的是容器如何管理、服务如何拆分、流量如何治理。而在 AI 原生应用架构下，其目标是在满足可扩展、可观测、安全合规的同时，最大化释放大模型的智能潜力。

1.3 AI 原生应用及其架构的定义

上一节我们讲到应用架构是指导如何系统性地构建应用。在 AI 原生应用架构下，其目标是在满足可扩展、可观测、安全合规的同时，最大化释放大模型的智能潜力。以下是典型的 AI 原生应用架构，涵盖了模型、应用开发框架、提示词、RAG、记忆、工具、网关、运行时、可观测、评估和安全等关键要素。我们将在第2章中展开。

在此架构之上，构建的 AI 原生应用，是以大模型为认知基础，以 Agent 为编排和执行单元，以数据作为决策和个性化基础，通过工具感知和执行的智能应用。AI 原生应用的出现，标志着智能软件形态的根本性转变，其核心能力可以归纳为以下四个方面。

1.3.1 大模型推理决策

在传统应用中，业务执行逻辑通常由开发者使用编程语言（如 Java、C++ 等）进行编码，其执行路径在设计阶段即被固定，缺乏灵活性和自适应能力。相比之下，AI 原生应用以大语言模型（LLM）为核心驱动，开发者可以通过 Prompt（提示词）等自然语言方式完成业务逻辑的构建与配置，从而显著降低开发复杂度。

依托大语言模型在语义理解与推理方面的能力，AI 原生应用能够在面对模糊和复杂的开放式任务时，自主生成和调整业务执行逻辑，并根据需要完成工具调用与流程编排。这一特征使其不再局限于处理预定义的有限问题，而能够扩展至应对更复杂和动态的现实世界场景。

此外，大语言模型不仅具备对既有信息的理解与处理能力，还具备生成新内容的能力。基于这一特性，AI 原生应用的价值已超越传统意义上的效率提升与服务优化，进一步演化为人机协作中的内容生成引擎和创新催化器，为产品研发和科学研究的创新模式提供了新的可能性。

1.3.2 Agent 编排和执行

传统应用更多是工具，Agent 却是一个助手或者伙伴。这个助手能够有聪明的大脑（模型），丰富的经验和记忆（数据），灵巧的双手（工具），并且基于设定的角色协同完成的任务。 当单 Agent 无法完成复杂任务的时候，可以协同多 Agent 编排完成复杂任务；当自身能力受限的时候扩展工具，乃至自己编写工具完成任务。从而确保能够自主感知，决策，行动。

1.3.3 数据优化决策

由于模型输出具有概率性和不确定性，AI 原生应用的逻辑表现也可能存在偏差，甚至在某些情况下完全不符合用户或业务方的预期。为了解决这一问题，AI 原生应用必须具备基于数据驱动的持续进化能力。

在多轮交互中，AI 原生应用需要能够持续保留并利用历史信息，以便理解用户的偏好、行为习惯与目标。这使得应用不仅能够准确响应用户需求，还能在长期使用过程中识别并把握用户的整体行为模式，从而形成更为精准的个性化响应。

同时，应用还需要通过数据采集构建高质量的评测数据集，并结合行业数据、用户反馈数据和客户业务数据进行持续评估与优化。通过这一机制，AI 原生应用不仅能够更准确地理解用户需求，还能更好地契合具体业务场景，实现越用越智能的持续进化。

1.3.4 工具调用与环境连接

尽管大语言模型在语义理解与生成方面展现出强大的能力，其运行机制的本质仍是基于“输入 Token → 输出 Token”的序列生成过程。受限于这一机制，模型既无法直接感知外部环境，也

无法获取实时更新的知识，更缺乏对物理世界的直接操控能力。

为弥补上述局限，AI 原生应用通常通过工具调用的方式扩展模型的环境连接能力。支持语音、图像乃至动作等多模态输入，支持个性化语音、界面交互；支持联网检索获取最新信息，并且通过 API 对接外部系统，或直接驱动企业内部系统的业务流程。

在工具调用的支持下，AI 原生应用能够构建起“感知–推理–行动”的闭环架构：一方面实现对外部环境的感知与分析，另一方面通过工具完成对现实世界的作用与反馈，从而推动“模型 + 工具”的协同运行模式。

1.4 AI 原生应用架构成熟度

1.4.1 AI 原生应用架构成熟度的定义

AI 原生应用架构成熟度是指用于综合衡量 AI 原生应用在技术实现、业务融合与安全可信等方面所达到的水平，客观反映其从简单功能集成到复杂智能决策的演进阶段与发展层次。成熟度评估不仅关注技术能力的实现程度，更着重于应用在真实场景中创造的业务价值及其可持续进化能力。

在技术实现方面，AI 原生应用与传统嵌入 AI 功能的应用存在根本性差异。其核心特征体现为 AI 作为中心决策系统，深度融入业务架构的底层逻辑与运行流程。具体而言，首先体现在自然语言交互能力上，系统能够以类人的理解与表达方式与用户进行无障碍沟通，显著降低使用门槛；其次，具备多模态理解与生成能力，可同时处理文本、图像、语音、视频等异构数据，并生成符合情境的多模态输出，增强应用的泛化能力与表现力。

在业务融合方面，高阶的 AI 原生应用需具备动态推理与自主决策能力。这不仅限于基于规则或模式匹配的响应，而是能够应对复杂、模糊甚至冲突的业务目标，在动态环境中进行多步推理、权衡优化并生成可信决策方案。同时，系统可通过在线学习、反馈闭环和知识沉淀等方式，不断适应新场景、优化性能并扩展能力边界，实现从专用智能到通用智能的渐进式跨越。

安全可信方面，AI 原生应用在具备的所有能力必须构建于安全可信的保障机制之上。这包括但不限于数据的隐私保护、模型的透明性与可解释性、决策的公平性与鲁棒性，以及生成内容的合规可控。安全可信是 AI 原生应用实现规模化产业应用的基石，需贯穿于设计、开发、部署与运营的全生命周期。

综上，AI 原生应用架构成熟度是在于推动 AI 从辅助工具转变为核心决策主体，通过安全可信、可持续进化的端到端架构，为规模化的产业智能升级提供一套可靠的实现路径与体系化支撑，以及用于衡量此类应用在技术实现、业务融合与安全可信等方面综合发展水平的评价标准。它不仅反映了应用从功能集成到智能决策的演进层次，更体现了其以 AI 为核心驱动、深度融合场景与数据、实现自主进化与持续创造价值的能力本质。

1.4.2 AI 原生应用架构成熟度的演进

AI 原生应用架构的成熟度评估是其从概念验证走向规模化、产业化应用的关键衡量标尺。本框架将 AI 原生应用架构的成熟度划分为四个连续演进、特征鲜明的等级：概念验证级（M1）、早期试用级（M2）、成熟应用级（M3）和完全成熟级（M4）。该分级体系旨在系统性地评估应用在技术实现、业务融合、价值创造及安全治理等方面的综合能力水平，为开发者和企业提供清晰的演进路径与优化方向。

级别	英文	中文	定义
1级	PoC Level	验证级	在特定业务场景中，通过基础大模型实现效率提升的初步探索。
2级	Pilot Deployment Level	试用级	AI 应用开始处理更复杂的任务，形成感知—决策-反馈的初步闭环能力。
3级	Operational Integration Level	应用级	AI 应用已深度融入现有业务系统并能够驱动核心业务流程，具备多模态感知和复杂推理能力。
4级	Enterprise Maturity Level	成熟级	高度自主化与自适应的 AI 原生应用，成为业务创新的核心引擎。

AI 原生应用架构成熟度分级描述

1、概念验证级 (M1)：单点功能辅助

在此阶段，应用的核心目标是验证 AI 技术在特定业务场景下的技术可行性。AI 功能通常以孤立的组件或模块形式存在，承担诸如图像识别、文本生成或简单问答等单项任务。其决策逻辑相对简单，多为预定义规则与基础模型能力的结合，尚未形成与核心业务流程的深度闭环。数据利用以离线、批处理为主，模型更新周期长，应用价值主要体现在效率提升的初步探索上，尚未触及业务核心。安全与治理机制处于初步构建阶段。

2、早期试用级 (M2)：场景化初步闭环

应用进入有限范围的试点试用。AI 开始深入特定业务环节，能够处理更复杂的场景化任务，并初步形成“感知-决策-反馈”的闭环能力。例如，在客服场景中，AI 不仅能回答问题，还能根据对话上下文进行意图理解与多轮交互。技术架构上，开始引入流水线化的数据预处理与模型微调机制，支持一定程度的在线学习与迭代优化。业务价值表现为在特定场景下实现自动化决策，有效降低人力成本。数据安全与隐私保护机制被纳入设计考量，但治理体系尚未完全成熟。

3、成熟应用级 (M3)：核心业务深度集成

AI 已成为驱动核心业务流程的关键组成部分。应用具备多模态感知、复杂推理和跨场景协调能力，能够在动态环境中进行实时决策与资源调度，显著提升业务运营的智能化水平。例如，智能供应链系统可基于实时市场需求、库存与物流数据，自主进行预测与补货决策。技术层面，建立了企业级的一体化 AI 平台，支持模型的持续集成、部署与监控，实现高效能的数据利用与模型迭代。业务价值从降本增效延伸至模式创新与收入增长。建立了体系化的安全、合规与伦理治理框架，保障应用的可靠性与可信度。

4、完全成熟级 (M4)：企业级自适应迭代

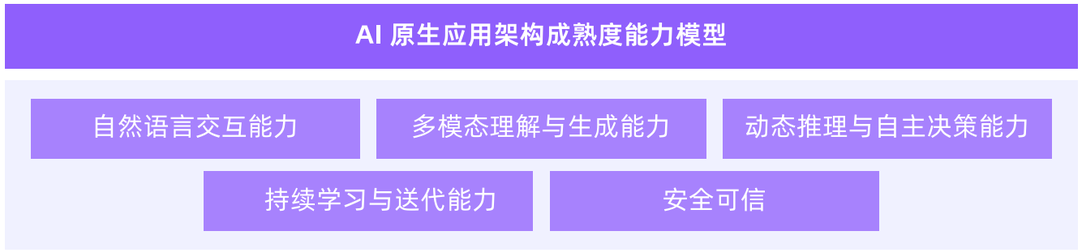
AI 原生应用达到高度自主化与自适应的形态，成为业务创新的核心引擎。应具备前瞻性预测、战略级决策与自我优化能力，能够应对未预见的变化，并主动驱动业务变革与增长。技术层面，构建了企业内外的知识融合与协同网络，模型具备持续自学习与跨领域迁移能力。应用的价值体现在创造全新的商业模式、产品与服务，并构建起可持续的竞争优势。安全、可信与伦理要求已内生于系统设计的每一个环节，能够实现前瞻性的风险防控与全局治理。

AI 原生应用架构的成熟度演进是一个从模块化到集成化，最终迈向驱动化的过程。每一等级的提升，都代表着技术能力、业务融合度、价值创造力和治理水平的系统性飞跃，为各类组织评估自身应用水平、规划未来发展路径提供了科学的参考依据。

1.4.3 AI 原生应用架构成熟度的评估

AI 原生应用架构成熟度评估是指衡量其作为以人工智能为核心驱动力的新型应用形态，在技术实现、业务融合与安全可信等方面所达到的综合水平。这一评估体系不仅关注技术能力的完备性，更强调应用在复杂场景中实现自主决策、持续进化及创造业务价值的实际效能。

成熟度间接反映了 AI 原生应用从初期的功能验证到高度自治的智能体形态的演进阶段，是指导其健康发展与迭代优化的重要依据。该成熟度评估体系以五大能力特征作为核心评估维度，通过系统化测评应用在自然语言交互、多模态理解与生成、动态推理与自主决策、持续学习与迭代以及安全可信保障等方面的能力水平，实现对AI原生应用架构发展阶段的精准诊断与量化评估。



AI 原生应用架构能力模型（持续迭代中）

1、自然语言交互能力

- **功能定义：**衡量应用以自然语言为媒介，实现高拟人化、无障碍人机沟通与任务执行的能力。其核心在于深度理解用户指令的语义、上下文及意图，并生成符合人类交流习惯的回应。
- **评估要点：**重点评估其意图识别准确率、多轮对话维持能力、上下文理解深度以及应答生成的自然度与准确性。该能力是应用实现低门槛交互和普及化的关键指标。

2、多模态理解与生成能力

- **功能定义：**衡量应用对文本、图像、语音、视频等多源异构信息的综合感知、融合理解与跨模态生成的能力。其功能在于突破单一数据模态的局限，实现对现实世界复杂信息的综合处理与表达。
- **评估要点：**主要评估其跨模态检索与关联精度、多模态信息融合效果、以及跨模态生成的质量与一致性。该能力是应用服务于复杂场景的基础。

3、动态推理与自主决策能力

- **功能定义：**衡量应用在复杂、动态且不确定的环境中，进行多步逻辑推理、态势研判并生成最优决策方案的能力。其功能超越了基于固定规则的自动化，实现了对未知情境的主动应对与策略规划。
- **评估要点：**重点评估其应对突发事件的响应与策略调整能力、多目标约束下的决策优化水平、反事实推理能力以及决策结果的准确性与可解释性。该能力决定了应用在关键业务场景中的核心价值。

4、持续学习与迭代能力

- **功能定义：**衡量应用在全生命周期内，通过反馈数据、新知识注入和环境交互，实现性能自我优化、知识库持续扩展以及功能迭代升级的能力。其功能确保了应用能长期适应需求变化，避免性能衰减。
- **评估要点：**主要评估其模型增量学习与微调效率、基于反馈闭环的优化效果、知识发现与沉淀能力以及版本平滑演进与回溯机制的完备性。该能力是应用保持长期活力和降低维护成本的核心。

5、安全可信

- **功能定义：**衡量应用在数据隐私、模型安全、算法公平及系统鲁棒性等方面提供的全面保障能力。其功能是确保应用在合规前提下安全、稳定、可靠地运行，并赢得用户信任。
- **评估要点：**系统评估其数据加密与隐私保护技术强度、模型对抗样本的鲁棒性、决策公平性与可解释性、内容生成的安全性过滤机制以及合规性认证情况。该能力是应用实现规模化部署和商业化推广的前提条件。

本功能性评估体系通过上述五个维度的标准化度量，旨在为 AI 原生应用的规划、开发、部署与优化提供一套完整的诊断工具和行动指南，最终推动其从概念验证走向规模化、高价值的产业应用。通过系统化的成熟度评估与建设，能够有效推动 AI 技术从单点工具向系统级智能基础设施演进，为各行各业的数字化转型提供核心驱动力，最终形成安全可控、持续进化、价值可衡量的智能应用生态。

AI 原生应用的关键要素

AI Native Application Components

CHAPTER

01

AI Native Application and Architecture

P25-P36

02

模型
框架
提示词
RAG
记忆
工具
网关
运行时
可观测
评估
安全

P39-P68

03

AI Development Frameworks

P71-P106

04

Context Engineering

P109-P130

2.1 模型

本章我们将介绍 AI 原生应用的 11 个关键要素。首先从模型开始。

在 AI 原生应用中，大模型扮演着大脑的角色，负责核心的理解、推理与生成任务。这与传统应用截然不同。传统应用依赖固定的规则，难以应对复杂多变的场景；而大模型的引入，则赋予了应用灵活的思考与决策能力，使其真正具备智能。

2.1.1 模型分类

大语言模型（简称大模型）是一类具有大量参数（通常在十亿以上），在极为广泛的数据上进行训练，并适用于多种任务和应用的预训练深度学习模型。

大模型的类型和提供厂商都非常多，根据用途和能力的不同，大致可以分为通用大模型和垂直领域模型。

通用大模型是我们最熟悉的类型，如 GPT、Claude、Qwen、DeepSeek、Gemini 等系列。它们拥有非常大的参数规模，具有广博的知识和强大的通用推理能力，它们可以是纯文本的，也可以是多模态的，能够同时理解图像、声音和文字。一般来说，模型越大，处理复杂、开放性任务的能力就越强，但是相应的成本和延迟都相对较高。

垂直或领域模型不具备通用知识，其核心理念是放弃追求通用性，只专注于在垂直行业或特定领域和任务上实现极致的效率和性能。比如在情感分析、语言翻译、意图分类等特定领域都有对应的垂直领域模型。一些简单的意图分类和信息提取任务，由一个轻量、快速的专用模型处理，往往更具效率和成本优势。

虽然大模型的综合能力很强，在处理开放性、高复杂度的任务时效果更好，但是认为所有任务都需要使用最顶尖的模型是一个误区，许多场景下小模型同样能出色地完成任务，甚至做得更好。在实际落地生产的复杂应用中，通常是大模型与垂直领域模型协同工作，分别处理复杂、开放的任务和简单、高频的任务，从而优化整个系统的效果、成本与响应速度。

2.1.2 模型能力和微调

虽然模型的能力非常强大，但是他们的能力主要源于庞大的预训练数据，这也意味着无论模型多强大，它的知识都是固化的。它不认识你公司的最新产品，也不了解你应用的具体 API，更不知道你数据库里的实时信息。它的所有能力，都建立在对预训练数据中存在的模式和逻辑的理解之上。模型本身并不知道你的业务场景里独有的规则和工具，你需要将这些信息结构化地描述清楚并且提供给它，它会在每次的交互中动态地理解你提供的内容，并结合自身的通用知识来完成内容的生成。模型是 AI 原生应用的重要组成部分，但两者并不等价。

为了追求极致性能，许多团队会考虑训练或微调自己的专属模型。他们会基于自身的业务场景，以及标注的私有数据来进行训练和微调。由于专属模型在训练过程中见过大量与你业务相关的标注数据，在业务理解和工具调用时的表现会显著优于通用的大模型。

虽然专属模型的表现非常好，但是它并不适用于所有企业，训练和微调的成本是巨大的，它不仅包括高昂的计算资源费用，还需要海量的高质量标注数据、专业的算法团队以及漫长的开发周期。这对于绝大多数企业而言，投入与产出不成正比，很难持续。

2.1.3 如何选择模型

考虑到模型的种类繁多，且各有能力边界，还有多种定制化的可能，模型的选择也是 AI 原生应用开发过程中的一个难点。这里不存在一劳永逸的银弹或通用的最佳模式，企业必须基于自己的实际业务场景，权衡任务复杂度、性能要求、开发成本和响应延迟等多个因素来综合选择和组合最合适的模型。

一个务实的策略是从顶配开始，逐步优化：先用能力最强的模型搭建原型以验证业务逻辑，再逐步将流程中的非核心、简单任务替换为更经济、更快速的小模型，最终找到成本与性能的最佳平衡点。一个优秀的 AI 原生应用，其模型架构往往不是单一的，而是一个经过精心设计的、由不同规模和专业度的模型协同工作的有机系统。

2.2 框架

相比基于 Spring 或 Dubbo 开发一个微服务应用，开发一个 AI Agent 应用，你将面临众多的开发框架选择。框架的丰富性，不仅仅是因为编程语言和上手难易程度的区别那么简单，深层次原因是：**代码的确定性 vs LLM 的不确定性：**

- 微服务应用，Spring 和 Dubbo 本质上解决的是确定业务逻辑下的组件编排，包括服务发现、RPC 调用、分布式事务、服务配置等。
- 这些问题的解法相对确定，整个行业逐渐收敛到一些最佳实践上，框架自然容易标准化。

AI Agent 则不同：

- LLM 驱动的 Agent，输出存在高度不确定性，没有一个能放之四海皆准的设计模式，Chain of Thought、ReAct、Plan-and-Execute、Multi-Agent 等模式的落地方式差异很大。
- 更别说原本就依赖设计模式来定义的开发框架的标准化了，有的开放框架偏重链式推理（例如 LangChain），有的偏重知识检索（例如 LlamaIndex），有的则强调人机的混合编排（例如 Dify）。
- 大家对“不同业务场景下，Agent 要如何解决核心问题”，是无法在框架层达成一致的。

因此，Agent 的应用开发框架天然就很难收敛。不同的框架都有自己的设计模式哲学，只要定位清晰，都能获得一部分开发者群体的青睐，一家独大的情况很难出现。

本节将带大家了解下业内主流的 Agent 设计模式，这对我们应用开发框架的选型，以及如何充分利用，至关重要。

2.2.1 Agent 设计模式

1、Chain of Thought（思维链）

- **提出背景：**Google Research 在 2022 年发表的论文《Chain-of-Thought Prompting Elicits Reasoning in Large Language Models》。
- **核心思想：**让模型在回答前，把推理过程一步步写出来。不是一口气报出答案，而是把整个推理过程展示出来。

- **场景例子：**问小王比小李大 1 岁，小张的年龄是小李的两倍。如果三个人的年龄加起来是 41 岁，问小王多大？思维链方式：假设小李的年龄是 x，那么小王 = x + 3，小张 = 2x，总和 = (x + 1) + x + (2x) = 4x + 1，4x + 1 = 41，4x = 38，x = 10，所以小王 = 10 + 3 = 13。结果小王 13 岁。这种方式在逻辑推理、数值计算、逐步分析类问题里，会显得更稳健。

2、Self-Ask（自问自答）

- **提出背景：**Microsoft Research 在 2022 年的研究工作《Self-Ask with Search》。
- **核心思想：**让模型在回答时学会“反问自己”，把大问题拆成多个小问题，然后逐个回答。
- **场景例子：**问2016 年奥斯卡最佳男主角的年龄是多少？Self-Ask 会先问：2016 年奥斯卡最佳男主是谁？（答：李奥纳多·狄卡比奥），再问他当时多大？（答：41 岁），最后组合答案。这种方式特别适合事实链路长的问题。

3、ReAct（推理 + 行动）

- **提出背景：**Princeton 与 Google Research 在 2022 年论文《ReAct: Synergizing Reasoning and Acting in Language Models》。
- **核心思想：**在推理（Reasoning）和外部行动（Acting，比如调用搜索引擎或 API）之间交替进行。ReAct 比 CoT、Self-Ask 更全能，原因在于它不仅是推理模式，还内建了与外部世界交互的闭环。
- **场景例子：**问杭州昨天的天气？ReAct 会先想：“我不知道昨天的天气，需要查询”，然后执行“调用天气 API”，再推理并回答。这让 Agent 既有思维，又能动手。

4、Plan-and-Execute（计划与执行）

- **提出背景：**出现在 2023 年前后的 Agent 应用开发框架实践（如 LangChain 社区）。
- **核心思想：**把任务拆成两个阶段，先生成计划（Planning），再逐步执行（Execution）。
- **场景例子：**假设你让 Agent 写一篇“新能源车的市场调研报告”，它不会直接生成报告，而是先拟定计划：收集销量数据，分析政策趋势，总结消费者反馈，撰写结论。然后逐条执行。适合多步骤、需长时间任务的场景。

5、Tree of Thoughts (ToT，树状思维)

- **提出背景：**Princeton 和 DeepMind 在 2023 年的论文《Tree of Thoughts: Deliberate Problem Solving with Large Language Models》。
- **核心思想：**不是单线思维，而是生成多条思路分支，像树一样展开，再通过评估机制选出最佳分支。
- **场景例子：**解一道数独时，Agent 会尝试多个候选解法（分支 A、B、C），逐步排除错误分支，最终选出唯一解。适合复杂规划和解谜任务。

6、Reflexion / Iterative Refinement（反思与迭代优化）

- **提出背景：**2023 年论文《Reflexion: Language Agents with Verbal Reinforcement Learning》。
- **核心思想：**Agent 具备自我纠错的能力，犯错后会总结失败原因，再带着反思尝试下一次。
- **场景例子：**让 Agent 写一段 Python 代码，如果第一次运行报错，它会读报错信息，反思“函数参数写错了”，然后自动修正并重试。适合代码生成、流程执行类场景。

7、Role-playing Agents（角色扮演式智能体或者说是多智能体协作）

- **提出背景：**源自 AutoGPT、ChatDev、CAMEL 等社区项目。
- **核心思想：**把任务拆分给不同角色的 Agent，每个 Agent 都有专属职责，通过对话协作完成任务。
- **场景例子：**一个软件开发任务里，有产品经理 Agent 写需求文档，程序员 Agent 写代码，测试 Agent 写测试用例。它们像团队一样协作。适合复杂系统开发或跨职能协同。

这些认知框架，其实构成了 Agent 世界里的思维模式库：

- CoT：一步步写过程
- Self-Ask：拆分成小问题
- ReAct：既思考也动手
- Plan-Execute：先计划再执行
- ToT：树状多分支探索
- Reflexion：自我反思迭代
- Role-playing：多人协作分工

它们并不是互斥的，可以混搭使用，理解这些模式，能让我们在应用开发框架选型和使用时，想的更为透彻，一些设计模式，例如 ReAct，已经被 LangChain、LlamaIndex、Dify、Spring AI Alibaba 等 Agent 开发框架内置成基础框架，帮助开发者提升模型的调用效果。

2.2.2 Agent 开发框架

说完 Agent 设计模式，我们再来看 Agent 开发框架。整体来看，开发框架可以归纳为三种主要形态：低代码、高代码与零代码。低代码相对已经比较成熟，高代码是当前生产的主流形态，而零代码则代表了未来的演进方向。与此同时，多 Agent 的分布式架构正在成为跨越三种模式的长期趋势。

1、从低代码到高代码

在 AI 原生应用的早期探索中，低代码工具发挥了重要作用。Dify、Flowise、Coze、阿里云百炼、Cloud Flow、n8n 等产品，通过可视化编排和模板化配置，使非专业开发者也能快速拼装出应用雏形。这类工具极大降低了试错成本，为企业内部的概念验证（PoC）和小规模试点提供了便利。

其实低代码平台在运行时也离不开底层引擎的支撑，大多数低代码平台的底层引擎和管控部署在一起，这限制了 Agent 的性能和可扩展性。但更重要的原因在于低代码平台是对于高代码的一层封装，其抽象层次很难满足所有场景，无法在性能、可扩展性和复杂业务逻辑方面满足大规模生产的要求。这也是为什么在进入大规模生产应用阶段后，很多低代码方案都需要迁移到高代码框架中实现。

高代码则代表了当下 AI 原生应用生产落地的主流形态。ADK、LangGraph、AutoGen、AgentScope、Spring AI Alibaba 等框架，为开发者提供了面向 Agent 的编程接口。相比低代码，高代码具备更高的性能可控性、更强的灵活性以及更好的可预测性，能够支撑复杂场景下的业务逻辑实现与系统集成。来自阿里云客户实践进一步验证了这一点：目前在大规模业务场景落地的 Agent，大部分都是基于高代码方案。

2、高代码的演进

在 AI 原生应用的三种构建模式中，高代码模式最贴近工程师对系统的可控需求。此类开发方式不限于使用现成 Agent 框架，更注重灵活的编排、精准的上下文控制、可靠的执行机制，以及对复杂任务的支撑能力。

高代码模式本身经历了从 ChatClient → Workflow → Agentic 的演进过程。

- ChatClient 阶段：最初的实现仅是一次单一的 LLM 调用，简单但缺乏复杂任务执行能力；
- Workflow 阶段：通过将传统工作流转化为 LLM 节点编排，实现了自主性与确定性的初步平衡，但由于编排复杂，维护成本较高；
- Agentic 阶段：逐渐成为主流形态。它通过提供面向 Agent 的 API，并内置多种通用的协作模式（Pattern），使开发者能够在 Agentic 自主性 和 Predictability（可预测性） 之间取得平衡，从而兼顾开发效率与执行准确性。

这一演进过程，折射出 AI 原生应用在落地时面临的核心挑战：既要让系统具备足够的智能性和自

主性，又必须确保其行为在工程意义上可控、可验证。

3、零代码的愿景

相比之下，零代码代表着更远期的方向。MetaGPT 等探索性产品，尝试让用户完全通过自然语言即可驱动应用开发，依赖模型本身的推理与规划能力完成任务分解、逻辑编排和工具调用。零代码的潜力在于真正实现 AI 应用的全民化与智能自治，但现实中受制于模型能力，其生产可用性仍不足：复杂业务场景对推理深度、上下文管理和可控性的要求，远超当前模型的稳定水平。

因此，零代码模式目前更多处于探索与验证阶段，难以承担大规模生产任务。但它所代表的愿景，展示了未来 AI 应用可能的终极形态。

白皮书的第 3 章中将以 Spring AI Alibaba 为例，探讨如何使用 AI 应用开发框架开发一个简单的智能体，以及单智能体和多智能体的区别和联系，智能体的部署方式，消息驱动的智能体开发。

2.3 提示词

在 AI 原生应用中，我们的编程方式发生了根本性的变化。不再编写复杂的代码，而是使用一种更接近人类语言的方式与 AI 沟通，这就是 Prompt（提示词）。Prompt 的质量很大程度上影响了 AI 应用的输出效果。

2.3.1 Prompt 是什么

Prompt 是用户向 AI 模型提供的输入指令，用于引导模型生成期望的输出。它可以是一个具体的问题、一段描述、一组关键词，或是相关的上下文信息，其核心作用是告知模型用户期望获得什么样的内容。Prompt 的载体也不仅限于自然语言文本，还可以包含代码片段、数据格式说明，甚至是图像与文字相结合的多模态输入。

2.3.2 Prompt 质量 = AI 输出质量

大模型输出质量并非完全取决于模型本身，还依赖于输入的 Prompt 是否清晰、完整、具体。在 AI 领域，有一句经典的话“Garbage In, Garbage Out”（垃圾进，垃圾出）。这句话在提示词工程中也同样适用，Prompt 的质量直接决定了 AI 生成内容的质量、相关性和准确性。

大型语言模型输出内容的质量很大程度上取决于 Prompt 的明确性与具体性。当用户提供一个模糊、开放式的指令时，例如“写点关于人工智能的东西”，模型由于缺乏明确的上下文和约束条件，其生成的内容往往会泛泛而谈。这类回答通常只是一篇缺乏深度和特定信息价值的通用性概述，难以满足专业或具体化的应用需求。

相反，当用户构建一个结构化、包含多维度要素的精确指令时，结果则截然不同。例如，一个明确要求模型“以科技专栏作家的身份，撰写一篇800字左右的文章，探讨人工智能在医疗影像诊断中的最新应用、优势与挑战，并列举实例”的 Prompt，为模型提供了清晰的目标、角色设定、内容框架和格式要求。因此，模型能够生成一篇逻辑严谨、信息详实、专业度高的文章，其输出结果的精准度和实用价值也得到极大提升，更加符合人们的预期。

2.3.3 如何优化 Prompt

优化 Prompt 是与大语言模型高效沟通的关键，一个好的 Prompt 能让模型更精准、更深入地理解你的意图，从而生成质量更高的内容。

早期的研究还表明，你对模型说“这个问题你回答对了，我会奖励给你100元”，“这个问题你回答错误了，你会被惩罚”，这种贿赂或者威胁也能优化模型的生成效果。不过随着模型的进化，这些小技巧都已经变得无效了。但有一个原则是不会变的，开发者需要清晰、有效地与模型交流，并明确指导它如何处理各种情况，这就像是你给一位聪明的助理分配任务，指令越清晰、背景信息越充分，他完成工作的质量就越高。

如何优化 Prompt 是 AI 原生应用开发中的难点，在第 4 章中我们将围绕上下文工程对 Prompt 展开详细介绍。

2.4 RAG

在之前的内容中我们提到过，大模型的知识都是固化的，它不认识你公司的最新产品，而 RAG 就是为模型提供知识库一种有效方法。

基于 RAG（Retrieval Augmented Generation，检索增强生成）构建知识库，是大模型兴起之后最快被采纳接受的架构范式之一。当前，RAG 系统已经被广泛地应用在客服问答、个性化推荐、智能对话助手等场景当中。RAG 技术能够弥补大模型因知识截止而无法获取最新信息的问题，并有效降低其产生幻觉的风险，而且 RAG 技术相比于大模型后训练或微调方式，以更加成本低的方式与企业的专有数据作对接，以实现大模型快速技术验证和商业化尝试。

2.4.1 RAG 知识库的应用架构

基于 RAG 构建知识库的应用架构如下。可以简单划分为离线索引构建和在线检索和生成过程。离线向量过程通过把用户上传的文档进行文档智能解析、切片，再进行向量化存储到向量数据库。在线过程则把用户的请求问题向量化之后与向量库中的切片向量进行相似度比对，从而召回最接近用户问题的相关切片。

如今构建如上图这样的 RAG 系统已经变得非常简单，开源社区和商业产品都提供了非常简便的构建方式。在满足复杂的业务需求的过程中，通常一个简单的 RAG 系统无法满足业务需求，会遭遇准确率和召回率的挑战、信息冗余噪声导致的模型幻觉、知识库庞杂难以管理等问题。当前 RAG 系统的构建也逐步向模块化、Agentic RAG 的高级架构演进。

从离线过程来看，文档解析技术除了经典的 OCR 和电子解析技术，也在利用大模型进行更准确的文档解析，比如对于图片类的文档，通过 VLM 视觉理解大模型，能够对这类文档进行更全面的文档理解。

从在线检索过程来看，检索前、检索中、检索后过程里，都发展出很多的技术手段来加强和管理整体 RAG 的效果。如检索前可以增加 Query 改写、知识库路由等模块，检索过程可以采用混合检索策略，检索后可以增加重排序、拒识模块等。

从构建包含 RAG 的 AI 应用来看，Agentic RAG 成为新的趋势之一，用户将知识库检索作为大模型的工具之一，由大模型来决定是否以及何时进行检索以获取必要的知识库信息。另外，多模态 RAG 技术也是当前蓬勃发展的领域，随着多模态理解大模型能力的增强，多模态 Embedding 向量模型也取得了重大的发展，包括阿里云通义团队也发布了业内广受好评的 Qwen3 Embedding 文本和视觉系列模型。基于多模态向量模型的 RAG 系统在商品搜推、视频创作等各类场景已经获得了规模化的落地。

2.4.2 RAG 知识库的应用场景

知识库落地有广泛的应用场景，包括客户服务、个性化推荐、AI陪伴、内容创作等。其中客户服务 RAG 是最广泛落地的应用之一，从其业务特征来看，通常就需要大量的业务背景知识，并且这些知识是不断更新的，例如常见问题解答（FAQ）、产品规格、故障排除指南以及公司政策等。

在这些场景里，知识库是严格知识的来源、可信任，作为降低大模型幻觉的重要手段。甚至在更加严肃的场景里，许多用户将大模型只作为知识库的整理工具，要求大模型回答需要严格遵循知识库里的知识，不能随意发挥，以避免严重的客诉问题。

不过我们也发现，当前 RAG 的应用也已经超越简单的问答。基于 RAG 的系统，叠加大模型分析客户对话数据等能力，能够帮助企业优化服务策略和挖掘销售线索等。RAG 的价值正在从解决幻觉这一技术问题，向赋能业务的更高层面演进。多模态 RAG 的兴起，将 RAG 的应用边界从纯粹的知识问答推向了更广阔的领域。比如在零售电商场景，用户可以通过上传图片来检索商品，从而实现商品图搜和个性化推荐。而在媒体娱乐领域，多模态 RAG 也帮助从海量音频视频内容中检索出特定的片段，从而服务于音视频内容分发以及新兴的 AI 视频创作场景。

2.4.3 RAG 知识库技术的未来发展

大模型发展至今，RAG 作为最成熟的 AI 应用架构之一，尽管基础 RAG 的实现已趋于成熟，但仍有人认为其技术含量不高。然而，我们观察到，构建一个真正满足复杂业务需求的高级 RAG 系统仍然充满挑战，并且该领域正在不断演进。比如在当前 Advanced RAG 架构里，仍然有许多技术问题待解决。多模态 RAG 相关的技术，也在快速地发展当中，其应用场景和想象力空间更大。

另外，我们把 RAG 拆开来看，知识库向量检索也是当前上下文工程（Context Engineering）的核心技术实现。我们试举两个例子：首先是动态样例（Few Shot）干预，通过在知识库里维护正例、反例，可以实时通过用户 Query 召回相关的样例，补充到上下文，从而降低大模型幻觉；然后是大模型工具检索，当一个 AI Agent 需要接入数量很多的工具时，对于大模型选择工具的和工具入参提取的准确率会造成比较大的挑战，通过构建工具的知识库和动态召回少数工具的方式，可以提升大模型工具调用的准确性和降低时延。

无论未来 LLM 架构如何演变，只要它们仍然依赖外部知识来增强其能力，向量检索作为一种高效、语义化的上下文获取机制，仍然将发挥重要的价值。在白皮书的第 4 章中，我们将围绕上下文工程对 RAG 展开详细的介绍。

2.5 记忆

提到 AI 应用的记忆，其实大家都不陌生，我们在使用 Qwen、DeepSeek 等应用的时候，已经习惯了查看历史对话，也可以在历史对话中继续之前的话题，这就是记忆功能给 AI 应用带来的体验提升。

2.5.1 记忆的核心作用

大模型本质上是无状态的，仅能依赖有限的上下文窗口进行交互，这就导致了交互的非连续性，使模型无法积累连贯的认知或沉淀长期的经验。为此 AI 原生应用的开发中需要引入记忆组件，为模型带来三个维度的能力：跨越会话的连贯性、高度自适应的个性化，以及基于历史信息的深度推理。从而驱动 AI 应用从一个单轮问答的机器人，成长为能与我们长期协作、处理复杂任务的智能伙伴。

1、跨越会话的连贯性

大模型的上下文窗口是无状态的，无法跨越单次会话来维持交互的连续性。每次交互的中断都意味着历史信息的丢失，会造成沟通效率低下和用户体验的割裂。

AI 应用通过引入记忆组件解决这一问题。它能够长期保存关键信息（如对话历史、任务状态、决策依据），并通过高效的检索与上下文注入机制，在新的交互中动态地为模型提供相关背景。这确保了多轮、长周期交互的逻辑一致性，进一步支持长期任务的执行。

2、高度自适应的个性化

在缺乏用户信息的情况下，大模型只能提供标准化的通用响应，无法满足个体用户的特定需求。AI 应用通过记忆构建和维护动态的用户画像来实现个性化。它系统性地记录用户的偏好，如格式要求、沟通风格等，也能记住历史行为模式与长期目标，使模型能够生成高度定制化的输出。这种基于记忆的自适应能力，是 AI 应用从一个通用问答工具转变为个人助理、专属顾问等高级应用形态的技术前提。

3、基于历史信息的深度推理

在处理需要深度分析的复杂问题时，模型若仅依赖当前上下文，其推理过程会因信息不足而变得片面，难以进行有效的因果推断或类比分析。

AI 应用借助记忆提供的历史知识和经验充当推理的证据库，当模型在决策时，能关联并整合分散在历史中的相关数据、相似案例或既有结论。通过这种方式，模型能够进行更全面、更具深度的综合判断，显著提升其在专业领域（如医疗诊断、法律咨询）中的决策质量与可靠性。

2.5.2 短期记忆和长期记忆

记忆通常分成短期记忆（Short-Term Memory, STM） 和长期记忆（Long-Term Memory, LTM）。它们各自有不同的技术实现、特性和应用场景，二者的协同工作构成了 AI 的完整记忆系统。

1、短期记忆

短期记忆，也被称为工作记忆或上下文记忆，指的是模型在单次、连续的交互会话中所能直接访问的信息。

技术上，这通常通过在每次 API 调用时，将完整的对话历史以 messages 列表的形式传递给模型来实现。另一种方式是利用 System Prompt，预置贯穿全局的指令或动态更新的摘要。由于信息被直接放在输入中，这种方式保证了记忆内容的高保真度和即时访问性，模型可以获取未经压缩的原始对话，无需额外步骤。

然而短期记忆的效用受限于模型的上下文窗口这一瓶颈，一旦对话历史的长度超出容量上限，最旧的信息便会被舍弃，不可避免地导致逻辑断裂和信息丢失。此外，不断增长的上下文也会带来 API 成本与推理延迟的显著增加。因此，这种记忆机制在本质上是易失、昂贵且容量有限的，无法独立支撑长期的、连贯的交互需求。

2、长期记忆

长期记忆旨在解决短期记忆的局限，让 AI 能够记住跨越不同会话，甚至数天数月前的信息，如用户偏好、历史项目经验、关键事实等。

具体而言，系统会将需要长期保存的信息，如对话摘要、用户画像或外部文档，进行向量化处理，并存入专门的向量数据库进行索引。当新的交互发生时，系统会根据当前输入的语义，在数据库中高效检索出最相关的历史记忆片段，并将其作为背景知识动态加入到模型的输入中。这种基于外部数据库和语义检索的机制，使得记忆具备了持久性与高度的可扩展性。

然而，长期记忆的引入也伴随着新的挑战与权衡。由于长期记忆存储的通常是信息的摘要或切

片，而非原始对话，因此必然存在一定程度的信息保真度损失。更为关键的是，系统的整体效果高度依赖于检索环节的质量。不精准的检索不仅无益，反而可能引入无关或错误的上下文，从而干扰模型的判断。同时，该架构也带来了更高的系统复杂度和额外的处理延迟，需要在实际应用中仔细考量。

有效增强 AI 应用记忆能力的关键，在于实现短期与长期记忆的动态协同。短期记忆保证了交互的即时性与上下文的完整性，长期记忆则提供跨会话的知识背景。然而，要实现理想效果，开发者必须根据具体业务场景进行精心设计，尤其要在上下文成本与检索质量之间做出权衡。关于这一核心权衡，我们将在第 4 章上下文工程中展开详细分析。

2.6 工具

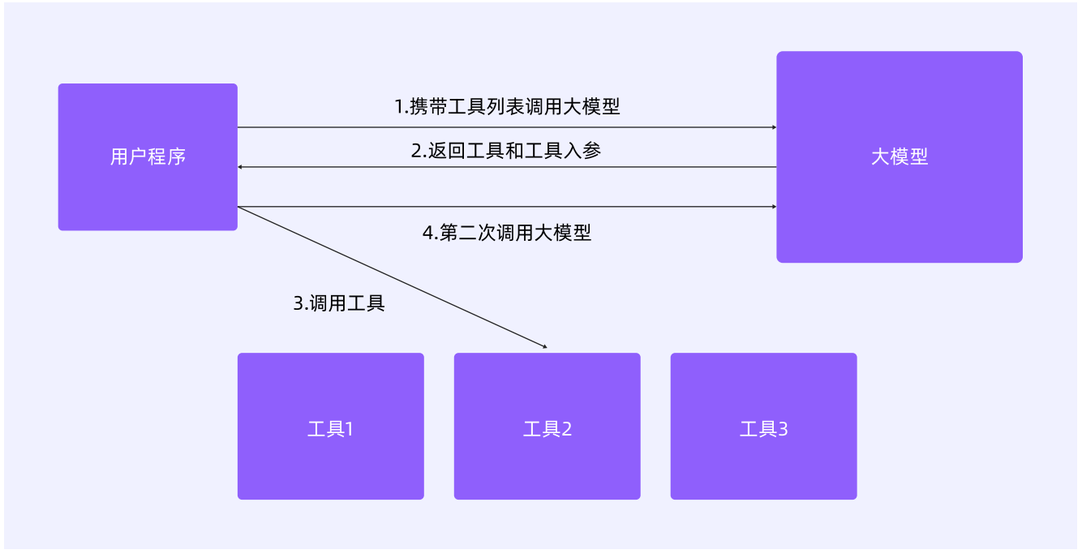
LLM 的知识是静态的，其知识内容截止于其训练数据的最后更新日期 。这使得它们无法直接回答人们关于实时的、动态的信息，例如最新的天气、股票价格或突发新闻。此外，模型本身无法直接与外部系统交互，执行如发送电子邮件、预订航班或执行数据库查询等实际操作。

正是由于这种内在的限制，模型供应商推出了大模型工具调用。工具本质上是模型可以调用的外部接口，使得模型的能力得以延伸至其静态训练数据之外。工具的采用，将 AI 应用从一个被动的回答者转变为一个能够主动采取行动的智能体。

2.6.1 如何调用工具

大模型本身不能够调用工具，它是作为一个思考引擎，根据用户的输入和可用的工具描述，智能地判断并决定调用哪个工具以及传递哪些参数。

当前 AI 原生应用构建的核心范式，主要是大模型通过理解用户意图并决定调用哪个工具，而实际的执行操作则由外部程序完成。这种工具调用分离不仅提升了系统的可扩展性和安全性，也为构建复杂、自主的 AI 原生应用奠定了基础。



OpenAI 率先在其推理 API 接口中支持了 Function Calling（后升级为 Tools） 的工具形式，OpenAI 兼容接口成为当下大部分大模型服务提供商的标准接口形式。函数调用（Function Calling）核心是让大模型生成调用外部 API 的结构化入参 。开发者需要预先向模型提供可用工具的详细描述，通常以 JSON Schema 的形式定义了函数名称、功能描述和参数列表 。当用户发起请求时，大模型会分析其意图，判断是否需要调用工具以及调用哪个工具，然后生成一个包含 tool_calls 字段的结构化 JSON 对象 。这个 JSON 对象精确地指定了需要调用的工具名称以及传递给工具的参数 。应用程序拿到这个结构化入参后，就可以解析并调用相应的工具。

当然，也有很多用户将工具的描述配置在系统提示词中，大模型也会生成一段遵循特定语法的文本，然后供外部应用程序去解析调用。一般上我们更推荐 Function Calling 方式，从而获得更稳定的结构化输出。

2.6.2 MCP 协议

随着大模型与工具的结合成为普遍需求，一个关键问题开始凸显：不同的模型提供商（如 OpenAI、Anthropic、阿里云百炼平台等）对工具调用的实现方式各有差异 ，而每个外部服务（如数据库、搜索 API 等）又都有自己的 API 接口，企业用户也有自己的私有 API。这类工具的对接，就要求开发者对每个工具进行适配开发，Agent 的开发变得繁重甚至不可行。

Anthropic 公司于2024年11月推出了模型上下文协议（Model Context Protocol, MCP），MCP 协议是为大模型与外部数据、应用和服务之间的通信提供一种安全且标准化的语言 。它被形象地比喻为 AI 应用界的 USB-C 接口，其核心在于定义一个统一的协议，使得大模型能够轻松、标准地连接到各种数据源和工具，从而轻松地实现互操作。MCP 协议迅速地成为当前大模型供应商的标准协议，包括 OpenAI、Google、阿里云百炼在内的模型服务商，都对 MCP 做了兼容性支持。

市场上也出现了不少 MCP 服务的托管平台，如阿里云百炼 MCP 广场、魔搭 MCP 服务、MCP.SO 等。MCP 社区为了统一标准，定义了 MCP Registry，旨在标准化服务器的分发和发现方式，使 Agent 和工具更容易连接，并于近期发布了 Preview 版本，用于公开可用的 MCP 服务器的开放目录和 API。Nacos 参与了社区共建、并丰富了该标准的落地形式，开源 Nacos MCP Registry，定位私有化 ，方便企业内部部署。

2.6.3 工具调用的挑战和发展

当前大部分主流模型供应商都已经将工具调用能力作为其大模型的原生功能进行内置，并且在模型预训练阶段对工具调用进行了特定增强。

当然目前基于 Funcation Calling 或 MCP 工具调用的 Agent 实践，也面临着诸多的挑战，比如工具调用时延、工具提取参数准确性、安全鉴权等问题。从长远来看，我们认为，大模型工具调用的这些挑战都将被有效解决，大模型将能更智能地组合和串联多个工具来完成更复杂的跨领域任务，我们将在白皮书的第 5 章 AI 工具展开详细的介绍。

2.7 网关

在 AI 原生应用中，如果说大模型是各具特长的专家团队，那么 AI 网关（AI Gateway）就是连接应用与这些专家的智能总调度中心。随着大模型以周为单位更新，AI 应用以天为单位演进，企业需要在安全、合规、成本、效率四重约束下交付稳定业务。AI 网关正是为了应对这一挑战而生，它解决了传统 API 网关无法处理的模型切换、Token 经济、语义缓存和内容风控等 AI 原生的需求，为整个系统带来秩序、可靠与安全。

2.7.1 什么是 AI 网关

AI 网关是一个专为 AI 应用设计的、位于应用和大模型之间、应用和工具之间、模型和模型之间的中间件。 它是传统 API 网关在 AI 时代的演进，其核心职责不再仅仅是路由和保护 RESTful API，而是要理解并管理以 Token 为中心的、高延迟、流式传输的流量。 由于 AI 应用背后的模型、运行时和框架标准难以统一，AI 网关通过抽象协议和统一治理，将模型、业务 API、外部工具纳入统一控制平面，成为了连接异构模型与多样化业务场景最明确、最关键的统一切面，是 AI 原生应用架构中不可或缺的组件。

2.7.2 统一的模型接入与厂商解耦

AI 原生应用面临的一大挑战是模型服务的多样性与协议的碎片化。 不同供应商（如OpenAI、Anthropic、阿里云百炼、DeepSeek）的 API 标准各异。

AI 网关的首要任务就是屏蔽这种底层复杂性，通过提供统一规范的 API，将所有模型的接口都转换成一个标准的、统一的接口对外服务。 这意味着开发者无需为新模型编写定制代码，无论是切换模型、A/B 测试还是组合模型都变得异常简单，从而有效避免厂商锁定，并提升了开发效率与系统灵活性。

2.7.3 融合存量系统与 AI

企业通常拥有成千上万个正在运行的 REST、GraphQL、gRPC 等协议的存量服务。 AI 网关能够通过其协议抽象层扫描这些服务的 API 规范（如 Swagger），自动生成符合大模型工具调用（如 MCP 规范）的描述文件，并借助 MCP Registry 注册到统一的服务目录中。

这使得企业无需改动存量业务接口的代码，就能将它们升级为 “AI-Ready API”，极大地盘活了企业现有的 IT 资产，避免重复建设。

2.7.4 智能路由与故障转移

AI 网关是一个智能的决策者，能够基于预设策略，动态地将请求路由到最合适的模型服务。

- **策略路由：**不仅可以根据请求内容或用户身份分发流量，还可以依据实时的 Token 单价、延迟、显存占用等权重进行动态推理流量调度，实现成本与性能的最佳平衡。 例如，将简单任务交给经济的小模型，复杂任务交给顶配模型。
- **故障转移：**通过持续监控后端服务健康状态，一旦检测到模型响应缓慢或不可用，便会自动将流量无感切换到备用模型，保障应用的高可用性。

2.7.5 精细化的成本控制与优化

大模型的推理成本是 AI 应用的核心开销，而付费模型的 Token 费用常常不可预测。AI 网关提供了一系列专为降本增效设计的功能。

- **语义缓存：**能够理解请求的意图，对于内容相似但表述不同的重复问题（例如 “北京的天气怎么样？” 和 “查询北京今日天气”）可直接返回缓存，避免对昂贵模型的重复调用。
- **成本与流控：**AI 网关能够实现基于 Token 数量的精准速率限制和预算配额管理。它可以按组织、用户、应用等维度管理 Token 预算，当超出限额后，可以自动降级到成本更低的模型，从而有效防止资源滥用和成本超支。

2.7.6 企业级安全与合规

在企业环境中，安全合规至关重要。AI 网关作为所有 AI 流量的统一入口，是确保全链路合规安全的关键节点。

- **安全合规能力：**网关可内置国密算法支持和敏感语料实时过滤等能力，满足数据安全要求。
- **审计与追溯：**所有流经网关的 Prompt、Response、Token 消耗量等多维度数据都会被记录落盘，以满足实时审计和追溯等合规要求。
- **统一身份认证：**能够与企业内部自有的身份授权基础设施对接，为 AI 生态提供统一的认证授权切面，解决了 AI 生态与企业现有安全体系难以直接融合的问题。

2.7.7 数据、观测与优化

AI 网关的价值远不止于管理和路由，其作为统一控制面的定位，使其成为承载统一数据采集、观测与优化的最佳载体。

正是因为所有 AI 相关的请求与响应都必须流经网关，它才能够捕获每一次交互的完整数据：包括原始提示词、最终响应、模型选择、Token 消耗、调用时延和业务成本等。

- **统一采集：**所有数据都通过一个标准化的方式汇集，为后续的分析提供了坚实基础。
- **全面观测：**这些数据让原本黑盒的模型调用过程变得透明，形成了对系统性能、成本和用户行为的统一视图。
- **驱动优化：**通过分析洞察，可以直接反哺系统，例如自动优化路由策略、更新缓存内容、或筛选出有价值的数据用于模型微调，形成一个持续学习、自我完善的闭环。

白皮书的第 6 章将详细介绍如何快速构建 AI 网关，以及 API 和 Agent 的货币化。

2.8 运行时

随着开发范式从传统的标准库与框架迁移到以 AI 模型与工具链为核心，运行时成为了承载和执行动态业务逻辑、协调智能体协作、管理数据流与模型交互的核心环节，并扮演了不同于传统应用运行时的作用。

2.8.1 什么是 AI 原生应用的运行时

运行时是 AI 原生应用的核心执行环境。与传统应用中逻辑由开发者预先编码、执行路径相对确定的模式不同，AI 原生应用的业务流程往往由大模型根据用户实时意图动态生成。这意味着运行时处理的不再是固定的代码，而是充满不确定性的执行计划。

因此，运行时的核心职责，就是为了驾驭这种高度的不确定性。它需要将模型、工具和数据流有机地整合在一起，不仅要能理解和执行模型生成的动态任务，还要为整个过程提供稳定、高效和安全的保障。

2.8.2 运行时的核心挑战

一个强大的 AI 原生应用运行时，必须直面由其动态化和数据密集型特性带来的三大挑战：

- **动态逻辑的可靠执行：**模型生成的任务计划可能存在逻辑错误或无法执行的步骤。运行时需要具备强大的容错和异常处理能力，确保动态任务能够顺利完成或优雅地失败，而不是简单地崩溃。同时，任务的每一步都可能需要不同的依赖库或计算资源，运行时必须能够按需、即时地准备执行环境。
- **海量与实时数据的高效处理：**数据是 AI 应用的燃料。尤其在 RAG 等场景下，运行时需要在毫秒级内从海量知识库中检索、处理并传递数据，这对存储 I/O 和网络延迟提出了极致要求。此外，对于在线学习、实时推荐等场景，运行时还必须具备高效的流式处理能力，确保模型能基于最新的数据进行推理。
- **异构组件的复杂协同：**AI 应用是由模型、向量数据库、外部 API 工具、多智能体等多个松散耦合的组件构成的复杂系统。运行时需要充当这些组件间的消息总线，提供强大的服务编排和治理能力，并原生支持 MCP、A2A 等主流交互协议，确保它们之间能够顺畅地通信与协作。

2.8.3 面向 AI 优化的 Serverless 架构

面对上述挑战，以 Serverless（无服务器）架构为核心，并为其注入状态管理和性能优化能力，正成为构建下一代 AI 运行时的关键路径。

- **为无状态的 Serverless 引入记忆：**传统的 Serverless 架构是无状态的，难以支持需要上下文的多轮对话或复杂的智能体协同。现代 Serverless 平台通过亲和性调度等机制，可以将同一会话的多次请求调度到同一个预热实例上，实现了状态的就近缓存，巧妙地解决了记忆问题，既保证了性能，又简化了开发。
- **兼顾极致弹性与低延迟：**AI 推理任务的流量往往具有潮汐效应。Serverless 按需执行、自动伸缩的特性完美契合了这一需求，能轻松应对流量洪峰，并在闲时将成本降至为零。针对延迟敏感的场景，通过预留实例和依赖预加载等技术，可以有效解决冷启动问题，实现毫秒级响应。
- **让 AI 工具即插即用：**Serverless 函数是承载 AI 工具的天然载体。每个工具可以被封装成一个独立的函数，由智能体按需、事件驱动地调用。这种按实际调用计费的模式，极大地降低了大量长尾工具的闲置成本。

未来，运行时将继续向着更智能、更自动化的方向发展。它将成为一个能为开发者屏蔽底层复杂性的超级电网，让开发者可以像用电一样，简单、高效、经济地创造和部署 AI 原生应用。白皮书的第 7 章将详细介绍为 AI 应用和工具提供经济、安全算力的 Serverless 运行时。

2.9 可观测

AI 原生应用，因其模型对话、工具调用和 RAG 等复杂的内部流程，决策路径充满了不确定性。这使得传统的监控方法力不从心。传统监控主要关注基础设施的性能指标与日志，难以应对 AI 应用特有的行为不可预测、输出质量波动和成本结构复杂等挑战。因此，我们需要一套专为 AI 应用打造的可观测性体系，以保障其稳定性、可维护性和安全性。

AI 应用可观测性与传统监控的核心区别在于：监控告诉你“发生了什么问题”，而可观测性则要回答“为什么会发生问题”。监控系统通常聚焦于 API 响应时间、错误率等关键指标，主要起预警作用。可观测性则更进一步，通过整合链路追踪、上下文数据等，构建从用户输入到最终输出的完整视图，从而精准定位问题根因，持续优化系统。

一个完整的 AI 可观测体系，应具备3大核心能力：

- **端到端全链路追踪：**提供端到端的日志采集和链路追踪，可视化展示请求在整个 AI 应用中的执行路径。支持对历史对话的灵活查询与筛选，以便于调试和改进。
- **全栈可观测：**包含应用、AI 网关、推理引擎可观测3个纬度，观测内容有实时追踪响应延迟、请求吞吐量、Token 消耗，错误率和资源使用情况（如 CPU、内存、API 令牌），并能在指标异常时触发警报，帮助团队在影响用户前快速响应，同时有效监控成本。
- **自动化评估功能：**通过引入评估 Agent，对应用和模型的输入输出进行自动化的评估，检测幻觉、不一致性或答案质量下降等问题。有效的工具通常会集成评估模板，方便工程师快速的对常见的质量和安全问题进行评估。

在工程实践中，落地可观测性需兼顾技术集成与数据安全。技术集成层面，OpenTelemetry 作为行业开放标准，为打通全链路数据提供了关键支持。它定义了统一的采集规范，使可观测系统能通过自动埋点等方式，无缝接入主流 AI 框架，显著降低了集成成本。同时，由于 AI 应用常处理敏感信息，系统必须内置数据脱敏和访问控制等安全机制，确保合规。白皮书的第 8 章将详细介绍 AI 可观测应具备的 3 大能力和实现方式。

2.10 评估

传统软件应用的评估测试基于确定性逻辑，其核心特点是固定输入必然产生稳定且可复现的输出。然而，AI 应用的行为本质上是非确定性的概率输出，即使输入相同，模型的输出也可能因上下文、训练数据分布或随机性而千差万别。这种非确定性行为使得传统的评估测试方法难以直接适用于 AI 应用。

这种差异使得 AI 原生应用的评估范式从传统的测试方法逐步转向以 LLM-as-a-Judge 为代表的前沿评估范式，并发展出更加完善的评估体系。在测试环境中验证通过的AI应用，在真实生产环境中面对复杂多变的用户数据时，其表现和预期相差甚远。因此，我们必须重新定义评估，它是深度整合到 AI 应用设计、开发、部署和运维全生命周期中的持续过程，旨在保障系统的长期稳定性与可靠性。

2.10.1 构建高质量的数据集

构建高质量、场景相关的评估数据集是评估的核心。构建高质量数据集主要有三种途径。第一种是人工构建，由领域专家依据业务逻辑，精心设计和标注一系列评估用例。这些用例包括标准的问答详情、复杂的多步推理任务，以及那些专门用来测试模型能力边界的边缘场景。第二种是自动化采集，通过对线上系统的观测日志和业务记录进行分析，可以挖掘出海量的真实用户交互数据，尤其是效果不理想的用例，相较于人工构建，成本低且效率高。除以上两种成熟的构建方式外，第三种是探索使用 AI 算法构建数据集，通过大语言模型或其他生成式 AI 技术，高效创建多样化的测试例，包括复杂场景和对抗样本，从而进一步提升数据集的覆盖范围，但需要额外关注生成的数据集的质量。

将以上方式结合，便构成了驱动数据质量持续提升的数据飞轮。在这个模式下，我们通过海量的业务数据和客户数据构评估 AI 原生应用，同时将评估过程中发现的失败案例和用户反馈的数据，进行重点采集、清洗和标注。这样不仅能丰富和强化评估数据集，还能为后续模型的微调提供了高质量的数据。这些真正反映业务挑战的核心数据资产，是整个 AI 原生应用自我进化和持续优化的关键。

2.10.2 明确评估目标

为了实现持续性的评估和优化，通常将复杂的 AI 原生应用评估进行分类，形成一个多层次的评估矩阵。具体而言，评估可以分解为以下几个关键层面：

- 语义评估从文本中提取实体、格式、抽象语义（如意图、情绪、主题）等多层次信息，并生成相关问题以深化理解。例如，对 Prompt 进行语义评估时，分析其是否含歧义、信息缺失或逻辑矛盾。
- Rag 评估则针对从知识库中召回语料的质量进行多维度衡量，包括检索准确性（找到最相关的知识片段）、生成内容可靠（回答不偏离事实）、重复性及多样性等指标。
- 工具调用评估针对需要执行动作的 AI 原生应用，评估其工具调用的合理性与效率。对这些原子能力的评估，是确保 Agent 应用可靠地与外部世界交互、完成实际任务的基础。
- 最后，在所有组件评估之上，是端到端的 Agent 评估。它从宏观视角出发，不纠结于具体步骤的详情，只关注 Agent 是否最终、高效地解决了用户的原始问题，并带来满意的整体体验。

2.10.3 搭建完整的自动化评估系统进行评估

自动化评估系统需要使用高质量的数据集，优秀的裁判模型，匹配的评估模版，对评估目标进行有效的评估。

- 数据飞轮驱动的高质量数据集是评估系统最重要的数据资产。
- 评估的有效性取决于裁判模型的能力上限。优先使用高阶的裁判模型对被测系统输出进行质量评分，或针对裁判模型本身进行优化。针对特定任务维度，还可以引入专业化模型辅助判断。比如利用自然语言推理模型评估生成内容的事实一致性，通过嵌入模型计算语义相似度，以衡量回答相关性。
- 评估模版与评估算子的设计需与评估目标相匹配，实现模块化、可配置的评估能力。例如，在 Prompt 评估中主要关注识别模糊、歧义或结构不良的输入；在 RAG 评估中主要衡量检索相关性与生成忠实度；在 Tools 评估中，则通过参数结构校验与执行结果语义解析，验证工具调用的准确性与理解能力。

自动化评估系统还需要覆盖完整的 AI 原生应用生命周期。评估的执行应该建立离线与在线相结合的机制。

- 离线评估面向应用迭代周期，在开发测试阶段对 Agent 在典型任务、边缘案例及对抗性输入下的表现进行全面测试，支持版本间的横向对比与回归分析，确保新版本满足生产上线的要求。
- 在线评估则通过小流量灰度发布，在真实生产环境中结合自动化评估器实时生成性能指标。通过与 A/B Test 框架相结合，在响应时间、业务效果等关键指标上进行综合对比，为上线决策提供数据支撑。

白皮书在独立的第 9 章中深入探讨 AI 原生应用评估的重要意义与完整体系，同时引入全新的评估范式 LLM-as-a-Judge，并详细阐述如何构建一个高效的自动化评估系统，以推动 AI 应用的持续优化与可靠性提升。

2.11 安全

AI 原生应用的开放性、自主性和多模态交互特性显著扩大了系统的安全风险敞口，给应用安全防护体系带来了新的挑战。AI 应用自主决策和执行任务的能力可能会成为攻击者恶意利用，例如通过提示词注入等手段操控 AI 应用行为，而越权访问操作则可能导致数据泄露。同时，若 AI 基础设施存在漏洞，还有可能引发注入、逆向攻击和算力被滥用等威胁。最后，AI 与生俱来的非预期行为以及输出的不可预测性风险，更加加剧了内部治理和合规的挑战。

在模型安全层面，安全威胁贯穿输入、推理与输出全过程。输入层面临对抗样本、提示词注入与恶意文件上传等攻击；推理层存在模型越狱、RAG 知识库爬取与函数调用劫持等风险；输出层则可能生成钓鱼信息、虚假建议或传递隐蔽指令。为此，需部署大模型原生安全护栏，作为连接应用与模型之间的可信中间层，提供覆盖全链路的一站式防护，保障输出内容合法可控，并实现 AIGC 内容可追溯、责任可追，满足监管要求。

数据安全是 AI 应用可信落地的基础。大模型涉及数据采集、传输、存储、访问、使用与删除六大阶段，每个环节都存在泄露、滥用或逆向推断的风险。企业担忧商业秘密被用于二次训练，个人用户关注隐私控制权，而多方参与的数据处理流程使责任认定变得困难。为此，需构建覆盖全生命周期的数据安全保障体系。具体而言，数据收集阶段需完成分类分级与脱敏，防止敏感信息进入训练流程；传输过程采用 HTTPS/TLS 加密与私有网络隔离，确保通信安全；存储环节通过租户隔离与端到端加密保护数据主权；访问控制需要实现精细化权限管理；访问和使用过程中嵌入 AI 安全护栏进行实时过滤；删除阶段支持账号注销后的彻底清除与迁移能力，保障用户数据主权。

身份安全方面，随着非人类身份（NHI）如 API 密钥、AK/SK、OAuth 令牌等大量涌现，身份安全管理面临严峻挑战。传统静态权限模型难以应对 AI 应用动态、复杂的行为模式，易导致凭据泄露与未授权访问。为此，需构建覆盖“事前、事中、事后”的身份安全闭环体系：事前通过异常访问监测识别风险；事中实施动态权限管理，采用即时授权与细粒度访问控制，遵循最小权限原则；事后通过自动化审计清理僵尸凭据。同时，依托密钥管理服务实现凭据的统一托管、自动轮转与细粒度访问控制，并通过 M2M (Machine to Machine) 能力集中管理 NHI 身份，支持动态授权与全生命周期自动化管控，全面提升 AI 环境下身份安全防护水平。

在 AI 应用的全生命周期中，基础设施的安全性直接决定了应用的可靠性、可信度和合规性。从模型训练到推理服务，AI 系统的复杂性、分布式架构以及对海量数据的依赖，使得任何基础设施层面的疏漏都可能引发模型盗用、数据泄露或服务滥用等严重风险。所以需要从构建安全、可控的运行环境，需要从全局安全态势到计算、网络等细节层面，建立多层次防护体系。

综上所述，AI 原生应用的安全防护是一项系统工程，涵盖应用、模型、数据、身份与基础设施五大维度。面对动态、自主、多模态的新一代智能体，传统的边界防御已不再适用，必须转向以“纵深防御、动态检测、全程可控”为核心的综合防护体系。唯有将安全能力深度融入 AI 原生架构的设计、开发与运维全流程，才能真正构筑起可信、可控、可审计的智能应用生态，支撑 AI 技术的可持续发展与规模化落地。

白皮书在独立的第 10 章中全面介绍 AI 安全风险的来源和分类，并从应用、模型、数据、身份，以及系统和网络的安全保护进行展开。

AI 应用开发框架

AI Development Frameworks

CHAPTER

3.1 智能体的定义与主流开发范式

3.1.1 什么是智能体

我们可以将智能体（Agent）理解为一个具备自主理解、规划、记忆和工具使用能力的数字化实体。想象一个高度智能的个人助理，你只需告诉他“帮我规划一次去北京的周末旅行”，他就能自主完成以下任务：

- **感知（Perception）：** 理解用户的自然语言指令。
- **规划与推理（Planning & Reasoning）：** 借助模型推理能力，将任务拆解为查询往返机票、筛选酒店、规划景点路线、预估预算等子任务。
- **记忆（Memory）：** 记住你之前的偏好，比如喜欢靠窗的座位、偏爱的经济型酒店。
- **工具（Tool）：** 调用外部工具，如机票预订 API、酒店查询系统、地图服务等，来执行这些子任务。
- **反馈与迭代（Feedback & Iteration）：** 将规划好的行程草案反馈给你，并根据你的修改意见进行调整，最终完成预订。

智能体让 AI 从一个只会内容生成的语言模型，进化成一个具备自主规划能力的行动者。

3.1.2 智能体的主流开发范式

随着智能体技术的不断成熟，业界出现了多种有效的智能体开发范式。根据解决问题的场景与复杂度，我们可以将主流的智能体应用开发范式大致分为4种：简单 LLM 应用、单智能体（Single Agent）、工作流（Workflow）以及多智能体系统（Multi-Agent）。

- **简单 LLM 应用：** 我们将直接调用模型 API 实现内容生成的应用归类为简单 LLM 应用，它们完全依赖模型服务实现内容生成。大家应该还记得，在大模型刚刚面世之时，各类 Generative AI 概念的 Chat 应用大部分都是这类应用。
- **单智能体：** 相比于直接调用模型 API 的应用，单智能体应用是一种 Augmented LLM 应用，它的核心理念是为普通 LLM 应用增加 RAG、Tool、Memory 等，让模型具备与特定环境交互的能力。
- **工作流：** 对于一些复杂的业务应用，使用单智能体的架构效果欠佳。为此，我们可以将应用拆分开来，形成多个独立子智能体（每个子智能体是一个 Single Agent）的架构，再将这些独立的子智能体按照预定义的流程编排起来，这就是 Workflow 范式的基本定义。
- **多智能体系统：** 和 Workflow 非常类似，都是遵循将复杂智能体应用拆分成多个独立子智能体的理念，但相比于 Workflow 的确定性流程，Multi-Agent 采用的是模型驱动的、对话式的流程编排，各个子智能体在协作上具备更多的自主决策权（通常通过模型决策）。

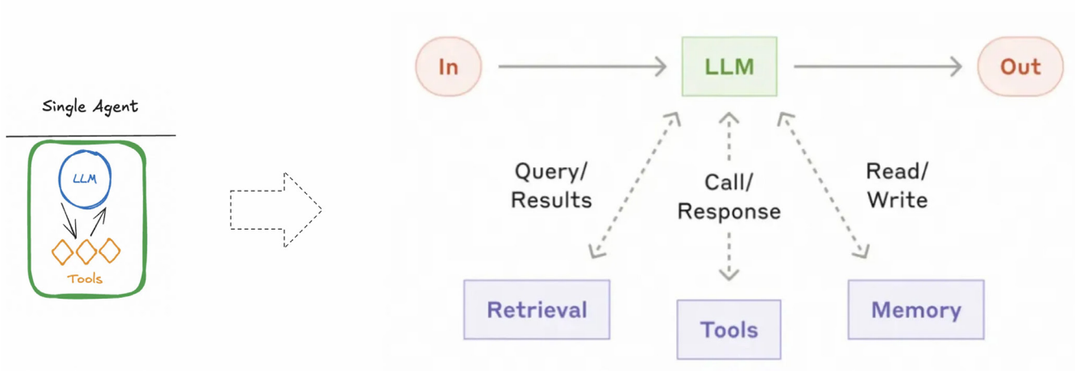
接下来，我们将主要围绕单智能体、工作流、多智能体系统这三种核心 Agent 开发范式展开，分别讲解它们的核心设计理念，为我们后续章节的智能体开发实践提供理论基础。

3.1.3 单智能体

前面我们提到了 Augmented LLM Application 的概念，这是为了弥补大模型预训练特性在应用开发上的局限性：

- 无法获取最新的数据、私有领域数据。
- 无法对本地工具发起调用。
- 无状态，因此无法形成有效的历史记录。

典型的单智能体架构如下。我们将这种架构称之为最简单的智能体应用，同时也是最高级的智能体应用。



图片出自 Anthropic 博客《Building Effective AI Agents》

说它简单，是因为我们不用做过多的智能体拆分编排等工作，开发者只需要把工具等上下文都给到模型，一切都由模型驱动。说它高级，是因为整个应用由模型作为大脑来驱动和决策，不论任何问题，我们预期模型都能够正确的推理、选择合适的工具、检索到正确的上下文，最终生成复合预期的答案。

单智能体架构的高级性非常依赖底层模型能力，然而，在当前的实践中，我们发现当前的模型能力是受限的（不够聪明、上下文窗口等），单智能体模式面临很多的挑战。为此，业内才设计出工作流、多智能体系统等模式，在工程上做更多事情，来弥补模型能力的不足。

以下是单智能体架构下的一些典型问题：

- 如果一个 Agent 包含有太多的可用 Tools，模型会在决策工具选择时无所适从，效果变差。
- 多轮执行下来，消息上下文会变得很大，消耗大量的 Token且影响模型当前专注度。
- 很多复杂的任务需要很多个步骤，通常在某些环节还需要专业领域技能支持，比如科学计算、深入研究、写代码、绘画、任务规划等。
- 单 Agent 在复杂任务场景下的可维护性会变差。

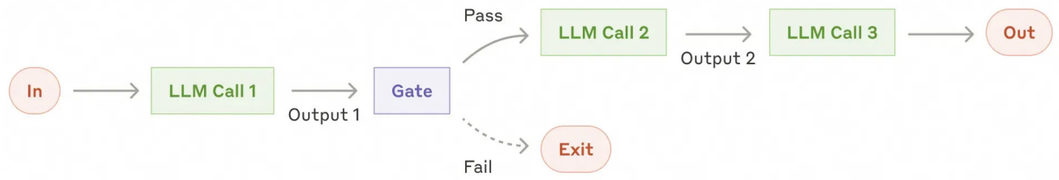
3.1.4 工作流

工作流是一种编排模式，需要开发者将一个复杂任务拆解为一系列有序步骤，这些步骤之间的调用关系和条件分支在设计阶段就明确好。例如，接下来要说到的链式工作流（Chain）和路由工作流（Routing），它们都强调通过规则、逻辑或预设分支，把任务从输入到输出，组织成可控的流程。

工作流的特点是确定性强、可调试性高、可控性好。你可以很清楚地看到“先生成大纲 → 检查→写文章”，或者“先判断输入类别 → 路由到对应模块”，因此非常适合流程清晰、可拆解的任务，以及对稳定性要求高的场景。简单来说，工作流像是固定的流水线，强调任务执行的透明度和安全性。

1、Chain - 链式工作流

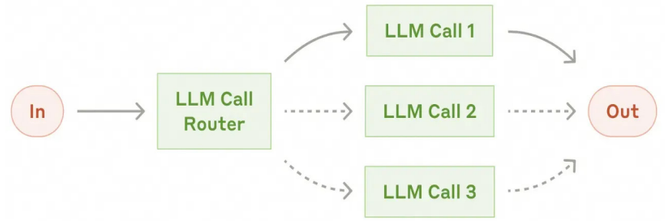
链式工作流是一种将复杂任务分解为一系列较小、清晰的子任务的方式。每一步由一个独立的 LLM 调用完成，并将上一阶段的输出作为下一阶段的输入。典型应用包括先生成营销文案，再翻译成另一种语言，或先写文章大纲、检查大纲是否满足要求，再基于大纲完整撰写文章。



图片出自 Anthropic 博客《Building Effective AI Agents》

2、Routing - 路由工作流

路由工作流通过对输入进行分类，将它们分派到专门设计的下游任务或处理路径（通常也叫做意图识别），以实现关注点分离与更精准的响应。例如，在客服系统中，可以将普通咨询、退款请求、技术支持等不同类型的用户问题路由到不同的处理流程与工具，或者在成本和速度优化中，根据问题复杂度，将简单请求分发给小模型，而复杂或罕见问题则交给更强大的模型处理。

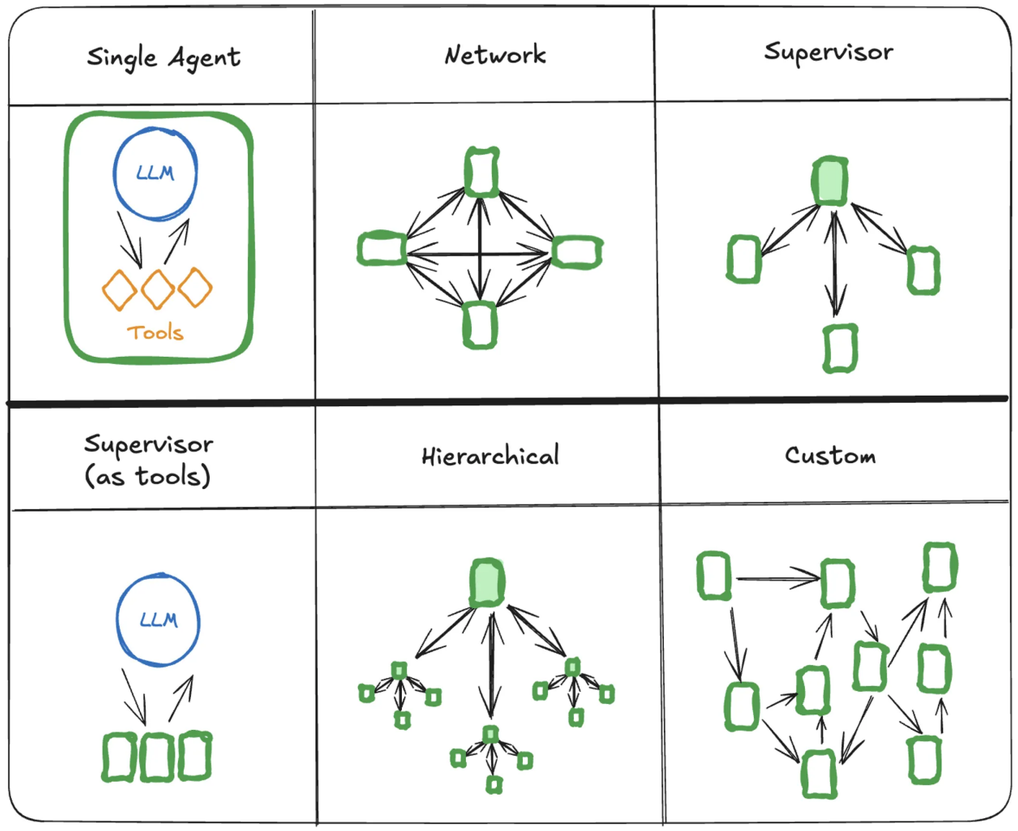


图片出自 Anthropic 博客《Building Effective AI Agents》

3.1.5 多智能体系统

多智能体系统更偏向一种自治协作模式。在这种模式下，系统由多个具备一定自治能力的 Agent 组成，每个 Agent 有自己的角色、能力或工具使用范围。它们之间可以通过消息或上下文进行交互，协同完成复杂目标。

这种方式更接近团队合作，而不是预先定义好的流程。优点是灵活性和适应性强，尤其在任务边界不清晰、需要动态规划时表现突出，比如研究型问答、复杂数据分析或跨领域问题求解。但多智能体系统的挑战在于难以预测和调试，因为不同 Agent 的交互和决策路径可能高度动态，不像工作流那样可控。



图片出自 Anthropic 博客《Building Effective AI Agents》

3.2 开发一个简单的智能体

3.2.1 智能体框架选型

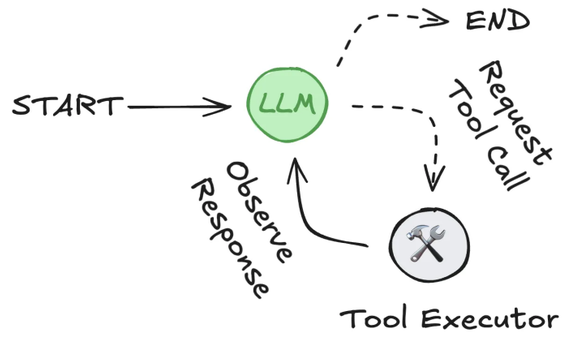
智能体火爆初期，开发生态主要以 Python 为主导，并且涌现了多个广受欢迎的开源框架，从早期的 LangChain、LlamaIndex、AutoGen、CrewAI 到近期开源的 ADK、Strands 等，国内则有阿里云开源的 AgentScope 等。

随着智能体从概念、Demo 走向企业级生产实践，框架与生态快速从 Python 拓展到其他语言生态，这其中包括在国内占据主导地位的 Java 语言，在这一背景下发展出 Spring AI Alibaba 等多种 Java 语言实现的开源框架，为 Java 应用开发者提供了快速构建智能体所需的核心能力。

笔者正好是 Java 的开发者，后面我们就用 Spring AI Alibaba 为例，深入探讨如何基于开源框架快速构建一个功能完备的智能体。

3.2.2 Spring AI Alibaba 中的 Agent 定义

在 Spring AI Alibaba 中，我们把 Agent 定义叫做 React Agent，从它的名字可以看出，它就是一个典型 ReAct 模式的一个具体实现。ReAct 是一种强大的智能体设计模式，它通过将推理（Reasoning）和行动（Action）交错进行，形成一个动态的“思考-行动-观察”循环，来解决复杂问题。您可以在本白皮书第 2 章的框架节，查看业内其他主流的 Agent 设计思想。



以上是 ReAct 循环的具体步骤如下：

- **思考（Thought）：**当智能体接收到一个任务时，它首先不会直接行动。相反，它会进行一次内部的思考。LLM 会分析当前的任务目标、已有的信息和可用的工具，然后生成一个关于下一步应该做什么的推理过程或计划。这个思考过程通常是自然语言文本，解释了它的决策逻辑。
- **行动（Action）：**基于思考的结果，智能体决定调用一个具体的外部工具（如搜索引擎、数据库查询 API 或自定义函数）。它会生成一个结构化的指令，明确指出要调用的工具名称和所需的参数。
- **观察（Observation）：**应用程序执行该工具调用，并将返回的结果（如搜索结果、API 响应或函数返回值）作为观察信息反馈给智能体。
- **循环迭代：**智能体接收到这个观察结果后，会将其与之前的“思考”和任务目标结合起来，进入新一轮的思考阶段。它会评估上一步行动的结果是否使其更接近最终目标，并据此规划下一步的行动。

这个循环会一直持续下去，直到智能体判断任务已经完成，然后它会生成最终的答案并结束循环。通过这种方式，ReAct 智能体能够动态地分解任务、利用外部工具获取信息、并根据中间结果不断修正自己的计划，从而处理比简单 LLM 应用复杂得多的任务。

3.2.3 示例代码与 Agentic API

1、快速定义一个简单的 Agent

在创建任何智能体之前，您首先需要明确定义它的身份和目标。这构成了智能体的基础配置，决定了它的核心功能以及在多智能体系统（接下来章节会展开）中如何与其他成员协作。

```java
ReactAgent agent = ReactAgent.builder()
        .name("single_agent")
        .model(chatModel)
        .description("A simple agent that can answer questions.")
        .build();

```

- **Name (名称，必填)：**每个智能体都需要一个唯一的字符串标识符。这个 name 在内部操作中至关重要，尤其是在多智能体系统中，因为智能体需要通过名字来互相引用或委托任务。建议选择一个能清晰反映其功能的描述性名称（如 writer_agent、reviewer_agent），并避免使用系统保留名称，如 user 。
- **Description (描述，可选，但在多智能体中强烈推荐)：**为智能体的能力提供一个简明扼要的摘要。这个描述主要供其他大语言模型智能体使用，以判断是否应将某个任务路由给当前智能体。因此，描述需要足够具体，以便与其他智能体区分开来（例如，应使用处理关于当前账单的查询，而不仅仅是账单智能体）。
- **Model (模型，必填)：**指定驱动该智能体进行推理的底层大语言模型。这是一个字符串标识符，例如 "qwen-max" 。模型的选择将直接影响智能体的能力、成本和性能。

2、引导 Agent 的行为（设置 Prompt 提示词）

```java
// Example: Adding instructions
ReactAgent capitalAgent =
    ReactAgent.builder()
        .model("qwen-max")
        .name("capital_agent")
        .instruction(
            """
            You are an agent that provides the capital city of a country.
            When a user asks for the capital of a country:
            1. Identify the country name from the user's query.
            2. Use the `get_capital_city` tool to find the capital.
            3. Respond clearly to the user, stating the capital city.
            Example Query: "What's the capital of {country}?"
            Example Response: "The capital of France is Paris."
            """)
        .build();
```

在框架中，instruction 参数可以说是引导 ReactAgent 行为最关键的配置，可以理解为我们常说的 system prompt 参数，用于告知智能体以下信息：

- **核心任务或目标：**明确智能体需要完成什么工作。
- **性格或角色 (Persona)：**定义智能体的交流风格，例如，“你是一个乐于助人的助手”或“你是一位言辞诙谐的海盗”。
- **行为约束：**为其行为设定限制，例如，“只回答关于X的问题”或“绝不能透露Y信息”。
- **如何及何时使用工具 (Tools)：**您需要在此解释每个工具的用途以及在何种情况下应该调用它，作为工具自身描述的补充说明。
- **期望的输出格式：**指定其最终响应的格式，例如，“以 JSON 格式回应”或“提供一个项目符号列表”。

以下是关于如何编写高效 instruction 的技巧，如要了解更多，可参考第四章上下文工程。

- **清晰具体：**避免模糊不清的表述。明确地陈述期望的动作和结果。
- **使用 Markdown：**对于复杂的指令，可以使用标题、列表等 Markdown 格式来提高可读性。
- **提供示例 (Few-Shot Learning)：**对于复杂的任务或特定的输出格式，直接在指令中包含一两个示例会非常有帮助。
- **引导工具使用：**不要仅仅罗列工具有哪些，更要解释智能体何时 (when) 以及为何 (why) 应该使用它们。

3、更多高级设置：为 Agent 配置工具、模型参数等

工具赋予 ReactAgent 获得模型预训练数据之外的能力，可以与本地私域数据、业务系统很好的结合，例如执行计算、获取实时数据或执行特定的操作。

```java
ReactAgent agent = ReactAgent.builder()
        .name("single_agent")
        .model(chatModel)
        .description("A simple agent that can write articles.")
        .instruction("You are a helpful article writer assistant, please write articles of the topics given by user.")
        .tools(List.of(DemoTool.getToolCallback()))
        .chatOptions(DashScopeChatOptions.builder().withTemperature(0.7).build())
        .build();
```

Tools (可选)：为智能体提供一个可供其使用的工具列表，列表中的每一项可以是：

- 一个原生函数或方法 (包装为 ToolCallback)。在 Spring AI Alibaba 中，您必须使用注解或 API 等方式创建工具，比如使用 FunctionToolCallback.builder().tool(your-tool).build() 来显式包装您的方法。
- 另一个智能体的实例 (AgentTool)，这使得智能体之间的任务委托成为可能（详见多智能体章节)

大语言模型会根据当前的对话内容和自身的指令，利用工具的函数/工具名称、描述 (来自文档字符串或 description 字段) 以及参数结构 (Schemas) 来决定调用哪一个工具。

3.3 工作流与多智能体

在上一节中，我们已经掌握了如何开发一个简单的智能体，如果是应对一个类似智能问答客服系统的场景的话，这些知识已经足够了，但通常我们要面临的实际业务场景比这个要复杂的多，接下来就让我们学习如何用前文提到的多智能体架构解决这类问题。

在多智能体系统开发过程中，有以下关键内容需要考量：

- 从一个简单的双智能体系统入手，例如先定义一个编写者，再定义一个评论者，让它们协作完成一篇短文。这有助于理解智能体交互的基础。
- 为每个智能体定义独特的角色、能力和明确的职责，设计清晰的通信协议和工作流程，规定它们如何协作、传递信息和解决冲突。
- 利用 AgentScope、Google ADK、LangGraph 或 Spring AI Alibaba 等主流框架来简化开发。这些框架提供了智能体通信、状态管理和任务编排的工具，能帮助您快速搭建系统。

开发的关键在于迭代优化，通过不断测试、监控智能体行为并调整其协作逻辑，逐步扩展系统复杂性，最终构建出强大的、能够自主解决复杂问题的多智能体应用。

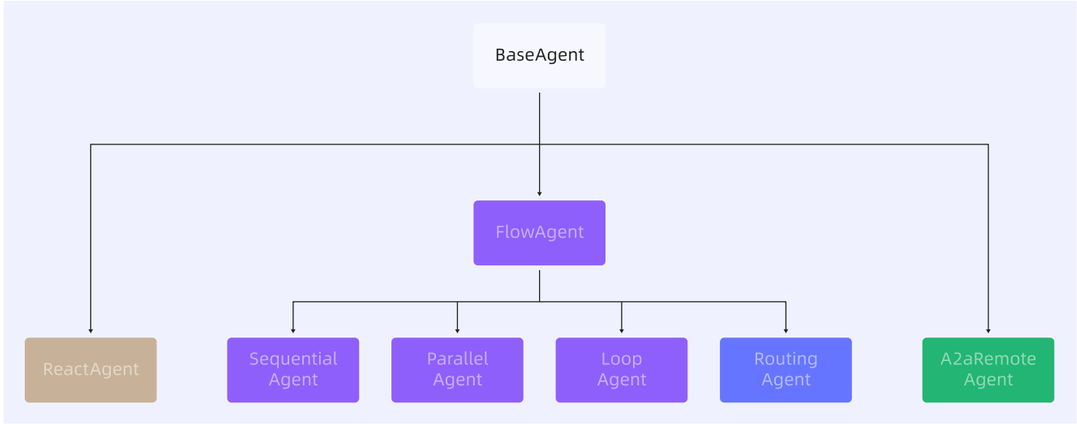
与上一节一样，我们继续使用业界流行的 Java 开源框架 Spring AI Alibaba 为例来讲解如何开发多智能体，对于其他语言框架，其思路和步骤是类似的。

3.3.1 Spring AI Alibaba 中的多智能体类型

以下是 Spring AI Alibaba 中的 Agent 定义与类继承关系，总体上分为三种 Agent 类型：

- **ReactAgent：**框架中对于 Agent 的基本定义，多智能体通常是指如何编排多个 ReactAgent 互相协作解决复杂问题。
- **FlowAgent：**FlowAgent 中包含有多个 ReactAgent，它们按照特定的流程相互协作。
  - SequentialAgent，串行依次执行的多个智能体的流程
  - ParallelAgent，可并行执行的多个智能体的流程
  - LoopAgent，循环执行多个智能体的流程，直到满足某个特定条件退出
  - LlmRoutingAgent，由大模型决策的执行哪个智能体

- **A2ARemoteAgent：**由于 A2ARemoteAgent 属于分布式 Agent 范畴，本章节会有独立段落展开。

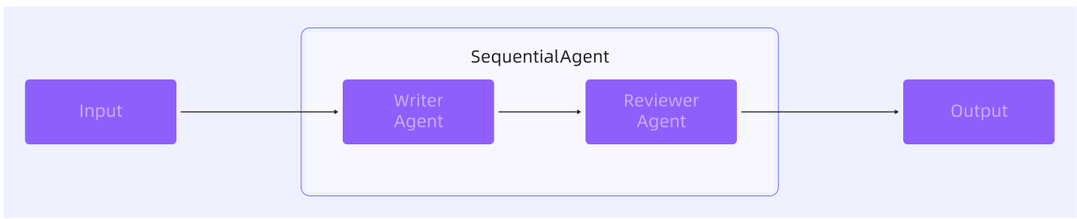


以下表格从分析了几种预置 Agent 类型的特点：

	ReactAgent	FlowAgent	MultiAgent
核心特性	推理、生成、工具使用	控制多个智能体流程	控制多个智能体流程
流程编排	无	预先定义好的流程，如传串行、并行、回环等	支持大模型决策的流程控制，如 LlmRouting、Supervisor 等
确定性	具有较高不确定性	具有较高确定性	介于 ReactAgent 和 FlowAgent 之间，具有较高不确定性

3.3.2 工作流 - SequentialAgent

SequentialAgent 的典型架构如下，Agent 依次执行，上一个 Agent 的输出会作为下一个 Agent 的输入。



以下 Spring AI Alibaba 示例代码中，我们将开发一个文章协作助手的智能体。它包含 writer_agent、reviewer_agent 两个子智能体，分别定义如下。

```java
ReactAgent writerAgent = ReactAgent.builder()
        .name("writer_agent")
        .model(chatModel)
        .description("可以写文章。")
        .instruction("你是一个知名的作家，擅长写作和创作。请根据用户的提问进行回答。")
        .outputKey("article")
        .build();

ReactAgent reviewerAgent = ReactAgent.builder()
        .name("reviewer_agent")
        .model(chatModel)
        .description("可以对文章进行评论和修改。")
        .instruction("你是一个知名的评论家，擅长对文章进行评论和修改。对于散文类文章，请确保文章中必须包含对于西湖风景的描述。")
        .outputKey("reviewed_article")
        .build();

```

然后，我们使用 SequentialAgent 将 writer_agent、reviewer_agent 依次串联起来。

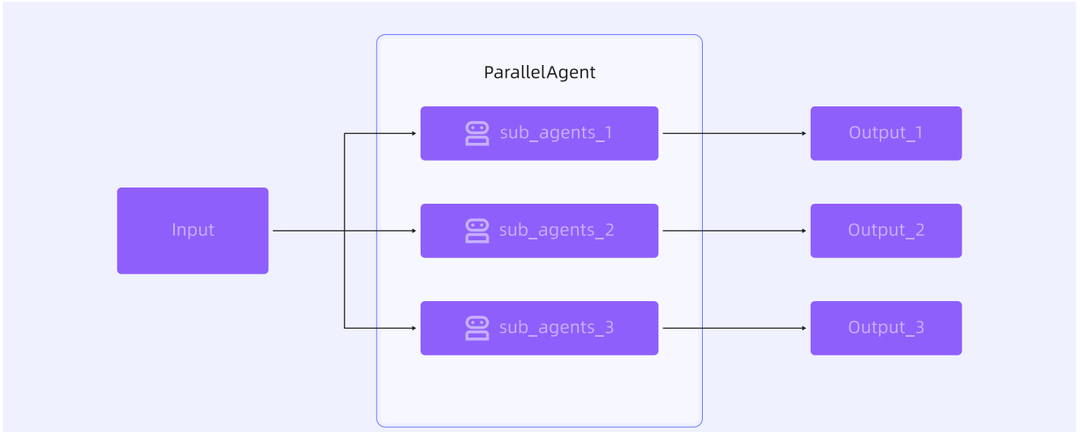
```java
SequentialAgent blogAgent = SequentialAgent.builder()
        .name("blog_agent")
        .state(stateFactory)
        .description("可以根据用户给定的主题写一篇文章，然后将文章交给评论员进行评论，必要时做出修改。")
        .inputKey("input")
        .outputKey("topic")
        .subAgents(List.of(writerAgent, reviewerAgent))
        .build();

Optional<OverAllState> result = blogAgent.invoke(Map.of("input", "帮我写一个100字左右的散文"));
```

在这里，我们将 SequentialAgent 类型的 BlogAgent 叫做 RootAgent，它作为我们 Agent 调用的总入口。

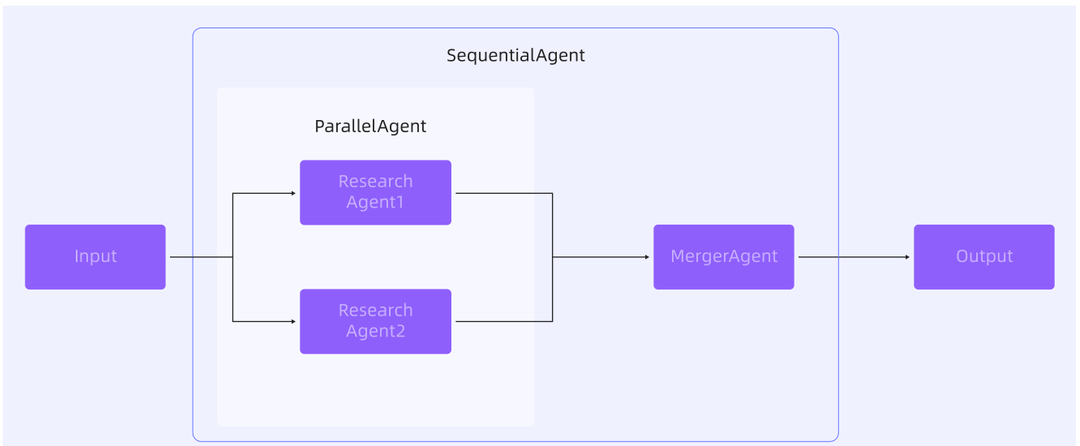
- **state（作为 RootAgent 必须）：** 在 Spring AI Alibaba 中，我们要求 state 必须在 RootAgent 中进行明确声明，state 中定义在整个 Agent 链路中用到的所有参数（所有 outputKey、inputKey）。
- **subAgents（作为父 Agent 必须）：** 定义父 Agent 关联的子 Agent。

3.3.3 工作流 - ParallelAgent

ParallelAgent 是并行执行的工作流模式。



以下 Spring AI Alibaba 示例代码中，我们将开发一个智能搜索助手的智能体，示例总体架构如下：



它包含 ResearchAgent1（擅长 AI Agent）、ResearchAgent2（擅长 Microsercies） 两个子搜索智能体，分别擅长特定领域的搜索。

```java
ReactAgent researcherAgent1 = ReactAgent.builder()
        .name("researcherAgent1")
        .model(chatModel)
        .description("Search AI Agent trends.")
        .instruction("""
                You are an AI Research Assistant specializing in AI Agent.
                Research the latest developments in 'electric vehicle technology'.
                Use the Google Search tool provided.
                Summarize your key findings concisely (1-2 sentences).
                Output *only* the summary.
                """)
        .outputKey("research1_summary")
        .build();

ReactAgent researcherAgent2 = ReactAgent.builder()
        .name("researcherAgent2")
        .model(chatModel)
        .description("Search Microservice trends")
        .instruction("""
                You are an AI Research Assistant Microservices.
                Summarize your key findings concisely (1-2 sentences).
                Output *only* the summary.
                """)
        .outputKey("research2_summary")
        .build();
```

我们使用 ParallelAgent 工作流将两个子 Agent 关联起来，如下 ResearchAgent 定义：

```java
ParallelAgent researchAgent = ParallelAgent.builder()
        .name("researchAgent")
        .description("Runs multiple research agents in parallel to gather information")
        .outputKey("topic")
        .subAgents(List.of(researcherAgent1, researcherAgent2))
        .build();
```

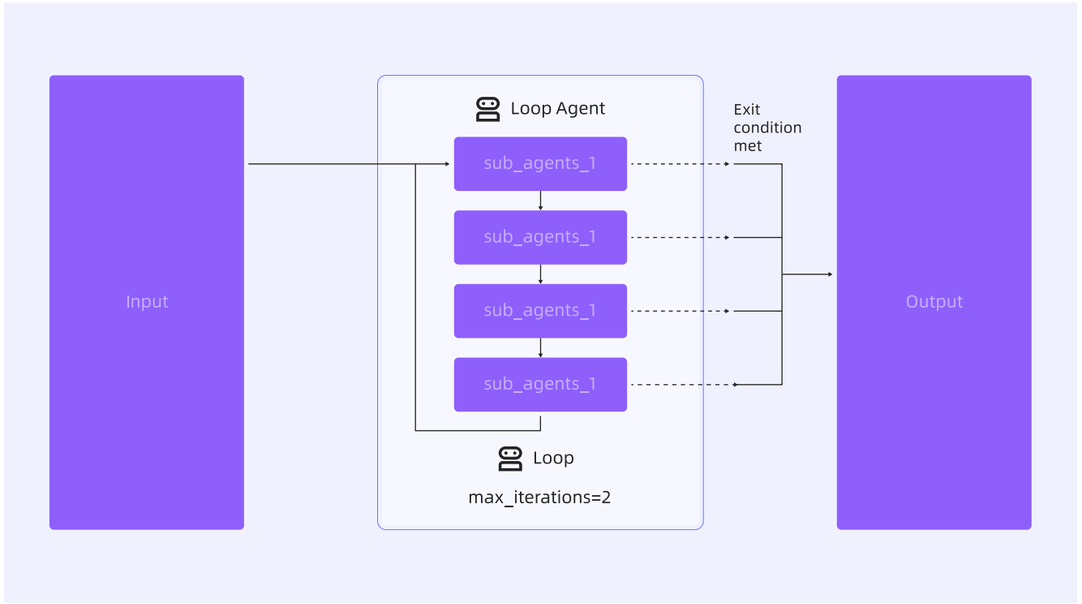
两个并行子 Agent 的输出需要进行合并汇总，因此我们定义一个 MergerAgent，最后，通过一个 SequentialAgent 将 ResearchAgent 和 MergerAgent 串联到一起。

```java
ReactAgent mergerAgent = ReactAgent.builder()
        .name("mergerAgent")
        .description("Merge the summaries")
        .instruction("""
                Merge the summaries from both research agents into a single concise summary.
                """)
        .outputKey("final_research_summary")
        .build();

SequentialAgent researchPipelineAgent = SequentialAgent.builder()
        .name("researchPipelineAgent")
        .state(stateFactory)
        .inputKey("input")
        .outputKey("topic")
        .subAgents(List.of(researchAgent, mergerAgent))
        .build();
```

通过以上示例我们可以看到，SequentialAgent 与 ParallelAgent 之间可以互相作为 subAgent 嵌套组成更复杂的工作流。

3.3.4 工作流 - LoopAgent

LoopAgent 是另一个工作流模式的智能体，它在循环中执行其子代理（即 loop）。它在指定的迭代次数内重复运行一系列代理，或者直到满足终止条件。

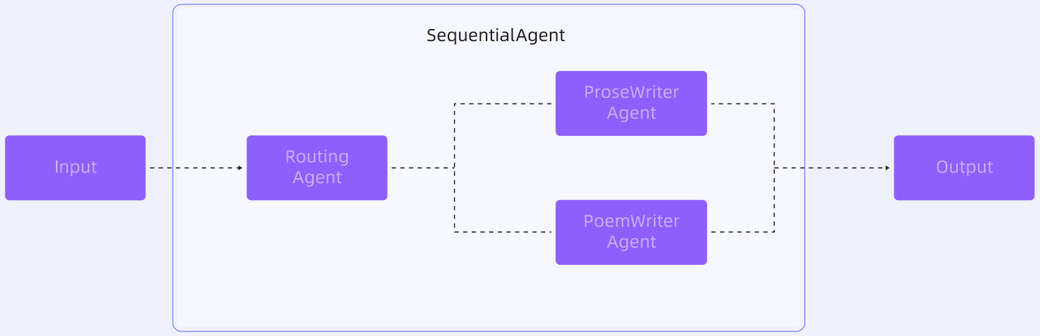


以下是 Spring AI Alibaba 中 LoopAgent 实现的代码示例。

```java
LoopAgent loopAgent = LoopAgent.builder()
        .name("loop_agent")
        .description("循环执行3次")
        .inputKey("loop_input")
        .outputKey("loop_output")
        .state(() -> Map.of("loop_output", new AppendStrategy(), "loop_input", new ReplaceStrategy()))
        .loopMode(LoopAgent.LoopMode.COUNT)
        .loopCount(3)
        .subAgents(List.of(writerAgent, reviewerAgent))
        .build();

```

3.3.5 多智能体系统 - LlmRoutingAgent

以下是 LlmRoutingAgent 模式的基本架构图和工作原理，相比于前面讲到的 SequentialAgent、ParallelAgent 等工作流模式，LlmRoutingAgent 最大的区别在于它通过模型决策下一个子智能体走向。



如上图所示，RoutingAgent 内置的 Prompt 定义如下，它会根据当前 RoutingAgent 的职责、所有可用子 Agent 的能力、当前用户的请求，来使用 LLM 智能决策下一个流程节点。

```LaTeX
You are responsible for task routing in a graph-based AI system.
Your capability seen by the user is: 可以根据用户给定的主题写文章或作诗。

There're a few agents that can handle the task given to you, you can
delegate the task to one of the following.The agents ability are listed
in a 'name:description' format as below:
- prose_writer_agent: 可以写散文文章。
- poem_writer_agent: 可以写现代诗。

Return the agent name to delegate the task to.
```

具体示例代码如下，可在下方此查看以上代码片段的完整示例。首先，定义两个子智能体，分别是写散文的 prose_writer_agent 和写现代诗的 poem_writer_agent。

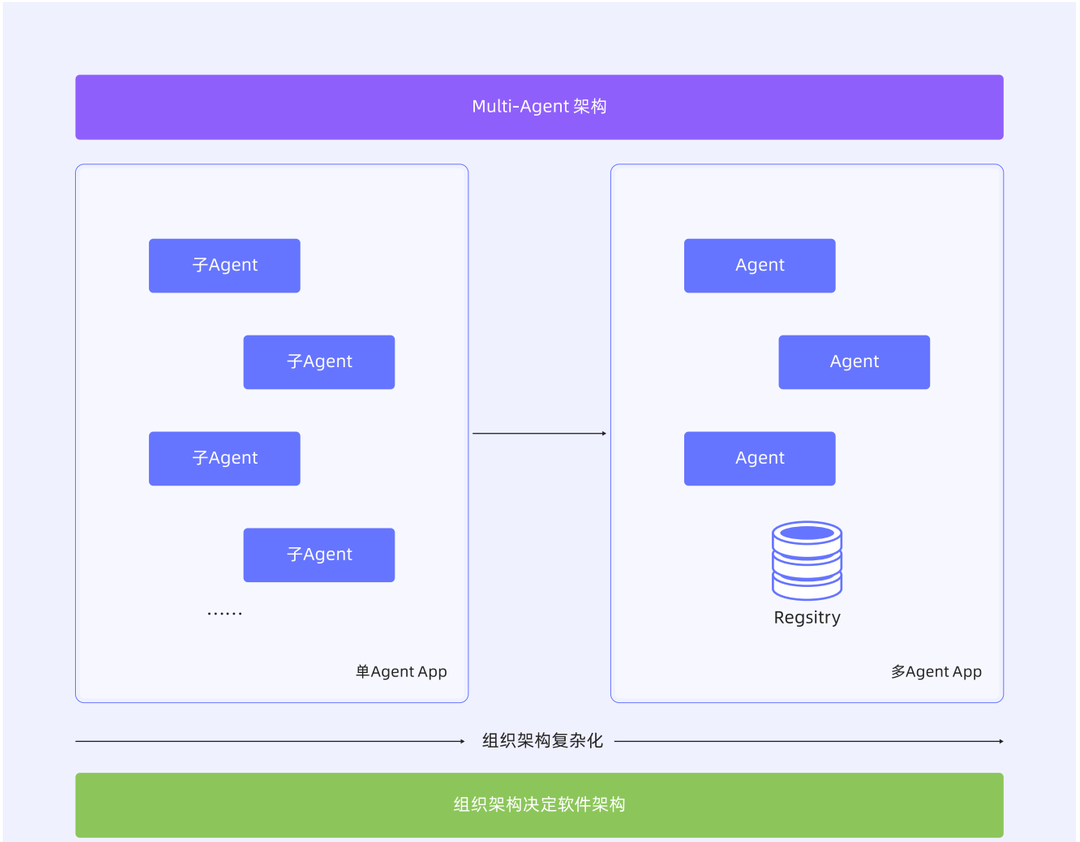

```Java
ReactAgent proseWriterAgent = ReactAgent.builder()
        .name("prose_writer_agent")
        .model(chatModel)
        .description("可以写散文文章。")
        .instruction("你是一个知名的作家，擅长写散文。请根据用户的提问进行回答。")
        .outputKey("prose_article")
        .build();

ReactAgent poemWriterAgent = ReactAgent.builder()
        .name("poem_writer_agent")
        .model(chatModel)
        .description("可以写现代诗。")
        .instruction("你是一个知名的诗人，擅长写现代诗。请根据用户的提问进行回答。")
        .outputKey("poem_article")
        .build();
```

定义 LlmRoutingAgent，API 使用上与 SequentialAgent 等基本相同。

```Java
LlmRoutingAgent blogAgent = LlmRoutingAgent.builder()
        .name("blog_agent")
        .model(chatModel)
        .state(stateFactory)
        .description("可以根据用户给定的主题写文章或作诗。")
        .inputKey("input")
        .outputKey("topic")
        .subAgents(List.of(proseWriterAgent, poemWriterAgent))
        .build();
```

3.4 从单进程到分布式部署

通过本章前3节的介绍，我们已经掌握了从单智能体到多智能体的开发模式与实践方案，但讨论都还是限定在单个应用、单个进程范围内的，那如何将智能体拓展到分布式场景那？



在开始讲解具体技术方案之前，我们先了解下为什么需要从单个智能体应用走向分布式？如果我们回顾微服务时代，几乎所有的企业都经历了从单体架构到微服务架构的技术演进，这其中有企业组织架构、文化方面的非技术因素，也有可扩展性、维护复杂度、性能等技术因素。因此，我们相信随着智能体在企业内的广泛应用，智能体也会走向分布式，接下来让我们一起来详细探讨分布式智能体的开发与部署方案。

在单个的智能体应用内，随着拆分的 Multi-Agent 增加、子智能体（Sub-Agent）的更新，发布会伴随组织架构的膨胀，沟通成本的上升导致 AI 应用的迭代效率及稳定性降低。从而引发 Multi-Agent 的架构体系向微服务架构学习和转变：通过将 Sub-Agent 进行独立部署和维护，同时多个 Sub-Agent间 通过远程调用代替内存调用的方式，实现各个 Sub-Agent 独立迭代和维护，避免不同的 Sub-Agent 节奏不同导致的效率和稳定性降低问题。

洞察到这个趋势之后，Google 迅速跟进，在开源社区中推出分布式智能体之间的通信协议，Agent2Agent 协议（以下简称为 A2A 协议）。本文将对 A2A 协议的定义和运行原理做简要介绍。

3.4.1 什么是 A2A 协议

A2A 协议是一项开放标准，旨在解决人工智能快速发展中面临的核心挑战：如何让由不同团队开发、采用不同技术构建、归属于不同组织的 AI 智能体实现高效通信与协作？

随着 AI 智能体日益专业化且能力增强，它们需要协同完成复杂任务的需求也愈加迫切。若缺少统一通信协议，将这些异构智能体整合为统一的用户体验将面临巨大的工程难题。每项集成都可能成为定制化的点对点解决方案，导致系统难以扩展、维护和迭代。

1、A2A 协议中的角色

A2A 协议中主要包含以下关键角色，每个角色分别承载协议中的不同作用：

- **用户（User）：** 发起请求或目标的最终使用者（人类或自动化服务），需借助智能体协助完成任务。
- **A2A 客户端（Client Agent）：** 代表用户向远端智能体发起操作或信息请求的应用、服务或其他 AI 智能体。客户端通过A2A协议发起通信，不依赖具体远端智能体的实现细节。
- **A2A 服务端（Remote Agent）：** 实现了 A2A 协议 HTTP 端点的 AI 智能体或智能系统，负责接收请求、处理任务并返回结果或状态。从客户端视角看，远端代理以“黑盒”形式运行——客户端无需感知其内部逻辑、记忆或工具集。

2、A2A 协议中的元素

A2A 协议中主要包含以下几种元素，这几种元素内容在单次的 A2A 工作流程中将会一次或多次的出现：

- **Agent Card（智能体卡片）**
  - 一个 JSON 格式的元数据文档，通常位于固定URL（如 /.well-known/agent-card.json ），用于描述 A2A 服务端的能力。
  - 包含智能体身份（名称、描述）、服务端点 URL、版本号、支持的 A2A 功能（如流式传输或推送通知）、提供的技能列表、默认输入/输出模式及鉴权要求。
  - 客户端通过解析该文档发现智能体，并获取安全交互的规范。

- **Task（任务）**
  - 当客户端向智能体发送请求时，若需通过状态化流程完成任务（如生成报告、预订航班），智能体会创建唯一任务 ID 并维护其生命周期（提交、处理中、需输入、完成、失败）。
  - 任务支持多次客户端与服务端的交互，涵盖复杂场景的多轮通信。

- **Message（消息）**
  - 客户端与服务端单次通信的基本单元，包含以下属性：
    - 角色： user （客户端发送）或 agent （服务端发送）。
    - 消息ID：由发送方生成的唯一标识。
    - 内容块（Part）：承载实际数据的结构化内容（如指令、问题、状态更新）。
  - 主要用于非任务产出的普通通信场景，即同步请求的单次调用回复。

- **Artifact（产出物）**
  - 任务完成后返回的实体化结果（如文档、图片、结构化数据），由多个内容块（Part）构成，支持流式增量传输。
  - 建议任务完成时通过该对象返回最终输出。

- **Part（内容块）**
  - Message（消息）或 Artifact（产出物）中的最小内容单元，支持多类型数据：
    - 文本块（TextPart）： 纯文本内容。
    - 文件块（FilePart）： 文件数据（通过 Base64 内联或 URI 引用传输），含文件名、媒体类型等元数据。
    - 数据块（DataPart）： 结构化 JSON 数据（如表单参数、机器可读信息）。

3、A2A 协议中的交互机制

A2A 协议中主要会用到3类交互机制，分别为 轮询（Polling），流式传输（Streaming） 和推送通知（Push Notifications）

- **轮询（Polling）**
  - 轮询通过传统的 请求/响应 方式实现。
  - 客户端发起请求（如调用 message/send RPC方法），服务端返回即时响应。
  - 长任务场景：若需状态化长时运行任务，服务端可能返回“处理中”状态，客户端需周期性调用 tasks/get 轮询获取状态更新，直至任务完成或失败。

- **流式传输（Streaming）**
  - 流式传输通过SSE（Server-Send Events）方式实现。
  - 适用场景：需增量生成结果或实时进度反馈的任务。
  - 流程：
    - 客户端通过 message/stream 发起交互。
    - 服务端保持 HTTP 连接开放，持续发送 SSE 事件流（包括 Task 、 Message 、状态变更的 TaskStatusUpdateEvent 或产出物更新的 TaskArtifactUpdate Event ）。
  - 前置条件：服务端需在智能体卡片（Agent Card）中声明流式传输支持能力。

- **推送通知（Push Notifications）**
  - 推送通知通过 WebHoook 方式实现。
  - 适用场景：超长耗时任务或SSE长连接不可行的场景。
  - 流程：
    - 客户端在任务初始化时提供回调 URL（通过 tasks/pushNotificationConfig/set 配置）。
    - 当任务状态变更（完成、失败或需输入）时，服务端向该URL发送异步HTTP POST 通知。

- **前置条件：**服务端需在智能体卡片（Agent Card）中声明推送通知支持能力。

3.4.2 A2A协议的工作流

A2A 协议的工作流程主要分为3个大步骤：

- 发现 A2A Server 的 AgentCard 。
- A2A 授权和权限认证。
- 发起 A2A 请求。

1、发现 A2A Server 的 AgentCard

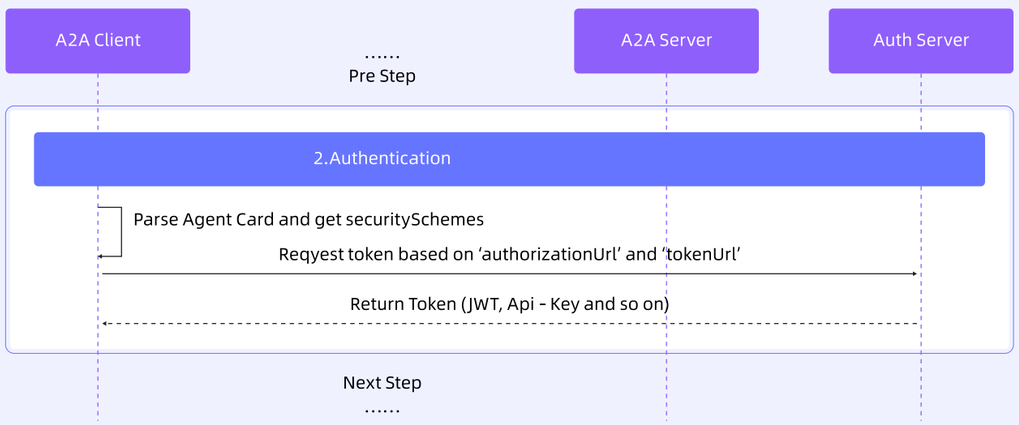
通过 A2A 注册中心 ，AgentCard 获取地址 ，固定配置 等方式获取目标 Agent Server的 AgentCard 元数据。关于具体的 AgentCard 的发现方式介绍，将在下个小节中展开，此处仅展示此步骤的基本流程：

2、A2A 授权和权限认证

对于一些需要访问鉴权的 Agent Server 来说，A2A 协议遵循标准Web安全实践：

- **认证声明：**智能体的认证要求需在 Agent Card 中明确声明。
- **凭据传递：**OAuth 令牌、API 密钥等凭据通常通过 HTTP 请求头传递，与 A2A 协议消息内容分离，避免耦合。

（注：具体鉴权方式由智能体自行定义，客户端需按 Agent Card 声明的要求适配）

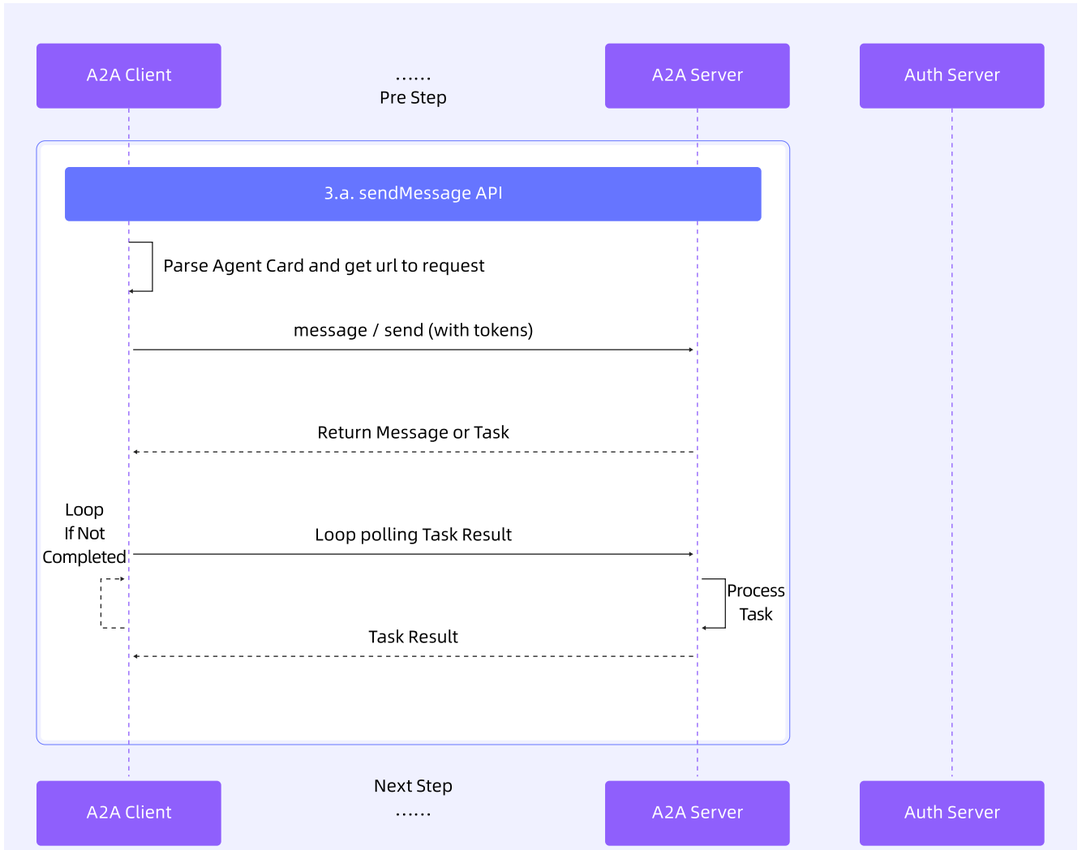


3、发起 A2A 请求

在通过步骤一、二获取 AgentCard 之后和认证 Token 后，A2A Client 其实就能够获取到 A2A Server 的所有可访问地址了，接下来就是根据需要，发送调用请求到 A2A Server。根据调用时的请求的交互机制不同，又大致可分为 同步请求（轮询） 和 流式请求 。

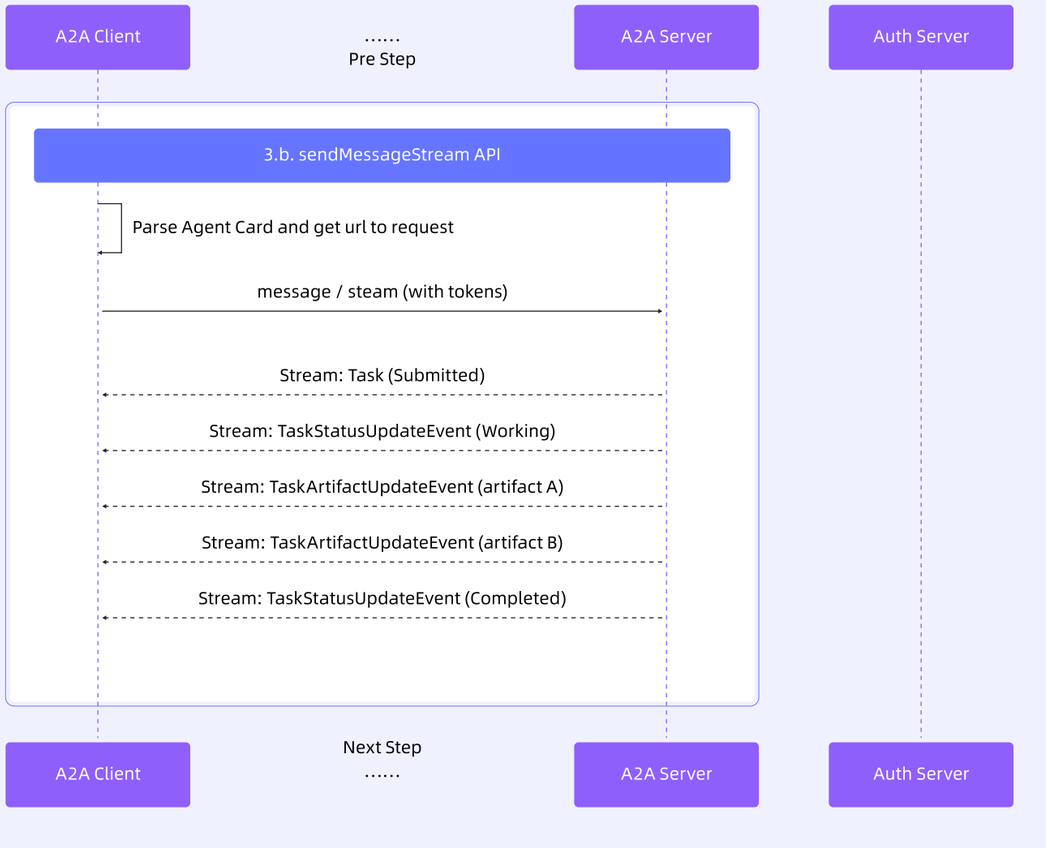
- **同步请求**

对于一些小型任务请求，或完全的后台运行任务，可以利用同步请求发起对远端 Agent 的调用，通过在远端 Agent 返回的 Message 直接获取结果，或定时轮询获取 Task 的运行结果，此步骤的大致流程如下：



- **流式请求**

而对一些需要较好体验的任务请求，可以采用流式请求的方式，实时获取任务执行的进度，并根据实时的Task状态和内容，进行 A2A Client 的实时结果展示，此步骤的大致流程如下：



3.4.3 基于 Nacos A2A Registry 的自动注册与发现

上一段中介绍了分布式智能体的发展趋势，并简要介绍了作为分布式智能体的基础通信协议 A2A 的核心概念、交互机制及工作流程。其中，工作流程的首个步骤就是进行远端 Agent 的 AgentCard 获取和发现。本段将对远端 Agent 的 AgentCard 获取和发现进行展开说明，先简要了解 AgentCard 的定义和作用，随后介绍目前 AgentCard 的几种主流获取和发现方式。

1、AgentCard 的作用

Agent Card 是 A2A Server（远端 Agent）的数字名片，为发现与交互提供必要信息，包含以下关键内容：

- **身份信息：**名称、描述、服务提供方。
- **服务端点：**远端 Agent 的访问URL，包含完整的域名/IP，端口，URI 等。
- **协议能力：**支持的协议功能（如流式传输 streaming 或推送通知 pushNotifications ）。
- **认证方式：**交互所需的鉴权机制（如 Bearer 令牌、 OAuth2 ）。
- **技能列表：**智能体可执行的任务或功能（ AgentSkill 对象），含技能 ID、名称、描述、输入/输出模式及示例。

A2A Client 通过解析 Agent Card 判断远端 Agent 是否适配当前任务、如何构建请求以及安全通信方式。因此 A2A Client 发现远端 Agent的 Agent Card 的过程，就是 A2A 协议中的 Agent 发现过程。

2、AgentCard 的获取方式

根据目前社区对发现方式的探索，以及官方对 Agent Card 发现方式的定义，Agent Card 的获取方式主要有 直接配置 、固定URI 和 注册中心3种获取和发现方式。

- **直接配置**

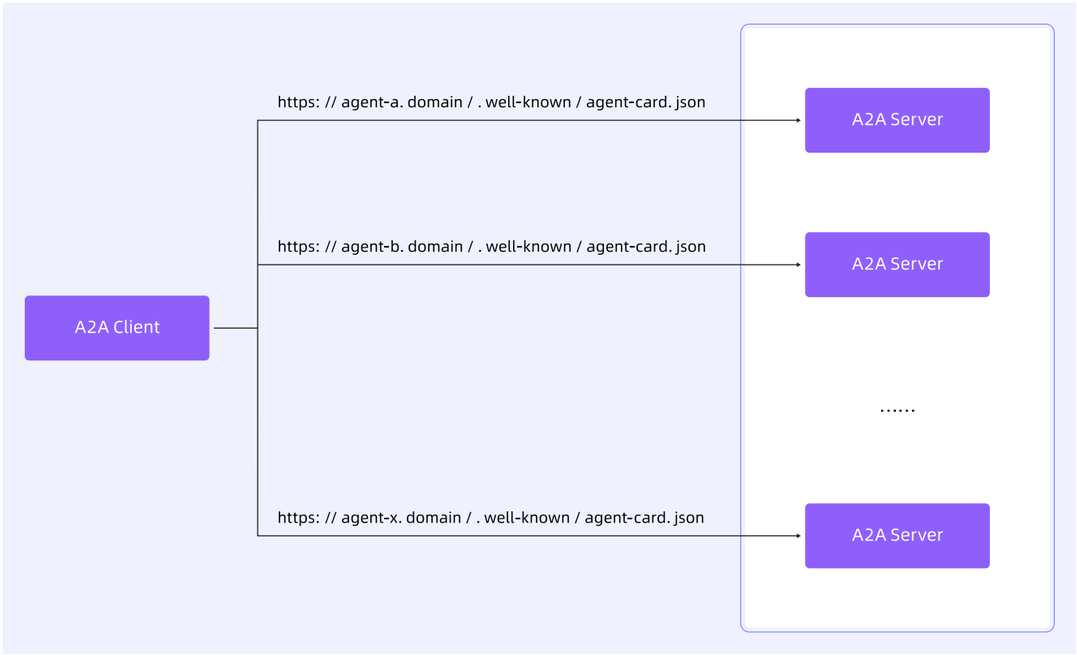
最简单的获取 AgentCard 的方式就是直接在 A2A Client 中直接配置，可通过硬编码、配置文件等方式，在程序中自主构建 AgentCard 的 Json 内容，并于 A2A Client 初始化时传递进去：

```java
// 伪代码，硬编码构建AgentCard，并初始化A2A Client

AgentCard agentCard = new AgentCardBuilder()
                          .name("testAgent")
                          .description("Only test")
                          .url("http://127.0.0.1/")
                          .capabilities(List.of(...))
                          .skills(List.of(...))
                          .build();
A2aClient client = new A2aClientBuilder().agentCard(agentCard).build();
client.sendMessage(...);
```

- **适用场景：**紧密耦合系统、私有智能体或开发测试环境。
- **机制：**客户端通过硬编码、配置文件、环境变量或私有 API 获取 Agent Card 信息。
- **流程：**根据具体应用的部署策略实现，无统一标准。
- **优点：**简单高效，适用于已知或静态的智能体关系。
- **注意事项：**灵活性低，动态发现能力弱；Agent Card 变更需客户端重新配置。

- **固定 URI（Well-Known URI）**

通过一个约定好的固定 URI，轮询或定期的从固定的远端服务获取远端 Agent 的 AgentCard，是一种兼顾了简单易用和灵活可变的发现方案。

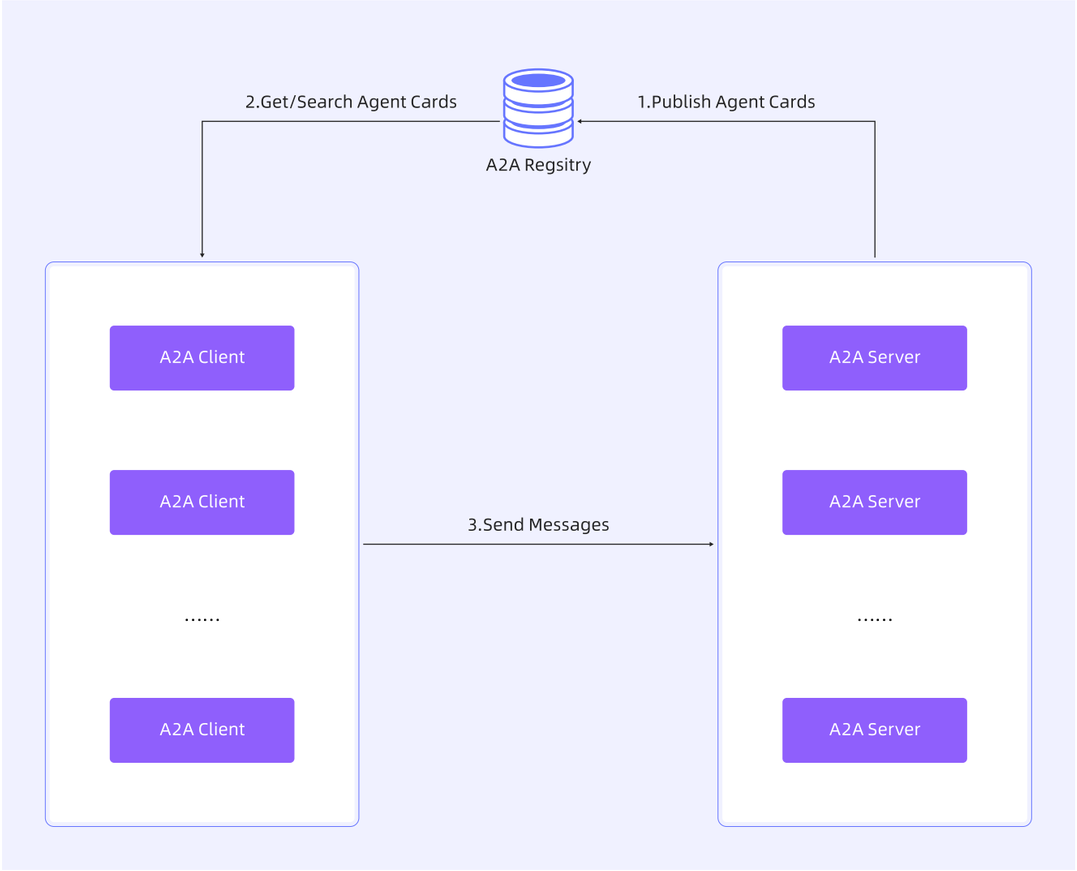


- **适用场景：**面向公开或特定领域内广泛发现的智能体。
- **机制：**
  - A2A 服务端将 Agent Card 托管在域名的标准化路径下。
  - 固定路径：https://{智能体服务域名}/.well-known/agent-card.json（遵循RFC 8615 标准）。
- **流程：**
  - 客户端获知目标服务域名（如 smart-thermostat.example.com ）。
  - 发起HTTP GET请求至 https://smart-thermostat.example.com/.well-known/agent-card.json 。
  - 服务端返回 JSON 格式的 Agent Card（若存在且可访问）。
- **优点：**简单、标准化，支持自动化发现（如爬虫或域名解析系统），将发现问题简化为“获取智能体域名”。

- 注意事项：
  - 适合开放或组织内可控域名的场景。
  - 若 A2A Client 依赖多个远端 Agent，需要维护和记录每个远端 Agent 的固定URI，同时也需要遍历所有的固定 URI。
  - 若 Agent Card 含敏感信息，访问端点需额外鉴权。

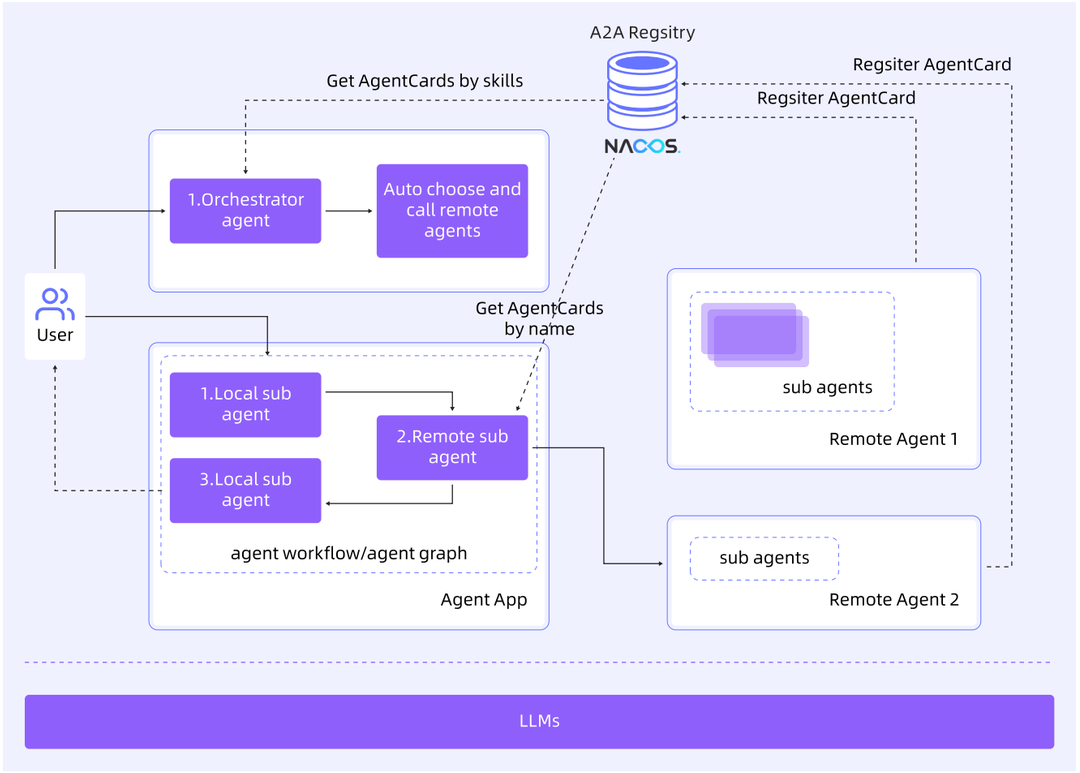
3.4.4 Nacos A2A 注册中心

如同微服务架构中进行微服务的服务发现体系，通过一个中心化的注册中心对远端 Agent 的 AgentCard 进行统一的集中管理。同时 A2A Client 也无需再保存和维护所需的远端 Agent 的公开域名或 Endpoint，仅需知晓注册中心的公开域名或 Endpoint 即可。是目前所有方案中，灵活度和可控性最高的一种发现方案。



- **适用场景：**企业环境、市场或生态系统中需集中管理的智能体。
- **机制：**
  - 由注册中心（中介服务）维护 Agent Card 集合，客户端通过查询条件（如技能、标签、提供方）检索目标。
- **流程：**
  - A2A 服务端或管理员向注册中心提交 Agent Card。
  - 客户端调用注册中心 API（如“查找支持流式传输且具备图像生成技能的智能体”）。
  - 注册中心返回匹配的 Agent Card 列表或引用。
- **优点：**
  - 集中化管理与治理，支持基于功能的动态发现。
  - 可实现访问控制、策略与信任机制。
  - 支持企业私有市场、公共市场等场景。
  - 注意事项：需额外注册中心服务。

随着 Nacos 3.1.0的发布，Nacos 可以作为 A2A 协议的注册中心，统一的进行 Agent 的管理，注册，发现和按需检索的能力；除此之外，Nacos 还支持对 Agent 的版本进行管理和快速回滚，以便正式的智能体应用在进行 A2A 远程访问时能够进行精细化的灰度和管理能力。



Nacos 除了作为 A2A 的注册中心，还可以作为 MCP 服务的注册中心，以及 Prompt 的动态管理中心，帮助用户从 Agent 应用视角管理所有 Agent 应用开发过程中的动态配置、工具、和远端 Agent 依赖内容，从而使用户像开发微服务应用一样简单地开发分布式 Agent 应用架构和服务体系。

ALIBABA CLOUD

3.5 消息驱动的智能体开发模式

在传统互联网应用中，消息队列广泛用于服务解耦、异步通信和削峰填谷等场景。它通过异步化的事件驱动方式，提升系统可扩展性与稳定性。典型场景如订单处理、日志收集、通知推送等。传统消息队列（如 Apache RocketMQ、Kafka）强调高并发写入、顺序消费和基本负载均衡，已形成成熟的技术范式。

然而，随着生成式 AI 的兴起，AI 应用呈现出截然不同的业务特征：推理耗时长达分钟级、上下文数据高达上百 MB、多轮对话需长期维护状态、多 Agent 协同依赖复杂异步编排，且严重依赖昂贵的 GPU 资源。

在此背景下，传统消息队列暴露出明显局限：无法高效支持百万级长会话隔离、缺乏对大消息的优化传输、难以实现消费速度的精细控制，更不具备优先级调度与资源导向的智能负载均衡能力。简单的异步模型已无法一步满足 AI 场景下对稳定性、成本控制与任务优先级的严苛要求。

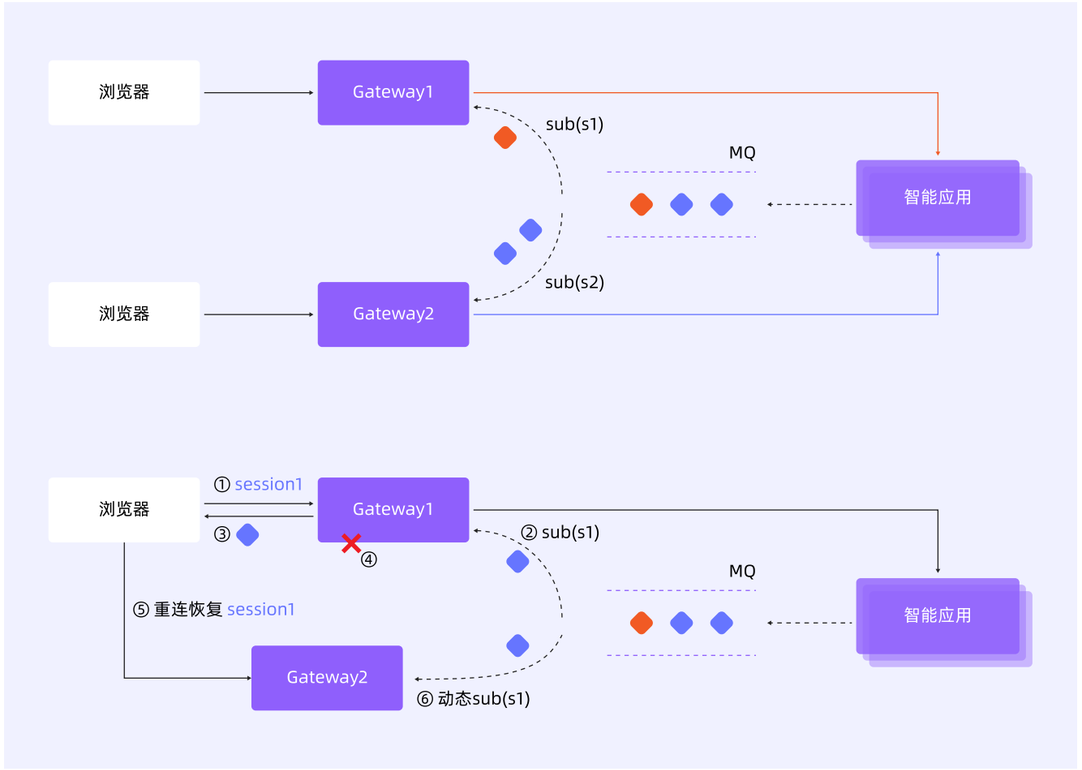
因此在AI 云原生架构下的消息队列必须具备以下特点：

- 支持长会话与大消息体的消息中枢。
- 实现削峰填谷、定速消费的智能调度能力。
- 提供优先级、权重控制的分级事件驱动机制。
- 构建高可靠、可恢复的 Agent 编排引擎。

3.5.1 消息模型提升 AI 通信效率

AI 应用的交互通常具有长耗时、多轮次和高算力成本的特点。当依赖 SSE 或 WebSocket 等长连接时，一旦连接中断（如网关重启、超时或网络波动），不仅会话上下文可能丢失，已执行的 AI 任务也会被迫中断，导致昂贵的计算资源被浪费。因此，构建一个可靠的会话管理机制，确保在长时间对话中上下文的连续性与完整性，减少因重连或重试带来的资源消耗，同时降低应用逻辑的复杂性，成为该场景下的关键技术挑战。

针对这一挑战，RocketMQ 提出了一种创新的轻量化架构——其核心理念是：为每个会话或问题动态创建一个独立的轻量级主题（Lite-Topic）。以客户端与 AI 服务建立会话为例，系统自动创建一个以 SessionID 命名的专属队列（如 chatbot/{sessionID} 或 chatbot/{questionID}），所有会话历史、上下文和中间结果均以消息形式在该主题中有序流转。通过将每个会话隔离在独立的消息通道中，不仅实现了上下文的持久化与顺序保障，也彻底解耦了会话生命周期与长连接状态，为构建高可靠、可恢复的 AI 对话系统提供了底层支撑。



这一创新架构的实现，依托于 RocketMQ 为 AI 场景深度优化的四大核心能力：

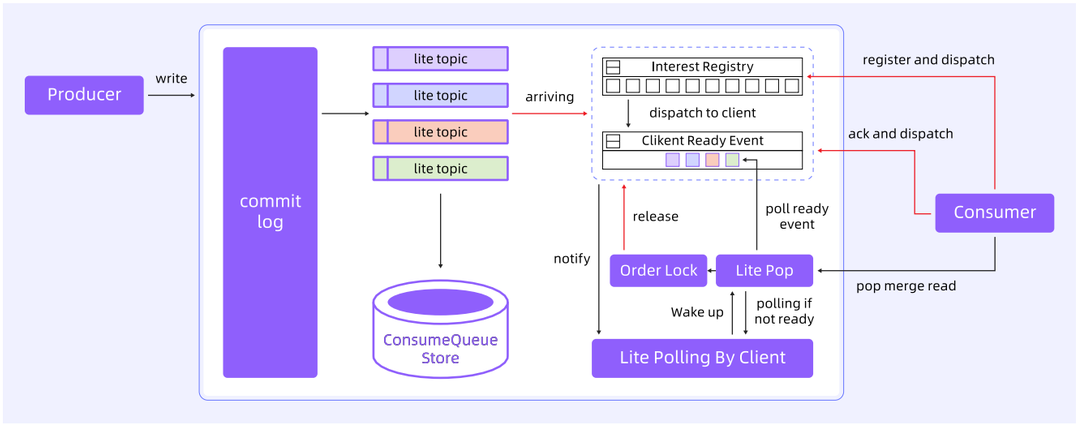
- **百万级 Lite-Topic 支持：**单集群可管理百万级轻量主题，为每个会话独立分配 Topic，实现高并发下的会话隔离，性能无损。
- **全自动轻量管理：**Lite-Topic 按需动态创建，连接断开后自动回收，彻底杜绝资源泄漏，运维零干预。
- **大消息体传输能力：**支持数十 MB 乃至更大消息，轻松承载长 Prompt、图像、文档等 AIGC 典型数据负载。
- **严格顺序消息保障：**在单队列内保证消息有序，确保 LLM 流式输出的 token 顺序不乱，支撑连贯流畅的交互体验。

从业务模型上来看，轻量级消息模型包括了轻量级发送、轻量级订阅以及全新的消费分发策略。

- **轻量级发送：**
  - 基于百万队列的方案，本质上是一个个 Queue。
  - 从全局上来看，一个轻量级 Topic 不会存在于每一个 Broker 上，在分配和发送时像顺序 Topic 的发送一样要做 Queue 的 Hash。
  - Queue 的消息是某个 Broker 专属的，一个轻量级 Topic 的发送在只会到一台Broker，而不是轮询发送。

- **轻量级订阅：**
  - 消费组 Group 的概念被弱化。
  - 订阅关系、消费进度管理粒度更细，以 client_ID 维度维护。
  - 新增互斥（ Exclusive）消费模式。
  - TTL 到期后自动删除订阅关系。

- **消费分发策略：**
  - 客户端发起读请求不再指定 Topic，而是 Broker 根据 client_ID 识别订阅关系，并返回多个 Topic 的多条消息。
  - 引入类似 Epoll 机制的 Topic ready set，在 POP 请求处理时直接访问就绪的topic。
  - 当订阅上线、新消息发送、消息 ACK（Acknowledgement，确认）后仍有消息、order Lock 释放时往 topic ready set 进行 add 操作。



轻量化消息模型突破了传统消息队列订阅关系单一、隔离粒度粗、管理复杂等局限，通过精细化的资源隔离机制，实现了海量 Lite-Topic 的高效生命周期管理与低延迟消息投递。该模型为 AI 场景下的会话管理、上下文持久化以及多 Agent 间的异步协同，提供了高可靠、易扩展的全新架构解决方案。

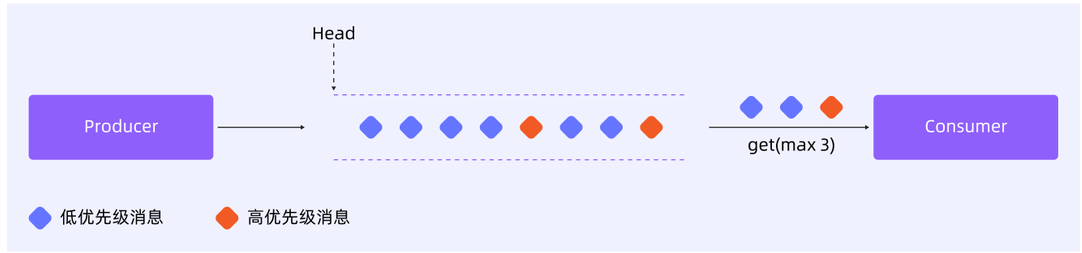
3.5.2 基于消息驱动的智能化资源调度

大模型服务普遍面临两大核心资源调度难题：

- **负载不匹配：**前端请求常突发波动，而后端算力资源有限且稳定，直接对接易引发服务过载或利用率不足，难以实现稳定服务与资源效率的平衡。
- **资源分配无差别：**在流量被平滑后，仍需解决关键问题——如何优先保障高价值任务（如 VIP 请求、核心业务）的资源获取，以最大化算力的服务价值。

RocketMQ 不仅实现了流量的平滑缓冲，更通过优先级与配额机制，赋予系统智能调度与资源优化的能力，推动消息系统从被动队列向主动控制中枢演进。开发者无需自研复杂调度中间件，即可实现对 AI 流量的精细化管控。其核心能力包括：

- **天然削峰填谷，保护 AI 算力：**消息队列作为“流量水库”，可缓存突发请求，使后端 AI 服务按自身处理能力自适应消费，实现负载均衡，避免因瞬时高峰导致服务崩溃或资源闲置。
- **定速消费，精准控制算力使用：**支持为消费者组（ConsumerGroup）设置消费配额（quota），实现稳定速率消费。开发者可精确设定每秒调用次数，在保障模型服务稳定的前提下，最大化 GPU 利用率与系统吞吐。
- **优先级调度，实现智能资源分配：**在资源竞争场景下，支持多维度调度策略：
  - 抢占式优先级：将 VIP 请求、关键任务标记为高优先级消息，确保其优先被消费，保障核
  - 心业务响应质量。
  - 权重动态分配：在多租户共享算力池场景中，可根据业务重要性或执行状态动态调整消息
  - 优先级，平衡吞吐效率与资源公平性，防止个别租户“资源饥饿”。



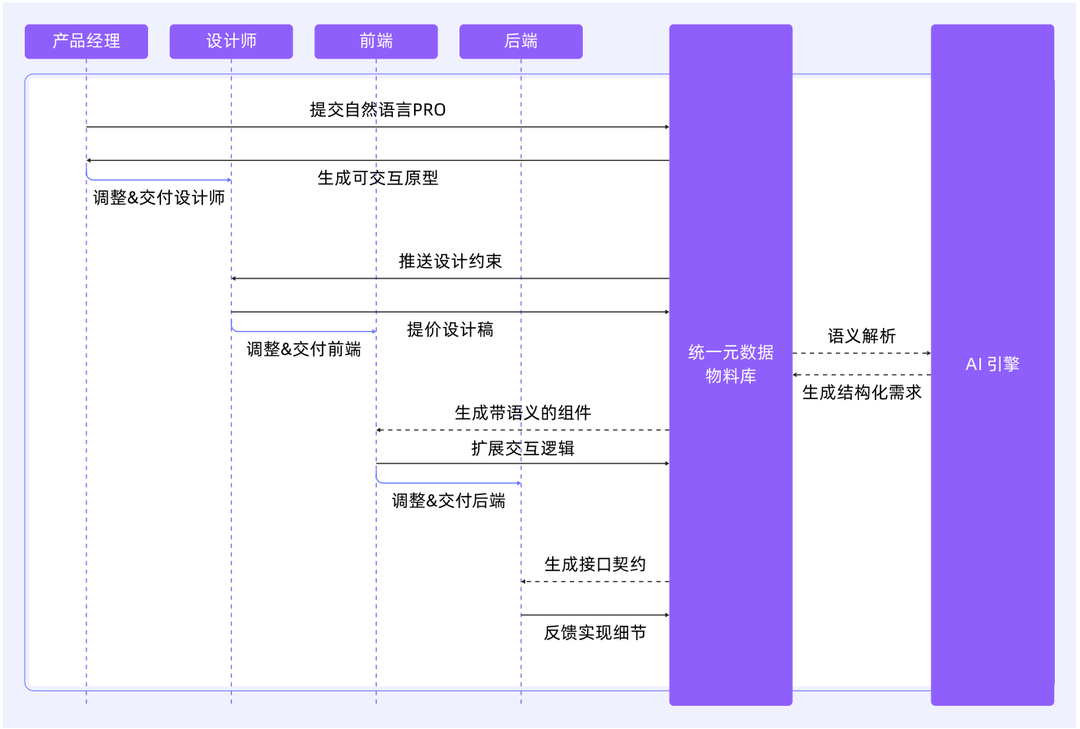
通过 RocketMQ 的综合调度能力，可以高效稳定的实现资源管理。用户将请求统一写入RocketMQ，突发流量被暂存为“待处理会话”。AI 推理服务按自身处理能力设置定速平滑消费，避免雪崩或空转，保障服务稳定性。当有更高优先级的用户消息进入时会被标记，系统优先调度处理，确保高价值客户获得毫秒级响应体验。而当多个业务线共享算力池时，根据 SLA 和执行状态动态调整消息优先级，保障核心业务的同时，避免低优先级租户长期得不到资源。

3.6 基于统一元数据的 AI 协同开发模式

前面章节我们介绍了使用 AI 框架进行智能体应用开发的基本过程。在实际场景中，应用的开发通常需要多角色协同，接下来换一个视角，看看 AI 能力如何在应用开发中带来了更高效的协同。

传统开发流程的痛点：产品原型→设计稿→前后端代码，存在很多重复劳动。产品经理用 Axure 等原型工具画的原型，设计师要用 Figma、MasterGo、Sketch 等工具画成高保真设计稿，交付给前端又要在本地 IDE 里改写成 HTML、CSS……核心问题是上游角色的交付物下游无法复用。

AI 时代，可以基于统一元数据，所有交付物都使用 HTML 格式，从建立需求即代码的团队协作基准开始；定义全局标准，协调每个角色的产出可以直接被下游角色使用，并且在开发全流程的每个环节基于 AI 提效。例如：产品经理描述需求，AI 生成交互式 HTML 原型；设计师在同样的 HTML 基础上，通过拖拽组件调整样式，AI 生成对应的 CSS 和组件结构；前端工程师在生成的 HTML/CSS 基础上，添加 JavaScript逻 辑，AI 可能辅助生成重复的逻辑代码；后端工程师根据前端定义的接口规范，由 AI 生成初步的 API 代码和数据库模型。



统一元数据需要长期建设，包含设计系统、组件库、API 规范、状态管理等，同时需要确保它们在不同阶段之间可以无缝转换。例如，产品经理的需求文档如何转化为可交互的原型，设计师如何在这个原型基础上进行视觉设计，前端如何直接使用设计稿生成的代码，后端如何根据接口规范自动生成代码。

该协同模型需要工具链的支持，例如开发一个集成 AI Agent 的平台，提供从 PRD 到设计到代码的转换工具，每个角色在这个平台上工作，AI 自动处理转换和生成。此外，测试和迭代也是一个重要环节。需要确保每个阶段的交付物经过验证，避免错误累积到下游。可能引入自动化测试，AI 辅助测试用例生成，或者实时预览功能，让各角色及时看到修改效果。还要考虑协作中的沟通和版本管理。统一物料库可能需要类似 Git 的版本控制系统，记录每次修改，方便回溯和协作。AI 可以在这里帮助解决冲突，或者自动合并不同角色的修改。

统一物料体系架构	
智能协作层	AI需求解析 → 智能校验 → 冲突检测
物料规范层	Design Token库 → 语义化组件 → API契约
转换引擎层	文生图 → 图生代码 → 代码重构 → 接口联调
基础数据层	结构化PRD → 设计系统 → 接口规范

在 AI 时代，产品经理构建业务上下文、设计师构建设计系统上下文、前后端构建专业/领域上下文，每个角色不再是孤立的，从建立需求即代码的团队协作基准开始，从组件契约标准化入手，逐步构建跨工具链的智能开发环境。

AI 上下文工程

Context Engineering

CHAPTER

4.1 提示词工程

大型语言模型（LLM）作为 AI 原生应用的技术核心，其输出质量高度依赖于输入的精确性与引导性。提示词工程（Prompt Engineering），正是围绕如何设计、构建和优化输入文本，以引导大型语言模型产生期望输出的系统性方法。

它并非简单的提问，而是一套涵盖指令设计、上下文注入、角色设定和格式控制的综合性技术。一个精心设计的提示词，能够系统性地引导模型的生成过程，最大化地激发其潜力，确保输出的准确性、相关性和一致性。

在 AI 原生应用的初期阶段，提示词工程构成了将用户意图转化为模型可执行指令的核心接口。它使得开发者和用户能够以低成本、高效率的方式驾驭强大的 LLM，完成从内容创作、代码生成到数据分析等一系列任务。

4.1.1 优秀提示词的核心实践

高质量的提示词能够显著提升模型的表现。尽管具体实现千变万化，但优秀的提示词工程实践普遍遵循以下核心原则：

产生幻觉和不确定性的根本原因错综复杂：

- **明确角色与目标：**

在提示词开头为 AI 赋予一个专家角色，能有效激活模型内部与该领域相关的知识库和语言风格，这比泛泛而谈的请求更具引导性。

优秀范例：“假设你是一位拥有10年经验的市场营销总监，请为一款新型智能手表起草一份面向年轻专业人士的产品介绍文案。”

- **提供清晰指令与完整上下文：**

明确告知 AI 需要执行的任务，并提供完成任务所必需的背景信息。指令越具体，歧义越少，模型偏离主题的可能性就越低。

优秀范例：“请总结以下技术文章的核心观点（不超过200字），并列出其中提到的三个关键数据。不要添加个人评论。文章内容如下：[在此处粘贴文章]”

- **运用范例进行引导：**

对于需要模型遵循特定模式或风格的复杂任务，提供一到两个“输入-输出”范例，能让AI迅速理解并模仿要求，这远比冗长的描述更有效。

优秀范例：“请分析以下句子的情感及关键对象。范例：输入：‘我喜欢这部电影的剧情。’ 输出：{‘sentiment’: ‘positive’, ‘aspect’: ‘plot’}。现在，请分析这个句子：‘这台相机的画质很棒，但电池续航太短。’”

- **定义结构化输出格式：**

在应用开发中，要求模型返回特定格式（如 JSON、Markdown 列表）的数据至关重要，这能确保AI的输出可以直接被下游程序解析和使用。

优秀范例：“请将你的回答组织成 JSON 格式，包含 ‘product_name’ 和 ‘key_features’ 两个字段，其中 key_features 是一个包含至少三项功能的数组。”

4.1.2 提示词工程的固有局限性

尽管提示词工程在特定场景下表现优异，但随着 AI 应用向更复杂的 Agent 系统和长流程任务演进，其固有的局限性日益凸显，标志着仅优化提问这一思路已接近其能力上限。

- **应对动态及长流程任务的局限**

提示词本质上是静态的、预定义的模板。这种模式难以应对需要动态适应环境变化、处理高并发请求或执行长时间、多步骤的复杂任务。当任务流程包含多个分支或需要根据前序步骤的结果动态调整后续行为时，单个或一组固定的提示词便显得力不从心，缺乏必要的灵活性和扩展性。

- **应对模型固有的无状态特性**

标准的提示词交互是无状态的，即模型不会自动记忆之前的对话历史。虽然可以通过在每次请求中手动拼接历史记录来模拟短期记忆，但这在多轮交互中会迅速变得低效且成本高昂，并受到上下文窗口长度的严格限制。它无法有效建立跨越多次对话的长期记忆，导致对话的连贯性和个性化体验受限。

- **无法突破模型固有的知识边界**

提示词工程的本质是优化对模型已有知识的访问与提取路径，而非扩充模型可访问的知识库本身。对于模型训练数据截止日期之后的新知识、企业内部的私域知识或需要实时更新的信息，无论提示词设计得多么精妙，模型都无法凭空生成。单纯依赖提示词工程，无法解决 LLM 固有的知识时效性滞后和领域知识缺失的问题。

- **长上下文中的信息利用不均衡**

研究与实践表明，LLM 在处理长上下文时存在“中间遗忘”（Lost in the Middle）现象。模型对位于上下文窗口（即提示词）开头和结尾的信息关注度最高，而对中间部分的信息则容易忽略。这意味着，即使开发者在提示词中精确地提供了所有必需信息，但如果关键指令或数据恰好位于中间位置，模型仍有可能“视而不见”，导致任务执行失败或结果偏离预期。这一现象在通过检索增强生成（RAG）等方式注入大量外部知识时尤为突出，严重挑战了长上下文处理的可靠性，也为构建需要依赖大量背景信息的复杂应用带来了根本性障碍。

这些限制清晰地表明，要构建更强大、更可靠的下一代 AI 应用，必须超越对单次交互的优化。我们需要一种全新的范式，将技术重心从优化静态的提问方式，扩展为动态构建与管理系统的认知环境。这并非简单的替代，而是将提示词设计作为整个上下文构建中的一个关键环节，进行更系统化的提升，确保模型在“思考”的每一刻都能获取最准确、最相关的信息。

为此，上下文工程（Context Engineering） 应运而生。

4.2 上下文工程

上一节我们剖析了提示词工程的固有局限性，揭示了其在应对动态任务、管理长期记忆、突破知识边界以及高效利用长上下文等方面的不足。这些挑战共同指向一个结论：要构建真正强大且可靠的 AI 原生应用，仅优化如何提问是远远不够的。我们必须转向一个更宏大、更系统的范式，即上下文工程（Context Engineering）。

上下文工程的核心理念，在于将 AI 应用开发的焦点从优化单次交互的指令，转向为模型的每一次推理动态构建一个完整、准确且高效的认知环境。它不再局限于设计静态的提示词模板，而是致力于创建一个系统，确保模型在执行任务时，能够实时获取并利用完成该任务所需的所有相关信息，包括外部知识、历史记忆、可用工具及执行环境。

这一范式转变的根本原因在于，AI 应用的复杂性已远超单次问答。现代 AI 系统，尤其是Agent，被要求执行长流程任务、维持个性化交互并确保事实准确性。随着 LLM 上下文窗口从几千扩展到超过百万 Tokens，如何有效填充和管理这个巨大的信息空间，使其成为 AI 的工作记忆而非信息噪声，成为了决定应用成败的关键。上下文工程正是为应对这一核心挑战而生的系统级解决方案。

4.2.1 上下文工程与提示词工程的核心区别

为了更清晰地理解这一范式升级，我们可以从目标、范围和关键技术3个维度对比上下文工程与提示词工程：

维度	提示词工程	上下文工程
核心目标	优化单次交互的指令，以获得最佳输出	构建动态的上下文系统，确保推理的准确性与可靠性
工作范围	聚焦于单轮或有限的多轮交互	整合多源、异构的数据流与工具
关键技术	指令设计、范例选择、角色扮演	检索增强生成（RAG）、向量数据库、工作流编排、记忆管理

简而言之，如果说提示词工程是“术”，专注于提升单次沟通的效率；那么上下文工程则是“道”，旨在构建一个能让 AI 持续、高效、可靠地解决问题的系统性框架。

4.2.2 上下文工程的关键组成部分

上下文工程通过协同工作的一系列核心组件，为 LLM 构建其动态认知环境。这些组件共同构成了下一代AI应用的基石。

- **外部知识库的动态供给**

为解决 LLM 知识陈旧和领域知识缺乏的问题，上下文工程的核心是为其接入外部知识库。通过检索增强生成（Retrieval-Augmented Generation, RAG）技术，系统能够在接收到用户请求时，首先从企业的私有数据库、实时信息流或互联网等外部来源检索相关信息，再将这些检索到的信息与提示词一同组合成最终的上下文，引导 LLM 基于准确、实时的知识进行回答。

- **长期与短期记忆系统**

为了实现连贯且个性化的交互，上下文工程引入了记忆系统。短期记忆负责管理当前对话的上下文，确保多轮对话的流畅性。长期记忆则负责存储跨对话周期的关键信息，如用户偏好、历史决策、重要事实等，使 AI 能够记住用户，提供真正个性化的服务。

- **工具与能力的扩展**

上下文不仅包含静态信息，也包括对模型可用的工具（动态能力）的描述。上下文工程通过为 LLM 提供一系列工具（Tools），本质上是 API 或函数调用来扩展其能力边界。这使得 LLM 不再局限于文本生成，而是可以查询数据库、调用外部服务、执行代码甚至操作物理设备，成为一个能够与数字和物理世界交互的智能体。

- **运行时的上下文管理**

面对有限且昂贵的上下文窗口，特别是在长对话或复杂任务中，如何高效管理上下文至关重要。这包括一系列运行时策略，如上下文压缩与摘要，用于在保留关键信息的同时减少 Token 消耗；以及上下文重排（Re-ranking），用于解决中间遗忘问题，将最重要的信息放置在模型最关注的位置，从而提升长上下文处理的可靠性。

通过对这些关键组件的系统性设计与优化，上下文工程为构建高效、可靠且具备深度认知能力的 AI 原生应用提供了坚实的基础。接下来的章节，我们将深入探讨实现上下文工程的核心技术与最佳实践。

4.3 RAG 技术原理与挑战

在上一节中，我们明确了上下文工程的目标，为 AI 构建一个动态的认知环境。上下文工程是一个系统性工程，涉及多种技术路径，其中，检索增强 RAG 生成是上下文工程中非常重要的组成部分。RAG 架构通过将 LLM 与外部可信知识库动态连接，从根本上弥合了通用模型能力与企业特定、实时性需求之间的鸿沟，使其成为当前构建动态上下文最重要和最被广泛采用的技术路径。

LLM 固有的三大局限性，知识时效性滞后、缺乏企业私域知识以及存在幻觉风险，构成了其在严肃商业场景中落地的关键障碍。RAG通过引入开卷考试的模式，在模型生成答案之前，先从指定的知识库中检索相关信息，并将其作为上下文提供给 LLM，从而引导模型基于可靠的外部知识进行回答。这种模式不仅有效缓解了上述局限性，更成为驱动 AI 原生应用业务价值实现的核心引擎。

4.3.1 RAG 的基础范式：三阶段工作流

RAG 的基础范式可被清晰地解构为三个核心阶段：索引（Indexing）、检索（Retrieval）与生成（Generation）。这三个阶段环环相扣，共同构成了一个完整的知识注入–匹配–应用的流程。

1、第一阶段：索引

索引阶段的目标是将原始的、非结构化的外部知识（如企业私域文档、实时数据流等）转化为机器可高效检索的格式。这一过程是 RAG 系统性能的基石，其质量直接决定了后续检索的准确率和效率。通用实现包含以下关键步骤：

- **文档解析与提取：**首先，系统需要从多种格式的原始文档（如 PDF、Word、HTML 等）中解析并提取出纯文本内容。高质量的解析能最大程度地保留原文的结构和语义信息。
- **文本分块（Chunking）：**由于 LLM 上下文窗口的长度限制，以及为了实现更精准的语义匹配，长文档必须被切分成更小、更易于管理的“块”（Chunks）。分块策略（如固定大小、按句子/段落切分）对检索效果有至关重要的影响。
- **语义向量化（Embedding）：**为让机器理解文本的语义，每个文本块都会通过一个预训练的Embedding模型（如 qwen3-embedding、bge-m3）被转化为一个高维度的数学向量。在生成的向量空间中，语义相近的文本块在空间位置上也更接近。

- **构建向量索引库：**将所有文本块的向量表示存储起来，并建立高效的索引，最终形成向量索引库（通常由向量数据库承载）。这使得系统在接收到查询时，无需暴力遍历所有向量，即可快速找到最相似的向量。

2、第二阶段：检索

检索阶段是连接用户意图与知识库的桥梁。其核心任务是根据用户的自然语言查询，从已构建好的向量索引库中，快速、准确地找出最相关的 N 个知识块（Top-K）。

该过程首先将用户的查询通过相同的 Embedding 模型转化为查询向量。随后，系统利用此查询向量在向量索引库中执行相似度搜索（通常使用余弦相似度等度量方法），计算查询向量与库中所有文本块向量的相似度得分。最后，根据得分高低排序，返回最相关的 Top-K 个文本块作为后续生成阶段的事实依据。

3、第三阶段：生成

生成阶段是 RAG 流程价值最终体现的环节。系统会将前一阶段检索到的多个相关知识块与用户的原始查询进行整合，构建一个增强的提示词（Augmented Prompt）。

这个提示词通常包含明确的指令，要求 LLM 基于提供的上下文信息来回答用户的问题。一个典型的提示词模板如下：

```JSON
请根据以下提供的上下文信息来回答用户的问题。如果上下文信息不足以回答，请明确告知。

上下文：
[检索到的知识块1]
[检索到的知识块2]
...

问题：
[用户的原始查询]
```

随后，这个包含了丰富、相关且具时效性知识的提示词被发送给 LLM。LLM 利用其强大的自然语言理解和生成能力，对信息进行综合、推理、提炼和总结，最终生成一个流畅、连贯且基于所提供事实的最终答案。

4.3.2 RAG 在工程化落地中面临的挑战

尽管 RAG 的基础范式逻辑清晰，但在复杂的现实应用中，将其从原型转化为稳定、高效的生产级系统，开发者会面临一系列严峻的工程挑战。这些挑战贯穿于数据处理、查询理解、召回匹配和复杂推理的全链路。

- **知识单元的完整性与信息密度的抉择**

RAG 系统的根基在于高质量的知识库，而知识库的构建始于对原始数据的精细处理。传统的固定长度或基于分隔符的切块方法，极易破坏信息的完整性，例如一个完整的逻辑段落、一张表格或一段代码块可能被强行截断，导致上下文信息丢失。同时，一个知识单元如果包含过多无关的“噪声”信息（如页眉、页脚、广告等），会稀释其核心语义，干扰检索模型的判断。

- **难以精准捕捉模糊、多样的用户意图**

用户与 RAG 系统的交互始于一次查询，而用户的查询往往是模糊、口语化甚至包含多重意图的。自然语言天然存在的模糊性与歧义性，以及用户提问的语言风格与知识库文档的书面化、专业化风格之间存在的“语义鸿沟”，都对系统的查询理解能力构成了挑战。若无法准确理解用户的真实意图，后续的检索和生成环节都将是无源之水。

- **召回匹配时难以兼顾语义相关性与关键词精确性**

召回是 RAG 系统的核心环节。主流的基于向量的语义检索，虽能很好地处理同义词、近义词问题，但在需要精确匹配的场景（如专有名词、技术术语、产品型号或代码函数名）时，可能会错失最相关的结果。而传统的关键词检索虽擅长精确匹配，却无法理解语义，对查询的措辞变化非常敏感。如何在二者之间找到最佳平衡点，是召回环节的一大挑战。

- **需要探索如何在检索精度与完整性之间取得平衡**

将召回的知识单元整合到 LLM 的提示词中，面临着大海捞针和上下文窗口限制的双重问题。如果将大量召回的知识单元不加选择地全部塞入上下文，关键信息可能被淹没在冗余内容中，触发 LLM 的中间遗忘问题。而如果上下文过少，则可能导致 LLM 因信息不足而无法生成高质量的答案。

- **应对需要多知识点综合推理的查询实现**

现实世界中的许多问题并非通过单一的知识片段就能回答，它们往往需要综合、比较、推理分布在多个文档、甚至多个数据源中的信息。这类“多跳（Multi-hop）”问题是检验 RAG 系统智能水平的试金石。传统的一次检索、一次生成模式，难以应对需要逐步推理的复杂问题，容易出现推理链条的断裂。

认识到这些局限性是推动技术演进的第一步。为了构建真正能够应对复杂业务场景、实现高精度、高鲁棒性知识服务的 AI 原生应用，我们必须在基础范式的之上，探索更为精细化、系统化的工程落地策略。接下来的章节，我们将深入剖析 RAG 系统的优化实践。

ALIBABA CLOUD

4.4 RAG 系统优化实践：索引构建

上一节我们探讨了 RAG 在工程落地中面临的一系列挑战，其中“如何保证知识单元的完整性与信息密度”是所有后续环节的基础。索引构建的质量，直接决定了 RAG 系统能力的上限。一个粗糙的索引过程，即便后续的检索与生成环节再精妙，也无法弥补其先天缺陷。

因此，本节将聚焦于 RAG 系统的第一个核心阶段：索引构建。我们将从基础的分块策略出发，逐步深入到高级优化技术，并最终探讨如何处理包含表格、图像等复杂结构的文档，从而为构建一个高效、精准的知识库奠定坚实基础。

4.4.1 基础文本分块策略

文本分块（Chunking）是将原始文档转化为可检索单元的第一步，其核心目标是在保持语义完整性和控制块大小之间取得平衡。

1、固定大小分块

这是最简单直接的方法，即按照预定义的长度（如字符数、单词数或 Token 数）将文本分割为统一大小的片段。

- **优势：**实现简单，标准化程度高，计算开销低，适合快速原型验证或处理结构化较弱的文本。
- **劣势：**完全忽略文本的语义边界，极易将一个完整的句子或逻辑段落强行切分到不同的块中，导致严重的语义断裂。

2、基于句子或段落分块

此策略利用文本的自然结构（如句号、问号等标点符号或段落换行符）作为分割依据。

- **优势：**尊重语言的语法和逻辑结构，能更好地保持单个知识点的完整性，提升块的可读性与语义内聚性。
- **劣势：**块的大小会随句子或段落的长度而波动，可能产生过短（信息量不足）或过长（超出上下文限制）的块。

3、语义分块

这是一种更先进的动态划分策略。它通过引入 Embedding 技术计算相邻文本单元（如句子）间的语义相似度，在相似度得分显著下降的位置进行切分，从而识别出文本中的语义边界。这旨在创建包含连贯思想或主题的语义单元。

- **优势：**最大程度上保留了自然语义边界和主题的连贯性，生成的知识块信息密度高，尤其适用于法律条款、技术文档等对逻辑连贯性要求高的场景。
- **劣势：**计算成本较高，因需要进行嵌入和相似度计算；且分块质量依赖于所用 Embedding 模型的性能。

4.4.2 高级分块优化策略

为了克服基础策略的局限性，业界发展出了一系列高级优化技术，旨在进一步提升检索的精准度和生成上下文的质量。

1、滑动窗口（Sliding Window）

该策略通过在相邻的文本块之间设置一个重叠区域（Overlap），来减轻因硬性切分导致的边界信息丢失问题。例如，在切分时，下一个块会包含上一个块结尾的一部分内容。

- **解决的问题：**缓解语义断裂。当用户的查询恰好与两个块的边界区域相关时，重叠内容确保了相关上下文能够被完整地检索到。
- **实践考量：**重叠区域的大小需要权衡，过小则效果不彰，过大则会增加存储和计算的冗余。

2、小块检索，大块生成（Small-to-Big）

这是一个核心的优化思想，旨在化解小块检索准，大块语境足的矛盾。其实现方式通常是父子文档映射：

- **索引阶段：**将文档切分成两种粒度。一种是适合检索的、语义集中的子块（Small Chunks），另一种是包含更完整上下文的父块（Big Chunks）。只对子块进行向量化并存入索引库。
- **检索阶段：**使用用户的查询去匹配精准的子块。
- **生成阶段：**命中子块后，系统映射回其对应的父块，并将这个包含更丰富上下文的父块传递给 LLM，用于生成答案。
- **解决的问题：**兼顾了检索的精准性和生成所需的上下文完整性，显著提升了复杂问题的回答质量。

3、元数据标注与过滤

该策略主张在索引阶段为每个文本块附加丰富的元数据（Metadata），如文档来源、章节标题、作者、发布日期、关键词等。

解决的问题：将单一的语义检索升级为元数据过滤 + 语义检索的混合模式。在检索时，可以首

先通过元数据对候选范围进行精确筛选（例如，只检索“第三章节”或“2024年之后”的文档），然后在此范围内再进行语义搜索。这极大地提升了检索效率和准确性，尤其是在大规模知识库中。

4.4.3 处理复杂文档的索引策略

真实世界的文档往往不只是纯文本，还包含表格、图片、代码、脚注等复杂结构。一个健壮的索引流程必须能够感知并区别处理这些元素。

1、层次化索引

对于具有清晰层级结构（如章节-段落-句子）的文档，可以构建一个层次化的索引结构。

- 实现方式：为文档的顶层结构（如章节标题或摘要）创建摘要索引，为底层的详细内容创建区块索引。检索时，可以先在“摘要索引”中快速定位到相关的章节，再到该章节的“区块索引”中进行精细查找。
- **优势：**模拟了人类目录-章节-段落的阅读认知过程，实现了由粗到细的高效检索，特别适用于长篇技术手册、法律文书和学术论文。

2、多模态内容的处理与表示

该策略的核心是为不同模态的数据采用专门的处理与表示方法，将其转化为适合索引的格式，并在最终的知识单元中保留其核心语义。其实现方式是：

- **文本：**采用前述的语义分块策略。
- **表格：**并非将其简单序列化为文本，而是提取其结构化数据（如转化为 Markdown 或 JSON 格式），或对表格进行摘要描述，并单独索引。
- **图像：**使用 OCR 技术提取图中文字，并结合图像描述模型（Image Captioning）生成对图像内容的语义描述，共同作为该图像的索引内容。
- **优势：**最大限度地保留了文档的原始信息，减少了因格式转换带来的语义损失，是构建真正能够理解复杂混合文档的 RAG 系统的关键。

索引构建是 RAG 系统中一项精细且至关重要的工程任务，远非简单的文本切分所能概括。它是一个需要根据数据源的特性、结构和应用场景，进行策略选择与组合优化的过程。一个设计精良的索引，能够从源头上保证知识的完整性、准确性和可检索性，是实现高质量、高可靠性 RAG 系统的坚实地基。

4.5 RAG 系统优化实践：检索流程

在上一节中，我们详细探讨了如何构建一个高质量、结构优良的索引库，这是 RAG 系统的静态基础。然而，一个优秀的知识库必须匹配一个同样高效的检索流程，才能在用户提出请求时，将其潜力完全释放。检索流程是 RAG 系统的动态核心，它承载着从理解用户模糊的意图，到在海量知识中精准定位，再到最终构建出信息浓缩、结构清晰的“完美上下文”的全部使命。

本章将聚焦于 RAG 系统的在线优化实践，深入探讨查询理解与增强、多路召回与混合检索，以及结果精炼与上下文重组这三大关键环节的策略与技术。

4.5.1 查询理解与增强

检索流程的起点是用户的原始查询，而这一查询往往并非最优的检索形式。查询理解与增强的目标，就是通过一系列技术手段，将用户原始、模糊的查询，转化为机器更容易理解、更适合下游检索的结构化或增强型查询。

1、查询改写与扩充

用户的原始查询可能包含口语化表达、拼写错误，或使用的词汇与知识库中的术语不完全匹配。查询改写与扩充旨在通过 LLM 的语言理解和生成能力，将原始查询转化为一个或多个更清晰、更规范、更适合检索的查询版本。

- **实现方式：**采用改写-检索-阅读（Rewrite-Retrieve-Read）框架，在检索前增加一个“改写”步骤。例如，一个模糊的查询“AI 原生应用咋回事”可以被改写为更精确的“AI 原生应用的架构定义是什么？”或“AI 原生应用的核心特征有哪些？”。
- **效果：**纠正了口语化表达，明确了查询焦点，并通过生成多个角度的查询，扩大了语义覆盖面，从而提升召回相关文档的可能性。

2、复杂问题分解

当用户提出一个包含多个子问题或需要多步推理才能解答的复杂问题时，单一的、整体性的检索往往难以奏效。复杂问题分解技术旨在将一个宏大的问题，拆解成一系列更小、更具体的子问题。

- **实现方式：**利用 LLM 识别出原始问题中隐藏的、需要独立解答的子任务。例如，查询“对比 A 产品和 B 产品在性能和成本上的差异？”可被分解为四个独立的子查询：“A 产品的性能是什么？”、“B 产品的性能是什么？”、“A 产品的成本是多少？”、“B 产品的成本是多少？”。系

统对每个子问题独立进行检索，最后综合所有子问题的检索结果来形成最终答案。

- **效果：**将复杂的多跳（Multi-hop）推理问题，转化为一系列简单的单跳事实检索，显著提升了处理复杂查询的准确性和全面性。

3、假设性文档嵌入（HyDE）

该策略旨在通过一种创新的思路来跨越“短查询”与“长文档”之间的语义鸿沟。其核心思想是：不直接用用户的简短查询去匹配冗长的文档，而是先利用 LLM 生成一个“假设性的”理想答案文档，然后用这个内容更丰富、与真实答案在形式和语义上更接近的假设性文档去进行向量检索。

- **实现方式：**
  - 接收到用户查询后，提示一个强大的 LLM：“请针对以下问题生成一个详细的回答文档”。
  - 将LLM生成的这个“伪答案”进行向量化。
  - 使用这个“答案的向量”去匹配知识库中“真实答案的向量”。
- **效果：**巧妙地将“Question-to-Document”的匹配问题，转化为了“Document-to-Document”的匹配问题，由于两者在形式、长度和语义丰富度上更加对等，从而显著提升了在零样本（zero-shot）场景下的检索精度。

4.5.2 多路召回与混合检索

在拥有了高质量的查询后，下一步是从索引库中找出所有相关的候选文档。单一的检索策略往往存在短板，因此采用多路召回与混合检索的策略，是保证召回广度与精度的关键。

1、向量检索与关键词检索的协同

现代 RAG 系统中最经典的混合检索组合，就是向量检索（密集检索）与关键词检索（稀疏检索，如 BM25 算法）的协同。

- **向量检索：**核心优势在于其强大的语义理解能力，能很好地处理同义词、近义词等问题。
- **关键词检索：**核心优势在于其字面精确匹配能力，对于专有名词、产品型号、代码片段等需要精确匹配的场景表现出色。
- **协同方式：**系统并行执行这两种检索，各自生成一个候选文档列表。这样既利用了向量检索的模糊语义理解能力，又利用了关键词检索的精准字面匹配能力，实现了优势互补。

2、倒数排名融合（RRF）

当从多路召回中获得了多个独立的、排好序的文档列表后，需要一种有效的策略将它们融合成一个最终的排序结果。倒数排名融合（Reciprocal Rank Fusion, RRF）是一种简单而极其有效的策略。

- **核心思想：**一个文档如果在多个不同的检索结果列表里都排在靠前的位置，那么它很可能是一个非常相关的文档。RRF算法忽略了不同检索系统的原始相关性分数，而是基于文档在各个列表中的排名（Rank）来计算一个融合后的新分数。一个文档在不同列表中的排名越靠前，其最终的融合分数就越高。
- **效果：**RRF 无需复杂的参数调整，鲁棒性强，能够有效地整合来自不同信息源的信号，优先展示那些被多方共识为重要的文档，显著提升检索结果的整体质量和鲁棒性。

4.5.3 结果精炼与上下文重组

经过查询增强和多路召回，我们获得了一个比原始检索更相关、更全面的候选文档池。然而，这个阶段的结果仍然可能存在噪声，且直接拼接作为上下文可能会破坏信息的完整性。因此，结果精炼与上下文重组是通往高质量生成的最后一道关键工序。

1、重排序（Rerank）

重排序是在召回（Recall）的基础上进行的一次更精细、更昂贵的二次排序过程。它使用一个更强大、更复杂的模型（如交叉编码器）来对查询与每个候选文档进行深度交互，从而计算出一个比向量相似度远更精准的相关性分数。

- **核心技术：**交叉编码器（Cross-Encoder）。与检索阶段将查询和文档独立编码的双编码器（Bi Encoder）不同，交叉编码器将查询和候选文档同时输入到一个强大的Transformer模型中，使其能
够充分捕捉两者之间复杂的深层交互关系，从而给出一个远比向量相似度更精准的相关性分数。
- **流程：**
  - 使用高效的召回方法（如混合检索）快速召回一个相对较大的候选集（如 Top 100）。
  - 将这个候选集和原始查询送入交叉编码器模型，逐一进行深度计算和打分。
  - 根据新的分数重新排序，并选择排名最高的 Top-K 个文档作为最终上下文。
- **效果：**在保持高效召回的同时，极大提升了最终上下文的精准度，有效减少了噪声，是提升生成答案质量、减少幻觉的关键步骤。

2、自动合并检索

该策略旨在解决传统分块方法可能导致的语义割裂问题。其核心思想是在数据处理阶段就建立起文档块之间的层级关系，并在检索后智能地将离散的子块合并回更完整的父块。

- **实现方式：**在索引阶段，将文档解析为层级结构（如父块-子块）。检索时，系统在细粒度的子块上进行精准定位。如果发现有多个被召回的子块都指向同一个父块，系统会判定这个父块所代表的、更完整的上下文可能对回答问题至关重要，于是会用这个父块的完整内容，替换掉所有被召回的、属于它的子块。

- **效果：**动态地、智能地调整提供给 LLM 的上下文粒度。既能利用小块进行精准定位，又能在必要时“向上追溯”，提供一个更大、更完整的语境，确保了信息的连贯性和完整性，从而极大地提升了 RAG 的生成质量。

综上，一个高性能的 RAG 检索流程，是通过查询增强、混合检索与结果精炼等一系列环环相扣的优化措施，共同构建的关键链路。它确保了从用户模糊的意图到模型精准的生成，整个信息流动的每一步都尽可能地减少信息损失和噪声引入，最大化知识的利用效率。

当前，这些优化策略主要聚焦于如何更准确地检索事实。然而，AI 应用的未来将要求系统不仅能找到信息，更能进行复杂的推理。下一节，我们将探讨RAG架构的前沿演进，看它如何融合更先进的技术，从事实检索迈向推理式检索的更高阶智能。

4.6 RAG 的未来方向

通过前文的探讨，我们已经清晰地看到，RAG 已经从一个简单的检索-生成模型，演变为一套复杂而精密的系统工程。它不仅是提升 LLM 事实准确性的关键补丁，更是上下文工程的核心支柱。然而，技术演进的步伐从未停歇。发展至今，RAG 技术正沿着三条主要路径加速发展，共同推动其从一个被动的静态检索工具，向一个主动的、多维的、具备推理能力的智能知识基础设施跃迁。

这三大演进方向是 Agentic RAG、多模态 RAG 以及与知识图谱的深度融合，共同预示着下一代 AI 原生应用的核心能力：即一个能够自主规划信息需求、理解并交互于多元世界、并在结构化知识之上进行深度推理的强大认知内核。

4.6.1 Agentic RAG

Agentic RAG 的核心思想，是将信息检索的主动权从 AI 应用开发者手中，移交给 Agent 本身。在传统 RAG 中，检索是一个被动、前置的步骤；而在 Agentic RAG 中，检索被封装成一个智能体可以自主调用、评估和迭代的工具（Tool）。

1、核心变革

智能体基于 LLM 的大脑进行判断，决定是否需要检索、何时检索、检索什么以及如何利用检索结果。信息的获取不再是固定的前戏，而是融入了智能体自我判断与迭代的闭环之中。智能体可能会：

- **自我评估：**首先判断自身知识是否足以回答问题，若不足，则触发检索。
- **查询重构：**对用户的原始查询进行调整和优化，以获得更好的检索效果。
- **迭代检索：**对首次检索的结果进行评估，如果不满意，会调整查询再次检索，直至找到满意的信息或达到最大迭代次数。
- **动态规划：**针对需要多步推理的复杂问题，自主规划一系列检索步骤。

2、价值与挑战

Agentic RAG 的模式更适合处理动态、开放域、需要多步推理的复杂查询场景。然而，这也带来了更高的实现成本和安全风险。如何设计完善的工具权限与审计机制，如何提升模型对工具选择的准确性，以及如何确保整个系统的鲁棒性和安全边界，是其在工程落地中必须面对的核心挑战。

4.6.2 多模态 RAG

随着多模态大模型的兴起，RAG 的知识边界也正从纯文本，扩展到图像、表格、音频、视频等更丰富的数据模态。对于企业而言，大量的知识资产并非以纯文本形式存在，多模态 RAG 的演进，将极大地提升这些资产的利用率和检索准确性。

1、核心变革

混合模态 RAG 旨在通过统一的嵌入与检索机制，将文本、图像、语音等不同类型的内容纳入同一个语义空间。

- **跨模态检索：**支持以文搜图、以图搜文等跨模态的检索能力。
- **复杂文档理解：**不再将图表丰富的文档简单视为纯文本，而是能精准地理解和检索其中的表格、图示和版式结构。
- **统一表示：**利用先进的多模态 Embedding 模型，将不同模态的数据映射到统一的向量空间中，实现无缝的语义检索。

2、价值与挑战

多模态 RAG 是解锁图表、版式复杂文档、多媒体、行业资料等商业价值的关键。然而，在检索形式上，当前落地案例仍以文搜图、文搜视频为主，而更复杂的图搜文、视频搜文等场景在企业级应用中仍鲜有成熟案例。如何高效处理和索引海量的非结构化多媒体数据，以及如何提升跨模态检索的精准度，是其面临的主要技术挑战。

4.6.3 知识图谱与 RAG 的融合

向量检索的核心在于捕捉语义相似性，而知识图谱（Knowledge Graphs）的优势在于显式地建模实体之间的结构化关系。将两者融合，旨在将 RAG 从查文本片段的模式，升级到懂知识关联的更高维度。

1、核心变革

基于知识图谱索引的 RAG 技术，通过将实体、关系和属性建模为图结构，与传统的向量、关键词检索协同，实现了双轨驱动的范式。

- **缓解“语义鸿沟”：**让系统能进行多跳路径遍历与关系约束，构建可解释的推理链并对答案进行证据溯源。
- **双轨范式：**将语义相似度驱动的片段召回升级为结构化推理+语义匹配的双轨范式。
- **应用场景：**在法律、医疗、金融等强调逻辑一致性与合规性的场景中，其价值尤为突出。例如，可以直接回答“糖尿病的并发症有哪些？”这类需要多步推理的问题。

2、价值与挑战

该融合技术极大增强了 RAG 处理需要复杂推理和高解释性问题的能力。然而，在企业级生产实践中，将海量企业级知识库转化为高质量的知识图谱，需要消耗大量的算力和时间。知识的动态更新也因此十分困难。如何在算法上精进构建知识图谱的效率，或在算法策略与图谱效果中进行动态取舍，是影响该技术未来能否广泛落地的关键因素之一。

面向未来，Agentic RAG、多模态 RAG 以及与知识图谱的融合，这三大方向共同将 RAG 从一个功能组件，升级为AI原生应用不可或缺的智能知识基础设施。

- **Agentic RAG：**提供了任务导向的自主规划与自我校验能力。
- **多模态 RAG：**扩展了外部大脑的感知边界。
- **Graph RAG：**增强了结构化推理与可追溯合规的能力。

三者协同演进，共同指向一个目标：提升知识获取的效率，增强生成内容与回答的可靠性。对于开发者而言，在实际落地中，应综合考虑业务的数据形态、风险约束与成本目标，进行合理的架构选型与组合优化。结合有效的工程治理能力，才能让好用且可管的 RAG 真正走向规模化，并在 AI 原生时代创造更大的价值。

ALIBABA CLOUD

4.7 上下文管理与记忆系统

前面的章节，我们深入探讨了如何通过 RAG 为 LLM 动态注入外部知识，这极大地增强了其回答问题的准确性和时效性。然而，当 AI 应用从问答式工具进化为能够自主规划、执行长流程任务的 Agent 时，我们面临一个全新的、更深层次的上下文挑战：状态管理与记忆。

智能体并非一次性的问答机器，而是一个需要长期运行、持续交互的自治系统。它必须能够记住“我是谁”、“我正在做什么”、“我过去做了什么”以及“我了解你什么”。如果缺乏这种状态和记忆，智能体将陷入失忆困境，无法执行任何需要跨越多步骤或多次交互的复杂任务。因此，一个强大而可靠的记忆系统，以及在此基础上的一整套运行时上下文管理策略，是构建真正智能体的核心，也是上下文工程在高级应用中的集中体现。

4.7.1 核心挑战：有限的工作记忆

LLM 的上下文窗口是智能体唯一的工作记忆，它进行思考、推理和决策的操作空间。尽管这个窗口越来越大，但它依然是有限、有成本且存在性能瓶颈的。对于一个需要长时间运行的智能体而言，将所有历史交互、工具调用结果、中间思考步骤全部塞入上下文，会迅速导致以下问题：

- **成本与性能下降：**更长的上下文意味着更高的 Token 使用成本和更慢的推理速度。
- **中间遗忘：**正如前文所述，LLM 在处理长上下文时，容易忽略中间部分的信息，这对于需要依赖长序列信息的复杂任务是致命的。
- **信息噪声：**过多的无关信息会稀释关键信息的浓度，干扰智能体的注意力，导致其偏离核心任务目标。

因此，智能体的上下文管理，其核心就是一套关于如何高效、智能地利用这个有限工作记忆的策略组合。

4.7.2 运行时上下文处理的四大策略

根据上下文工程的研究与实践，智能体在运行时的上下文处理策略可归纳为四大类：写入（Write）、选择（Select）、压缩（Compress）和隔离（Isolate）。

1、写入：将信息保存到外部记忆

该策略指将信息保存到上下文窗口之外，以便未来使用，这是构建记忆系统的基础。

- **暂存区（Scratchpads）：**它特指LLM在生成下一步行动前，被引导输出的中间推理过程（即“思想链”，Chain of Thought）。这些思考步骤被记录下来，并与任务的观察结果一同作为下一轮推理的输入，从而形成一个连贯的思考与行动循环。
- **记忆（Memories）：**帮助智能体跨越多个会话记住信息的持久化存储，如通过反思（Reflexion）模型在每次执行后生成的自我总结和经验。

2、选择：将相关信息拉入上下文

该策略指在需要时，精准地将最相关的信息从外部拉入当前的上下文窗口。

- **记忆检索：**智能体根据当前任务，从长期记忆库中选择最相关的记忆。这通常利用向量检索（用于语义相关性）和/或知识图谱（用于实体关系）来实现。
- **工具选择：**当智能体需要使用工具时，通过 RAG 技术，根据当前任务的语义描述，从众多可用工具中选择最匹配的几个，将其 API 描述放入上下文，供智能体决策调用。

3、压缩：为上下文瘦身

该策略指在保留核心信息的前提下，减少上下文中的token数量。

- **上下文摘要：**使用 LLM 对冗长的对话历史或工具调用返回的密集信息进行总结，生成一个简短的摘要。
- **上下文修剪：**通过过滤或修剪上下文，移除旧的或不重要的信息，例如只保留最近 N 轮的对话历史。

4、隔离：拆分与保护上下文

该策略指将上下文进行拆分，以帮助智能体更好地执行任务。

- **多智能体系统：**将复杂任务拆分给多个子智能体，每个智能体拥有独立的、更小的上下文窗口，专注于自己的子任务。
- **状态对象：**通过定义结构化的模式（如 Pydantic 模型），在每个回合中只将指定的关键字段暴露给 LLM，而其他信息则保持隔离，避免干扰。

4.7.3 构建智能体的多级记忆系统

上述运行时策略的有效执行，依赖于一个设计精良的多级记忆系统。这个系统通常分为短期记忆和长期记忆两个层面，共同构成了智能体的认知基础。

1、短期记忆：管理当前对话

短期记忆管理着当前任务会话的完整上下文，它不仅包括对话历史，还应涵盖最近的工具调用结果、当前的执行计划和中间结论等。

- **管理策略：**
  - 滑动窗口：维护一个固定大小的窗口，只保留最近N轮的对话。
  - 关键信息保留：识别并优先保留对话中的关键信息，如用户的明确指令、重要的实体等。
  - 定期清理：定期清理无关或冗余的对话内容。

2、长期记忆：沉淀持久化知识

长期记忆存储跨越多个对话的持久化知识库，如用户偏好、过去项目的摘要、需要长期记住的事实等，是实现个性化和长期连贯性的关键。

- **构建策略：**
  - 定期提取：定期从短期记忆中提取关键信息，并合成为长期记忆。
  - 高效检索：使用向量嵌入和向量数据库实现高效的语义检索。
  - 版本管理：实现记忆的版本控制和更新机制，确保记忆的准确性。

3、记忆转换时机

确定何时将短期记忆转化为长期记忆，是一个关键的决策点。常见的时机包括：

- **对话自然结束时：**提取关键信息保存为长期记忆。
- **识别到重要特征时：**当系统识别到重要的用户偏好或个人信息时，主动进行保存。
- **定期摘要：**定期（如每5轮对话）对短期记忆进行总结并存入长期记忆。

对于 Agent 而言，上下文管理与记忆系统并非两个独立的组件，而是其认知核心的一体两面。运行时上下文管理是战术层面的操作，它决定了智能体在此时此刻如何最高效地利用其工作记忆；而多级记忆系统则是战略层面的支撑，它为智能体提供了历史感和个性化的基础。

通过将这两者有机结合，上下文工程使得智能体能够摆脱无状态的束缚，成为真正能够处理复杂任务、提供个性化服务的智能伙伴。这是 AI 应用从能用走向可靠和好用的关键一步。

AI 工具

AI Tools

CHAPTER

05

AI 工具简介

AI 工具标准化

MCP 实践

P133-P156

06

AI Gateway

P159-P202

07

AI Runtime

P205-P224

08

AI Observability

P227-P256

5.1 AI 工具简介

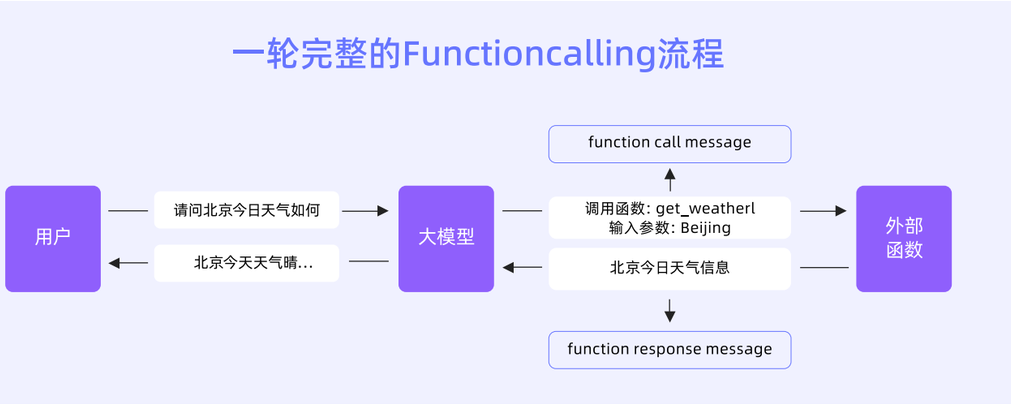
大模型虽然越来越聪明，但是缺少工具向物理世界进行延伸，它对物理世界既不可读也不可写。因此，业界才通过工具调用（Function Calling）、检索增强（RAG）、以及具备计划与执行能力的代理式工作流（Workflow）来弥补模型与现实世界之间的鸿沟，把智能从会说推进到能做。

从原理上看，通用大模型善于在给定上下文中生成连贯文本，但它并不具备内建的 I/O 与状态管理，也无法直接访问数据库、API、知识库或执行系统命令。为了让模型“办成事”，我们通常要让它做三件事：通过 RAG 读取外部知识、通过工具调用触发外部动作、通过代理式规划协调多步流程。这些机制把世界的读写权以受控的方式暴露给模型，让其在安全边界内发挥效能。

然而，Function Calling 方案在工程落地上暴露出大量痛点，阻碍了规模化与治理化的推进。随着 MCP（Model Context Protocol）协议的提出并迅速获得关注，行业看到了以通用协议重塑模型&外部能力接口层的可能。

5.1.1 Function Calling

Function Calling (函数调用) 是一种允许 LLM 根据用户输入识别它需要的工具并决定何时调用该工具的机制。LLM 接收用户的提示词，LLM 决定它需要的工具，执行方法调用，后端服务执行实际的请求给出处理结果，大语言模型根据处理结果生成最终给用户的回答。



如下是阿里云百炼平台 Function Calling 的方法：

Python | 复制代码

```python
# 检查是否需要调用函数
if run.required_action:
    for tool_call in run.required_action.submit_tool_outputs.tool_calls:
        if tool_call.function.name == "translate_text":
            args = json.loads(tool_call.function.arguments)
            translation = translate_text(args["text"], args["target_langua
ge"])

            # 提交工具输出
            Runs.submit_tool_outputs(
                thread_id=thread.id,
                run_id=run.id,
                tool_outputs=[{"tool_call_id": tool_call.id, "output": tra
nslation}]
            )

            # 等待新的运行完成
            run = Runs.wait(thread_id=thread.id, run_id=run.id)
```

从上面步骤中可以看出来，Function Calling 使用起来非常不方便，需要编写代码，因为是自定义的调用方式，因此很难在不同的平台复用。

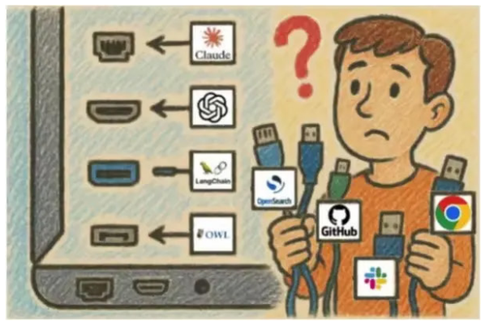
总结来说，Function Calling 存在如下4部分问题：

- **规格碎片化。**不同厂商对工具的描述方式、类型系统与 JSON Schema 兼容度、调用时序、错误语义、流式行为都各不相同。即使是类似的功能，在 A 厂商上表现为同步调用、在 B 厂商上却需要异步轮询；有的支持严格校验与参数默认值，有的则完全凭提示工程“约法三章”。这使跨厂商、跨模型的适配成本居高不下。
- **可靠性挑战。**模型常常会错误地选择工具或以错误顺序调用。即便选对了工具，也可能生成不满足约束的参数，出现结构缺失、类型错误、边界值越界，甚至编造参数。多步或并行调用尤为脆弱：中途失败无法优雅回滚，调用结果彼此覆盖，流式响应时序相互干扰，最终让上层业务出现不可预测的行为。
- **工程治理缺位。**大多数团队缺少集中式的工具目录与自描述元数据，更谈不上版本与依赖管理、权限分级、调用审计、成本归集。工具的定义散落在各个微服务与提示模板里，难以复用、难以维护、无法衡量 ROI，也难以符合企业的合规与安全要求。
- **厂商锁定与迁移成本高。**由于工具适配逻辑深度绑定在某一家模型供应商的接口与语义上，想要在多模型、多云之间切换就需要重复实现同一套工具封装，迁移周期长、风险大，阻碍了

“按需选模、按价择模”的策略。

5.1.2 MCP

或许是大家也看到了当前大模型应用数据源接入的混乱现状，在2024年11月初，Anthropic 提出了他们新的模型上下文协议（Model Context Protocol，简称 MCP）。旨在“整治 LLM 数据源接入混乱现状，打通构建 Agent 的最后一公里”。MCP 是一个开放标准，帮助连接 AI 助手与数据所在的系统，包括内容存储库、业务工具和开发环境。其目标是帮助前沿模型产生更好、更相关的响应。



Function calling

MCP

图片来源：https://www.sequoiacap.com/article/ai-ascent-2025/

MCP 的核心理念是把模型可用的上下文与外部能力协议化，形成独立于模型供应商的标准接口层。它把工具（Tools）、资源（Resources/用于检索与长上下文读写）、提示（Prompts）、会话（Sessions）、取样/推理（Sampling）等能力统一纳入协议语义，并标准化“枚举–描述–请求–响应–权限–审计”的全链路。传输层上，MCP 不限定具体载体，既可通过 HTTP/WebSocket，也可通过 stdio 等方式交换消息，使之既能嵌入云端推理服务，也能运行在本地或边缘环境。

MCP 可以看作是 AI 应用程序的 "USB-C端口"。就像 USB-C 为连接设备与各种外设提供了标准化方式，MCP 为 AI 模型连接不同数据源和工具提供了标准化方法。

5.1.3 MCP 就足够了吗？

MCP 的价值在于用统一协议取代碎片化集成，把大模型获取外部数据与工具的方式从 N×N 适配转为“一次对接、处处可用”，显著简化并提升可靠性。它定义了代理可用的工具与上下文提供机制，既可本地运行也可通过远程服务器在云端共享能力，适合规模化分发与复用。MCP 在这个过程中定义了标准的协议，并且得到了业界的认可，我们看到大量的生态在短时间内支持了MCP协议，诸如 Cursor、Windsurf、Cline 等已开始引入 MCP，带动开发者与经典在线应用联动，甚至出现如 Blender MCP 这类将自然语言接入专业软件的场景，在这种趋势下，MCP 网关也如雨后春笋般出现，随着 MCP 相关的基础组件的出现与完善，MCP 被视为潜在的“AI 原生基础设施”。

当然归根结底来说，无论是 RAG 、Function Call 还是 MCP，其实都是为了让模型获取更多知识、调用更多工具来完成更复杂的事情。从技术演进看，MCP 承接了 RAG 与 Function Call的路径，主打跨模型兼容与统一标准，尽管仍处早期，但具备形成事实标准的“滚雪球”机会。同时，MCP 也被比喻为“AI 扩展坞”，通过协议化自描述终结工具调用碎片化，降低对接复杂度。

但 MCP 不是“万能药”，其普及仍取决于生态成熟度、落地实用性与产业响应，短期内存在不确定性与不同声音，且国内是否成为事实标准亦未定。不少批评者指出，仅有协议难以补齐当前模型与 Agent 的能力与可靠性缺口，推广仍需时间验证与配套工程实践。随着远程 MCP 服务器在云上共享工具，安全与治理（身份、授权、审计）要作为一等公民纳入平台设计，而不仅是通信层面的规范。尽管 MCP 在密钥暴露最小化、访问校验与传输加密等方面具备先天优势，真正安全可控仍依赖平台级策略与治理体系完善落地。

因此，更现实的结论是：以 MCP 为协议底座，叠加 AI 工具接入的最佳实践与企业级安全治理，才能把“可接入”变成“可规模、可移植、可审计”的 Agent 能力。

5.2 AI 工具标准化

登高而招，臂非加长也，而见者远。

在 AI 工具篇章的首节对 MCP 协议的技术定位与生态价值进行全景式剖析后，我们已然认识到这一协议在重塑 AI 应用架构中的战略意义。然而，任何技术标准的真正生命力都需在工程实践中验证。本节将从协议架构的微观实现出发，深入探讨 MCP 在工程化落地中的核心优势、现实困境及其破局之道。

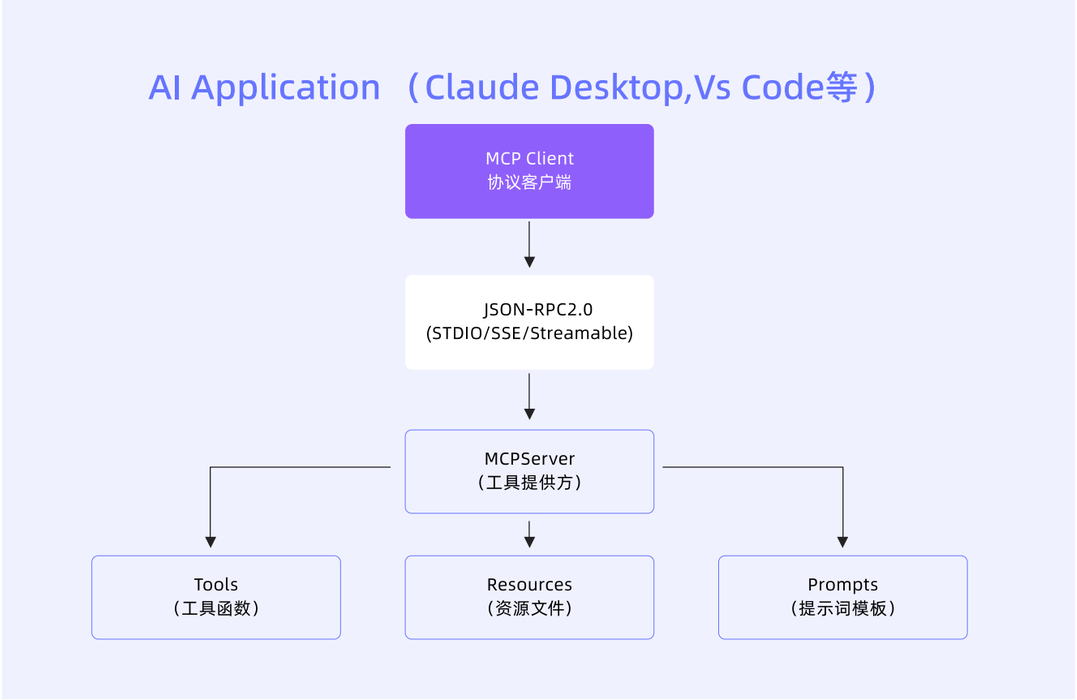
5.2.1 MCP 协议介绍

MCP（Model Context Protocol，模型上下文协议，下文中简称 MCP）是一个开放协议，旨在为模型服务提供标准化的接口，使其能够连接外部数据源和工具。MCP 通过统一规范取代了工具的碎片化集成，从而打破数据孤岛，提升 AI 应用的动手能力。这与 USB-C 的设计理念类似：USB-C 通过统一接口连接各类物理设备，MCP 也提供了一种标准化的方式，将 AI 应用连接到不同的数据和工具。

MCP 协议遵循客户端-服务器（Client-Server）架构，它允许 AI 应用程序通过 MCP 客户端与多个 MCP 服务器建立连接，实现灵活的上下文传递与功能扩展。MCP 架构包含下面几个部分：

- **MCP Host**：协调和管理一个或多个 MCP 客户端的 AI 应用程序，如 Claude Desktop、Visual Studio Code 等，需要通过 MCP 协议访问数据。
- **MCP Client**：MCP 客户端，运行在 MCP Host 内部，负责与 MCP Server 建立并维护连接。MCP Host 会为每一个 MCP Server 单独创建一个 Client，Client 与 Server 保持一对一连接。
- **MCP Server**：MCP 服务端，轻量级程序，通过 MCP 协议公开特定功能。MCP 协议规定了 MCP Server 可为 Client 提供工具、资源文件以及提示词模板。

JSON



MCP 协议采用 JSON-RPC 2.0 作为通信规范，所有消息均使用 JSON 格式进行序列化，确保跨语言和跨平台的兼容性。消息类型包括：

- **Request**：请求消息，包含方法名和参数，期望对方返回响应；
- **Result**：对请求的成功响应；
- **Error**：表示请求处理失败，包含错误码和错误信息；
- **Notification**：单向通知消息，无需响应。

以下是获取工具列表的请求和响应示例：
请求消息：

JSON

```json
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {}
}
```

ALIBABA CLOUD

响应消息：

```JSON
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "get_weather",
        "description": "Get current weather information for a location",
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City name or zip code"
            }
          },
          "required": ["location"]
        }
      }
    ],
    "nextCursor": "next-page-cursor"
  }
}
```

AI 应用程序与 MCP Server 交互的整体流程如下：

- **协议初始化**：AI 应用程序启用 MCP Client，MCP Client 向 MCP Server 发起初始化请求，附带 MCP 协议版本和基础上下文等信息。MCP Server 返回初始化响应，包含 MCP Server 名称、版本以及功能概览。
- **能力发现**：MCP Client 查询 MCP Server 的工具/资源文件/提示词模板列表，不同类型的标准方法如下：
  - tools/list：获取可调用的工具列表；
  - resources/list：获取可用的资源列表；
  - prompts/list：获取可用的提示词模版列表。
- **功能调用**：MCP Client 根据业务需要动态请求工具调用/资源文件/提示词模版，MCP Server 返回具体的响应内容。
- **会话终止**：交互完成后，MCP Client 和 MCP Server 均可以主动关闭连接。

5.2.2 MCP 的核心优势

MCP 重新定义了模型与外部世界的交互方式，解决了 AI 应用开发中模型与工具结合所面临的诸多痛点，它的核心优势主要体现在以下几个方面。

- **标准化模型与工具的连接方式**

MCP 为模型与外部数据源、服务之间提供了标准的通信方式，它就像 AI 领域的“USB-C”，模型可通过 MCP 协议适配多种工具。Function Calling 虽然实现了工具调用从手动到自动化的跃迁，但它需要用户为不同模型和服务开发适配逻辑的工作，开发和维护的成本较高。MCP 定义了统一的通信规范和数据格式，用户只需遵循 MCP 协议定义接口，就可以让模型与外部世界通信。这种标准化解耦了模型与工具开发，让开发人员更专注于工具本身的能力，降低了运维复杂度。

- **灵活的资源调度方式**

MCP 支持 STDIO、SSE、Streamable HTTP 三种连接模式，STDIO 可通过进程间通信访问本地的文件系统、数据库等资源，而 SSE、Streamable HTTP 能够支持对远程服务的调用。一个 AI 应用程序可以同时使用本地和远程两种方式调用工具，这种灵活性使得 AI 应用程序能够在分布式环境中自由、按需的调度资源，实现跨网络的数据交互。

- **支持双向、异步通信**

MCP 协议能够实现双向交互的能力，AI 应用程序可同时作为 MCP 客户端和 MCP 服务端，既能主动访问外部的 MCP 服务，又能将自身的能力封装为 MCP 服务供其他客户端使用。MCP 也支持异步通信模式，MCP 客户端在请求完成后可不用等待服务端的响应，当任务完成后由服务端主动推送异步事件，这种设计能够适配一些处理耗时较长的复杂任务，如视频解析等。

支持异步通信模式，MCP 客户端在请求完成后可不用等待服务端的响应，当任务完成后由服务端主动推送异步事件，这种设计能够适配一些处理耗时较长的复杂任务，如视频解析等。

- **支持能力共享**

MCP 协议能够实现双向交互的能力，AI 应用程序可同时作为 MCP 客户端和 MCP 服务端，既能主动访问外部的 MCP 服务，又能将自身的能力封装为 MCP 服务供其他客户端使用。MCP 也支持异步通信模式，MCP 客户端在请求完成后可不用等待服务端的响应，当任务完成后由服务端主动推送异步事件，这种设计能够适配一些处理耗时较长的复杂任务，如视频解析等。

- **构建方式便捷**

MCP 服务本身是一个轻量程序，开发门槛低，社区提供了像 MCP-Framework、Spring AI MCP 等 MCP 开发框架，简化了用户开发流程，如果结合 AI IDE 等工具，可进一步提高开发效率。将一个已有的服务升级为 MCP 服务所需的改动也非常小，通过 Higress 网关与 Nacos 等中间件，存量应用可以在不修改代码的前提下被封装为 MCP 服务，只需配置工具描述、参数定义等元信息即可。

5.2.3 MCP 面临的挑战

尽管 MCP 在技术愿景和生态共建上展现出巨大潜力，但企业在实际落地 MCP 应用的过程中也遇到了诸多问题。

- **安全问题**

MCP 协议的身份认证体系尚不完善，虽然官方已引入了基于 OAuth 2.1 的授权方案，提升了协议的安全性，但 Oauth 授权流程仍依赖开发者自行实现，且不同语言对 Oauth 功能的支持也存在差异，这无疑增大了开发复杂度和集成门槛。此外，提示词注入和工具投毒攻击等都是 MCP 服务中常见的高危风险，MCP 将大量系统提示词、工具描述等上下文信息直接添加到模型输入中，而模型本身无法识别信息的合法性，因此无法感知到上下文是否被恶意篡改。攻击者可利用这个漏洞，在工具的描述信息或返回数据中嵌入恶意指令，诱导模型执行非预期行为。

- **大规模应用问题**

当 MCP 服务或工具的数量过多时，模型可能会出现选择困难症，因为大量的上下文输入让模型难以区分和回忆每个工具的能力，也就无法有效的选择与目标问题关联度最高的工具。而且，由于模型在每次对话中都需要接收全量的工具描述信息，对话内容很容易超出模型支持的上下文窗口的长度限制。更关键的是，过长的提示词信息也会加剧 Token 的消耗速度，尤其是在多轮对话中，工具列表被重复传递，造成 Token 资源的浪费，拉高调用成本。

- **集中管理问题**

MCP 的通用性使得开放、共享 MCP 服务变得流行，用户通常会同时使用来自于不同平台的 MCP 服务，这种共享模式促进了 MCP 生态的发展，让开发人员避免了重复造轮子，但也带来了不少管理难题。不同平台 MCP 服务的管理方式不统一，调用关系错综复杂，人工维护、治理成本非常高，而且全局视角下的可观测能力、统一的计量计费能力等均因为跨平台问题而难以实现。

5.2.4 可行的解决方案

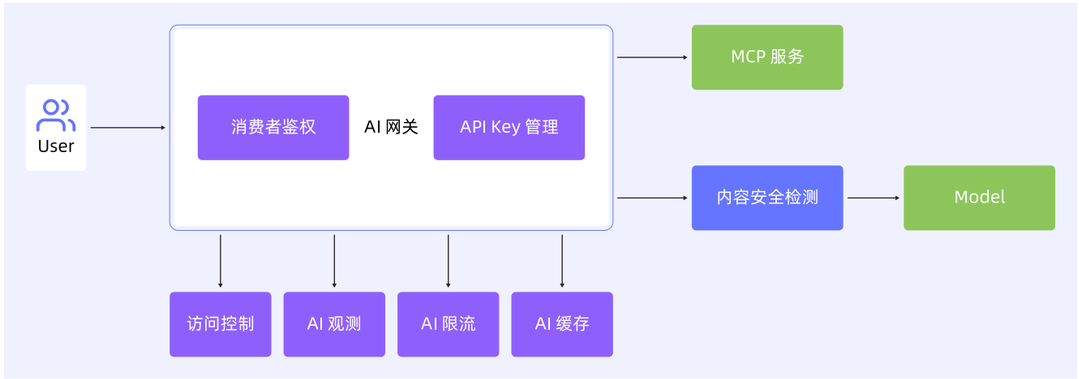
1、AI 网关解决 MCP 安全问题

阿里云 API 网关提供的 AI 网关能力支持 MCP 服务的流量代理，且能够在零代码改造的基础上将已有的 HTTP 服务转化为 MCP 服务。所有流经 MCP 服务的请求均由 AI 网关统一接入和处理，安全防护、流量管理、可观测等能力全部收敛在网关侧，让开发者能够专注于核心业务逻辑的实现。

在身份认证方面，AI 网关提供细粒度的消费者鉴权功能，网关会验证所有请求的合法性，它支持 API Key、JWT 等多种业界标准的认证方式，实现了从 MCP 服务到具体工具级别的权限控制。管理员可以为调用方分配专属的消费者凭证，结合网关的 AI 观测能力，可有效追溯每次工具的调用行为。对于安全需求更高的企业级场景，AI 网关还提供插件拓展能力，支持对接企业内部的认证服务或实现自定义的认证协议。对于密钥的安全管理，AI 网关与阿里云 KMS 服务深度集成，为企业提供安全可靠的密钥存储方案。

在内容安全防护方面，AI 网关集成了阿里云内容安全服务，可对请求和响应内容进行合规检测和敏感信息过滤，有效防御提示词攻击等恶意行为。在 MCP 应用中，网关会对传递给模型的所有上下文信息（包括 MCP 服务的配置、描述等）执行内容安全检测，筑起一道针对恶意指令的“免疫防线”，确保模型得到的提示词未受到“污染”。

在流量安全防护方面，AI 网关的 IP 访问控制可以在网络层阻断非法客户端的接入，限流、熔断、缓存等能力可以保障 MCP 服务的安全水位。AI 还网关集成了 Web 应用防火墙和 DDoS 防护等安全组件，可有效识别和阻断恶意攻击流量，确保 MCP 服务的稳定运行。

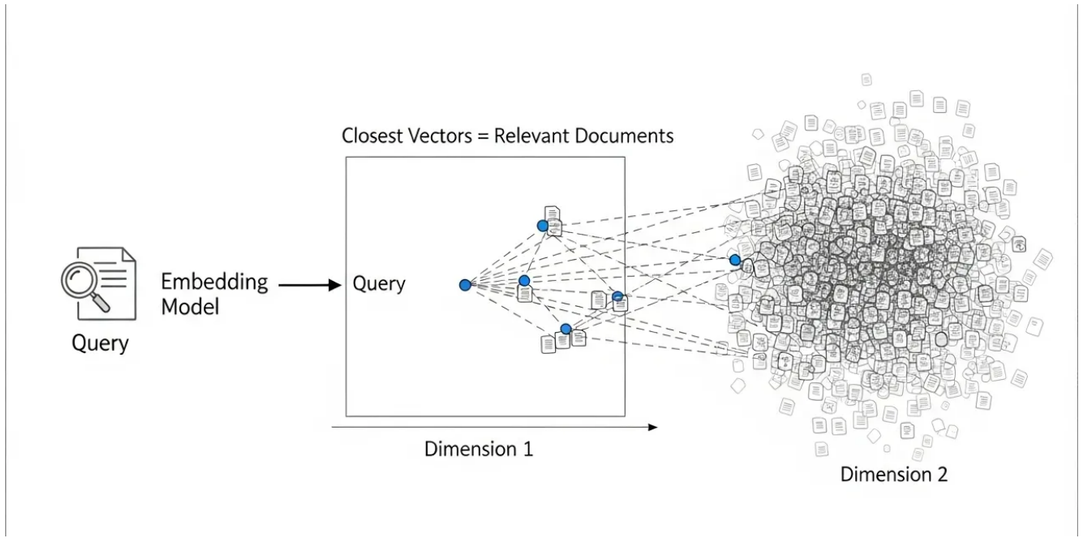


2、工具优选和语义检索提升批量 MCP 的调用质量、降低调用耗时

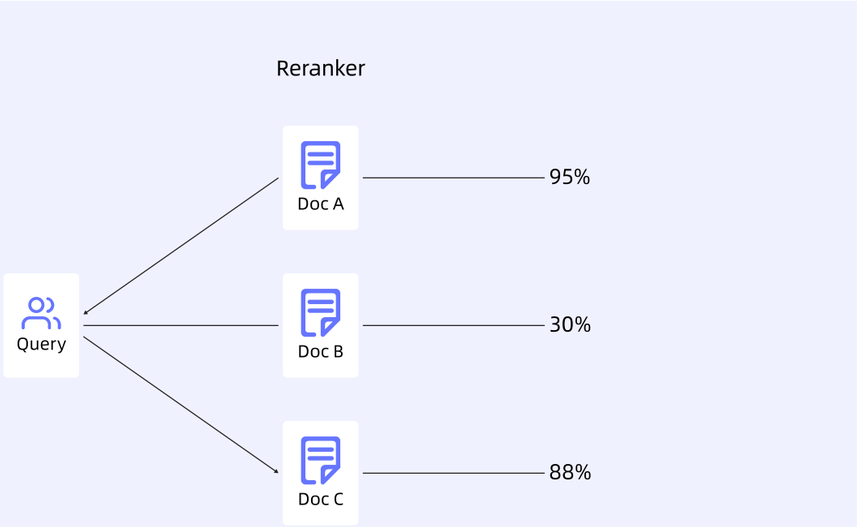
工具优选和语义检索是提升批量 MCP 调用质量的两种不同的技术实现方式。

工具优选是指当模型请求在经过网关调用 LLM 时，携带含有大量工具的 tool_calls 数组时，网关侧基于 Qwen3 Embedding 和 Qwen3 Reranker 的方案，提供工具的精选能力，将 tool_calls 的数量压缩至目标数量，以提升模型响应速度与工具选择精确性。

- **Qwen3 Embedding：**它的主要任务是将非结构化的文本转换成能够捕捉其语义信息的数值向量（即“嵌入”）。这些向量使得机器能够度量文本之间的相似性，从而可以快速地从大规模文档库中检索出与用户查询在语义上相关的候选文档。这个过程侧重于效率和广度（召回率），目标是在短时间内捞出所有可能相关的结果。



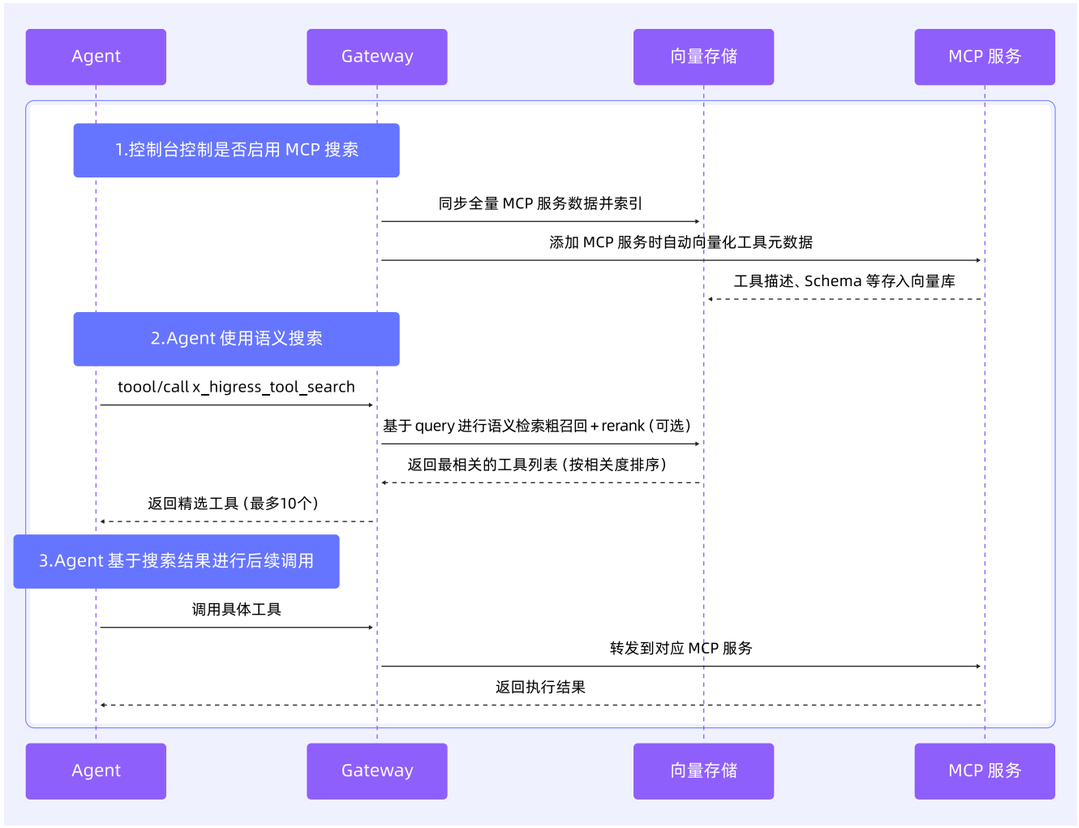
- **Qwen3 Reranker：**它的核心功能是优化和重排序一个已经经过初步筛选的文档列表。它会更精细地评估每个“查询-文档”对的相关性，并给出一个更准确的相关度分数，然后根据这个分数对文档进行重新排序，将最相关的结果排在最前面。这个过程侧重于准确性（精确率），目标是提升顶部搜索结果的质量。



语义检索则是基于 Higress 网关的 WASM 插件机制，通过创建一个 "All-in-One" 的 MCP Server，将用户在网关实例中注册的所有 MCP 工具进行统一聚合和管理，并提供智能的语义化检索能力。

- **统一入口设计：**通过 Higress 网关创建统一的 MCP 服务入口，所有 Agent 通过单一端点即可访问网关实例中的全部 MCP 工具，避免了分散管理多个 MCP Server 的复杂性。
- **智能工具发现：**内置 x_higress_tool_search 语义搜索工具，基于 Dash Vector 向量数据库和 Qwen 系列模型，提供精准的工具推荐能力。Agent 可以通过自然语言描述快速找到最相关的工具，而无需了解具体的工具名称。
- **双重检索保障：**采用"Embedding 粗召回 + Rerank 精排"的双重检索机制。首先通过 Qwen Embedding 模型进行向量相似度检索，获取候选工具集合；然后可选择性地使用 Qwen Rerank 模型进行精确排序，确保返回给 Agent 的工具列表质量最优。
- **实时数据同步：**建立完善的 MCP 工具生命周期管理机制，当用户在控制台中新增、修改或删除 MCP Server 时，系统自动进行工具元信息的采集、向量化和存储，确保向量数据库与实际 MCP 服务保持实时同步。

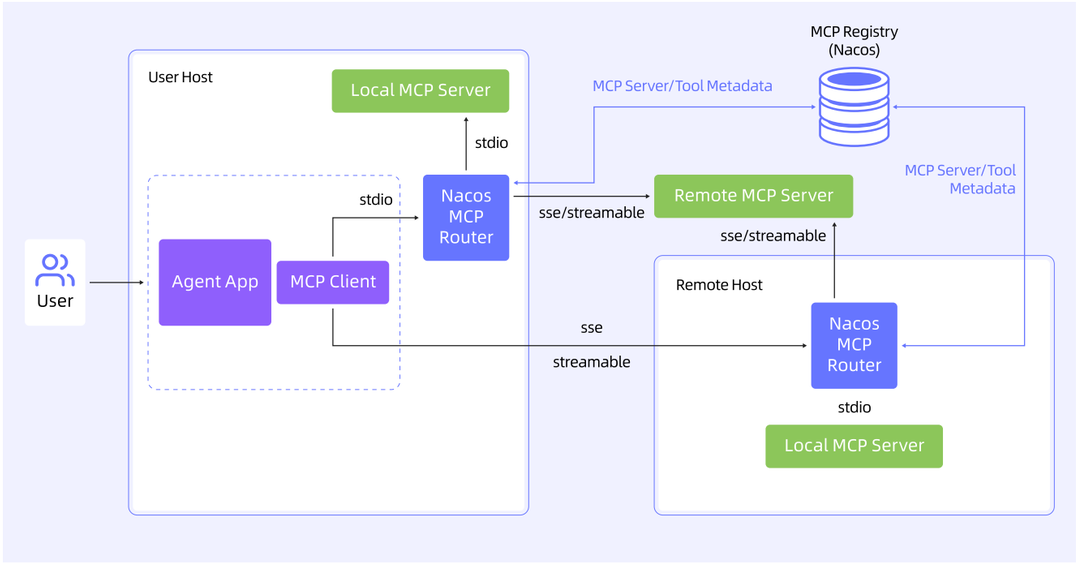
- **控制台一体化：**用户只需在控制台中开启语义搜索的开关，系统即可自动完成 DashVector 集合创建、模型配置、路由下发、数据同步等全流程操作，提供开箱即用的用户体验。



3、Nacos 解决“MCP 爆炸”问题

当 MCP 服务、工具的数量过多时，模型容易产生“工具幻觉”，针对这个问题，Nacos 3.0 提供了 MCP Registry 和 MCP Router（下面简称 Router）的能力。MCP Registry 允许用户将已有的 MCP 服务统一注册到 Nacos 上，Router 则可以根据用户任务的语义描述和关键词，从 MCP Registry 中筛选出最匹配的 MCP 服务，然后将这些服务提供给模型进行决策。

Router 是一个标准的 MCP 服务，引入 Router 后，用户不再需要手动的为不同任务配置不同的 MCP 服务，而是只需接入 Router 这一个 MCP 服务即可，极大的简化了配置的复杂度。更重要的是，Router 显著优化了 Token 使用效率。初始阶段，AI 应用程序仅需向大模型传递 Router 自身的轻量级工具描述，避免了全量工具信息的冗余传输。在实际执行过程中，Router 会动态匹配并仅返回与当前任务相关的 MCP 工具描述，从而大幅减少上下文长度，有效缓解因工具描述过多导致的 Token 消耗问题，降低推理成本，提升响应速度。



4、HiMarket 解决 MCP 管理问题

HiMarket 是一个开箱即用的 AI 开放平台，可用于帮助用户实现 MCP 服务的集中化管理。作为一个统一的管控面，HiMarket 能够纳管来自于不同系统的 MCP 服务。平台底层集成了各个系统的配置管理能力，且屏蔽了具体的操作差异，用户可以以一套管理方式在 HiMarket 上完成 MCP 服务的配置、发布、开放、运营等工作，将所有的管控操作收敛在一个平台，解决服务分散、管理混乱的问题。

HiMarket 与 AI 网关深度集成，可为 MCP 服务提供安全管控能力。用户可以在 HiMarket 上为 MCP 服务统一配置身份认证、设置流控策略、管理订阅授权等。平台提供了多维度的业务观测能力，支持从全局视角查看 MCP 服务的调用情况，便于用户定位问题以及分析系统的性能瓶颈。

HiMarket 还支持用户构建私有的 MCP 市场，实现 AI 能力的产品化运营。用户可以创建定制化的 MCP 门户，并将 MCP 服务封装成标准的产品，搭配上使用文档、安全防护策略，最后发布到门户上，供企业内外的开发者使用，促进业务创新。在能力变现方面，HiMarket 提供了灵活的计量计费规则，帮助用户实现 MCP 服务的货币化。

MCP 不仅是技术协议，也是一种生态愿景，它打破了 AI 模型与现实世界之间的壁垒，让大模型真正成为“能看、能听、能操作”的智能体。虽然目前在企业内大规模落地仍存在不少挑战，但这是技术迭代过程中的必经之路，也是推动技术演进的持续动力。MCP 在不断的发展和完善，其标准化、模块化、可复用的设计理念，已为 AI 应用的爆发奠定了基础。未来，随着更多开发者的加入和技术的持续创新，MCP 必将释放出更大的潜力。

5.3 MCP 实践

纸上得来终觉浅，绝知此事要躬行。

在大多数成熟的企业中，许多有价值的信息并非存在于公开的互联网，而是沉淀在企业内部，包括数据库里的业务数据、API 背后的服务能力、文档库中的规章制度与知识沉淀...这些资源构成了企业运营的数字基石。

那么，如何打破 AI 与企业内部数字世界之间的壁垒呢？答案在于通过标准化的协议（如 MCP）实现资源的统一接入与智能化封装。本节将以 MCP 为主线，探讨把传统资源改造为可供 LLM 实时调用、理解和利用的动态知识与能力。这不仅是技术的升级，更是企业迈向深度智能化，让 AI 真正融入业务流程、赋能决策的关键一步。

5.3.1 构建 MCP 的实践

理论付诸实践，企业可以通过多种路径构建 MCP Server，以实现对内部资源的智能化封装。以下我们将介绍从零构建和基于存量资源改造这两种实践路径。

1、从零快速构建 MCP Server

- **基础环境与技术选型**

技术选型上，TypeScript/Python/Java/Kotlin/C# 作为主流开发语言，均拥有官方/社区 SDK 支持；开发框架上，为避免重复造轮子，社区涌现了众多优秀的开源 MCP Server 框架，它们极大地简化了开发流程。

- **MCP-Framework (TypeScript)**：一个专门为 Claude MCP 协议设计的开发框架，提供了自动化工具、资源和提示加载等功能，是快速搭建Node.js环境MCP Server的利器。
- **Spring AI MCP (Java)**：作为Spring生态的一部分，它与Spring Boot无缝集成，允许开发者通过注解驱动的方式，轻松地将现有Java服务暴露为MCP工具。
- **FastMCP (Python)**：基于Python的MCP框架，用于快速构建MCP服务器和客户端，旨在简化MCP协议的实现，提供高效、简洁的开发体验。

- **核心逻辑实现**

核心功能上，重点聚焦 Tool 的实现。一个 Tool 本质上是一个可被 LLM 调用的函数，它必须包含清晰的名称、描述以及参数定义。以下是利用不同框架构建一个“查询天气”工具的示例：

- **MCP-Framework（TypeScript）示例**

通过声明式对象来定义一个Tool，结构清晰，语义明确。

TypeScript示例–MCP工具 TypeScript

```typescript
import { Tool } from "mcp-framework";

export default new Tool({
  name: 'getWeather',
  description: '获取指定城市天气数据',
  parameters: {
    city: {
      type: 'string',
      description: '城市名称，如：杭州'
    }
  },
  async execute({ city }) {
    // 核心实现逻辑
  }
});
```

- **Spring AI MCP（Java）示例**

支持注解驱动，利用 @Tool 注解，将一个普通的 Java 方法转化为 AI 可用的工具。

Java示例–MCP工具 Java

```java
@Service
public class WeatherService {
    // 一些属性定义

    @Tool(description = "获取指定城市天气信息")
    public String getWeather(
        @ToolParameter(description = "城市名称，如：杭州") String city) {
        // 核心实现逻辑
    }
}
```

- **FastMCP（Python）示例**

FastMCP 提供了一个 @mcp.tool() 装饰器对实现函数进行装饰即可，函数名称作为工具名称，参数作为工具参数，并通过注释来描述工具与参数以及返回值。

Python示例–MCP工具 Python

```python
from mcp.server.fastmcp import FastMCP

'''初始化MCP服务实例'''
mcp = FastMCP("weather-service")

'''定义工具函数'''
@mcp.tool(description = "获取指定城市天气信息")
def getWeather(city: str) -> str:
    """
    查询指定城市天气
    Args:
    city:城市名称，如：杭州

    Returns:
    包含城市温度、天气状况的字符串
    """
    # 核心实现逻辑
```

- **LLM：**除了利用以上开源框架外，开发者也可以借助 Claude 等强大的 LLM来 辅助生成 MCP Server 框架代码，进一步提升开发效率，具体可参考 MCP 官方提供的教程示例：

https://mcp-docs.cn/tutorials/building-mcp-with-llms。

- **传输协议实现**

传输协议上，MCP 支持两种通信方式。

- **stdio（标准输入/输出）：**适用于本地开发和调试。MCP Server 作为一个子进程运行，通过标准 I/O 与客户端（如本地的 VS Code 插件）通信。这种方式简单直接，但无法被网络上的其他服务调用。
- **SSE (Server-Sent Events)：**基于 HTTP 长连接实现，是生产环境的推荐方式。它将 MCP Server 暴露为一个标准的 HTTP 服务，可以部署在服务器上，供远程的 AI Agent、应用后端等进行访问和调用，具备良好的扩展性和通用性。

- **调试与部署**

开发完成后，可使用 MCP Inspector 或者集成了 MCP 客户端功能的 IDE 插件或应用（Claude Destop、VS Code 等）进行功能调测，验证 Tool 的描述是否清晰、参数是否正确、返回是否符合预期。最后，将构建好的 MCP Server 部署到本地环境或服务器环境。

2、基于存量 OpenAPI 构建 MCP Server

对于拥有大量存量 OpenAPI 的企业而言，逐一重写业务逻辑是不现实的。更高效的策略是构建一个“适配层”，将现有的 OpenAPI 能力自动或半自动地转化为 MCP Server。这种方式无需从头开发业务逻辑，只需要构建一个中间转转换层，将 MCP 协议的请求转换为对现有 OpenAPI 的调用，并将 API 的响应转换为 MCP 协议的格式，其常规步骤包括：

- **OpenAPI解析：**读取并解析存量 OpenAPI 规范文档，自动提取 API 路径、请求方法、请求参数、响应格式和认证方式等元数据。
- **MCP 映射与定义：**根据解析出的元数据，按照 MCP 规范自动生成 MCP 中新的描述。例如，将 API 中执行类操作，例如获取天气、操作网页等，定义为 MCP 的Tool；将 API 中返回的数据/可复用或频繁读取的数据，例如定位信息，映射为 MCP 的Resource。
- **请求/响应转化：**
  - 解析 MCP 调用：监听并解析来自MCP客户端的 tool/call JSON-RPC 2.0请求。
  - 构建调用 API：通过配置好的参数映射信息、路径、后端地址等信息，匹配到对应的 OpenAPI 定义，动态构建 HTTP 请求（包括 URL、Header、Body），并调用真实的后端 API 服务。
  - 封装 MCP 响应：将 API 返回的响应数据包装为 MCP 协议要求的标准 tool/call 结果格式，返回给客户端。

然而，通过上述方式手动将批量 OpenAPI 转化为 MCP Server 是一个需要时间和人力的繁重体力活。幸运的是，为提高生产和转化效率，成熟的 API 网关和开源工具为此提供了自动化解决方案。

- **利用 Higress AI 网关实现零代码接入**

Higress 作为云原生 API 网关，提供了强大的 AI 能力支持，其 openapi-to-mcp 工具是实现存量 API 快速接入的典范。该工具可将存量 OpenAPI 自动转化为 MCP Server，输入一个 JSON，得到一个标准的 MCP 配置，由此支持开发者无需从零开始，即可快速高效构建 MCP Server。其优势在于：

- **零代码改造，接入便利：**开发者无需编写一行代码，即可将 OpenAPI 定义文件直接转换为 Higress 可识别的 MCP Server 配置。
- **白屏化操作，维护便利：**生成的配置可以导入 Higress 控制台，通过图形化界面进行修改和管理，极大降低了维护成本。
- **无需提供 MCP 运行时，运维便利：**Higress 自身承担了 MCP Server 的运行时角色，负责协议转换和请求路由，业务团队无需关心 MCP Server 的部署和运维。

借助 Higress 这一特性，业务可以将重心放在 MCP 工具的描述与 Agent 如何更好地协作上，而不是如何编写 MCP Server 的代码实现上，从而帮助业务智能化进程提效。具体可参考以下示例进行安装和使用：

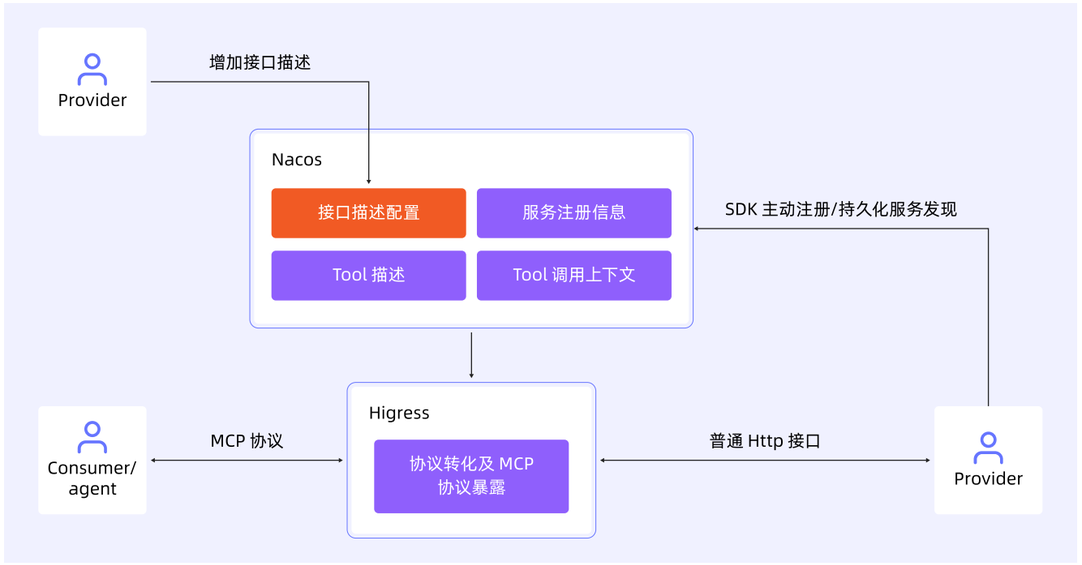
openapi-to-mcp的安装和使用 Shell

```shell
# 安装工具
go install github.com/higress-group/openapi-to-mcpserver/cmd/openapi-to-mcp@latest

# 将openapi.json转换为mcp-config.yaml
openapi-to-mcp --input openapi.json --output mcp-config.yaml --server-name openapi-server
```

将生成的 mcp-config.yaml 文件导入 Higress，即可完成路由配置。此时，Higress 即优雅地实现了将AI Agent的调用请求转发至后端 OpenAPI 服务。
此外，Higress 还支持将数据库（MySQL、PostgreSQL、Clickhouse 等）直接暴露为 MCP Server，只需提供数据库连接信息（用户名、密码、域名/IP、端口），即可自动生成用于数据查询和操作的 Tool，进一步拓宽了数据智能化的边界。

- **结合 Nacos 提供 MCP Registry 能力**

当 MCP Server 数量增多时，如何统一管理和动态更新这些 Tool 的元信息成为新的挑战。Nacos 提供了 MCP Registry 的能力，可以帮助更好地集合 MCP 信息和管理 MCP Server 运行时。同样，Nacos 也支持将存量 API 服务转换为 MCP 可调用的工具服务，其直接通过Nacos 配置进行实现，而无需对任何业务代码进行修改，并可同时结合 Higress 实现 MCP 协议和存量协议的转换。通过将 Higress 与 Nacos 服务注册与发现中心结合，可以构建一个更为强大和灵活的 MCP 服务治理体系。其具体调用流程图如下所示：



在智能化改造过程中，使用 Nacos MCP Registry 的优势在于：

- **存量 API 快速构建 MCP Server：**通过Nacos和Higress集成，用户可以零代码快速构建 MCP Server，迅速跟进MCP协议，无缝对接存量 API；
- **统一注册与动态发现：**所有 MCP Server（包括存量 API 转换、自研、第三方的）均可以注册到 Nacos 进行统一管理；
- **MCP 信息动态下发实时生效：**Nacos 可以帮助管理和下发 MCP 信息、Tools 及 Prompt，实现动态调整和实时生效，以达到更好的效果；
- **MCP 版本与灰度管理：**Nacos 天然支持配置的历史版本管理和灰度发布。当需要调整 Tool 描述时，Nacos 支持灰度分批生效，可以对比不同版本描述对 AI 调用效果的影响，确保变更平稳、可控，并可在出现问题时快速回滚；
- **MCP 服务管理及健康检查：**Nacos 在 MCP 服务数量的增加下提供大规模服务管理能力，包括健康检查、实时更新和负载均衡，确保 MCP 服务的高效运行。

3、MCP Server 优化与扩展

构建一个可工作的 MCP Server 只是第一步，要使其在生产环境中稳定、高效、安全地运行，还需进行精细的优化与扩展。

- **数据处理优化：**
  - 数据聚合与精简：并非所有API返回的字段都是 LLM 所需要的，可以考虑在 MCP Server

端进行数据聚合和精简，只返回 LLM 必需字段，减少传输的数据量，降低 LLM 处理上下文的 Token 消耗，从而直接节约成本并提升响应速度。

- 数据结构转换：某些 LLM 对深层嵌套的 JSON 结构处理能力较弱，可以通过扁平化或重构复杂嵌套、对复杂参数分拆为多次单层调用后在服务端聚合等方式调整 API 返回的数据结构，使其更易于 LLM 理解和处理。

- **性能与可靠性：**
  - 缓存与异步：可以采用缓存机制与异步处理等技术手段进一步提高数据处理和传输的性能，例如为慢接口/稳定数据加入缓存；对于耗时较长的操作，采用异步处理模式，避免阻塞。
  - 连接管理：数据库连接池、HTTP 客户端连接池等技术，提高资源利用率；设置合理的超时、重试、退避与熔断。
  - 可观测性：集成 Prometheus、Grafana、OpenTelemetry 等监控工具，对 MCP Server 的网络延迟、错误率、服务器负载、Tool 调用频率、QPS 等关键指标进行监控，以便定位性能瓶颈并定义相应告警。

- **安全与合规：**
  - 身份认证与授权：使用 API 密钥、OAuth 或临时令牌等方式对客户端进行身份验证，对 Tool 的调用进行细粒度的权限控制，并对传入参数进行严格校验，防止任意URL访问或参数注入等安全风险。
  - 日志记录：详尽的日志对于调试和审计至关重要。记录每次 Tool 调用的请求、参数、响应及耗时，有助于追踪问题和分析使用模式。
  - 持续升级策略：MCP 协议和相关 SDK 仍在快速演进。应制定明确的升级策略，定期跟进官方规范和依赖库的更新，以获取新功能和安全补丁。

- **功能扩展：**
  - 基础功能完善后，可以进一步扩展 MCP Server 的能力，例如：集成数据库进行复杂查询、封装文件系统操作、调用外部 SaaS 服务的 API（如 GitHub、阿里云服务）、支持 Webhook 以实现事件驱动的交互、提供流式资源等，不断丰富 AI 的工具箱。

5.3.2 从第一个 MCP Server 开始

借助 MCP，企业可以将散落在各个角落的传统数字资源快速、标准化地暴露为 LLM 可安全使用的工具。从0到1的关键在于选择熟悉的 SDK、用最小面工具打通首个业务场景、以中间层适配 OpenAPI/数据库降低接入成本，并在生产阶段补齐安全、性能与治理。

MCP Server 的开发之旅，从技术上讲并不复杂，但要构建一个真正生产可用、性能卓越、智能高效的服务器，则需要在协议理解、业务封装、性能优化和安全防护等多个维度上精益求精。建议以“一个小而确定的用例”为起点，充分利用开源能力（如 Higress、Nacos、各语言 SDK）开始动手实践，在此基础上逐步优化、迭代和扩展到更复杂、更高价值的业务流程。

亲眼见证这些经过智能化改造的数字资源，如何为 AI 原生应用注入前所未有的活力与智慧，这本身就是一场激动人心的变革。

AI 网关

AI Gateway

CHAPTER

05

AI Tools

P133-P156

06

网关的演进历程

AI 网关的定义、特点与应用场景

AI 网关的核心能力和最佳实践

使用 AI 网关快速构建 AI 应用

API 和 Agent 的货币化

P159-P202

07

AI Runtime

P205-P224

08

AI Observability

P227-P256

6.1 网关的演进历程

网关一词相信大家都不陌生，还记得最早的网关是什么样子吗？没错，就是一个简单的反向代理。那时候它的工作很单纯：把客户端的请求转发到后端服务器，再把响应返回给客户端。这就像一个单纯的传话筒，听到什么说什么。随着互联网的普及与企业网络的扩展，网关的作用也愈发重要，它不仅承担着数据传输的桥梁，还逐渐加入了安全、负载均衡、流量控制、服务治理等多种功能。同时从网关的演进形态也能一窥软件架构的发展史，所谓业务驱动技术迭代发展，技术反哺业务快速增长，珠联璧合。

读到这里你可能会有这样的疑问：既然都是网关为什么会有这么多的形态或者叫法呢？背后的原因简单总结是因为不同的应用场景会有不同的功能诉求，不同的功能特性需要有相应的特征描述词方便大家记忆与区分，例如，大家提到微服务网关就会很自然的联想到微服务，进而联想到微服务的服务注册、发现与治理。

在了解了软件架构演进过程中的各网关形态后，我们不禁要自问一下：是什么在推动软件架构的演进呢？相信读者心中已经有了答案：业务规模。业务从互联网、移动互联网以及即将到来的万物互联，其规模仍然在高速增长，也就意味着软件架构的复杂度也会越来越高，网关的特性也会越来越多、越来越复杂、越来越智能。接下来，我们从流量网关开始，逐一介绍不同形态网关的核心能力。

网关演进形态概览

伴随软件架构的演进网关形态也在持续进化，K8s 成为统一运维界面，AI 成为流量增长的核心驱动力

6.1.1 流量网关

流量网关作为网络架构中的关键组件，主要负责管理和优化数据流量，以提升业务的可伸缩性和高可用性。 Nginx 作为流量网关的代表性软件，以其高效的性能和灵活的配置广受欢迎。流量网关的核心目的是解决多业务节点间的流量负载均衡问题，通过将客户请求分配到不同的服务器上，从而均匀分摊负载，避免单点故障，确保服务的稳定性和连续性。流量网关在单体架构与垂直架构中被广泛应用。

6.1.2 ESB 网关

企业服务总线（ESB）是一个集中式的业务网关，旨在标准化和简化不同系统和服务之间的通信与消息传送。作为核心通信基础设施，ESB 网关可以减少系统间的耦合性，提高互操作性和灵活性，确保数据和服务的无缝整合。遵循服务导向型架构（SOA）原则，ESB 通过集中管理消息路由、转换和安全，实现服务的快速部署和高效运作。它支持不同协议和数据格式，提升了系统的扩展性和可维护性。ESB 是 SOA 架构中的核心网络组件。

6.1.3 微服务网关

微服务网关是微服务架构中的关键核心组件，负责集中管理微服务的路由规则，增强系统安全性，简化访问流程，从而提高整个系统的可靠性。微服务网关可以实现负载均衡、限流、熔断、降级、身份验证等功能，通过统一入口管理和优化各微服务间的交互。此举不仅简化了客户端与微服务的通信复杂性，还为系统安全提供了额外的保护，Spring Cloud Gateway 是一个广泛应用的微服务网关，它基于 Spring 生态系统，易于与 Spring Boot 项目集成，因其灵活、高效和可扩展性受到了开发者的青睐。

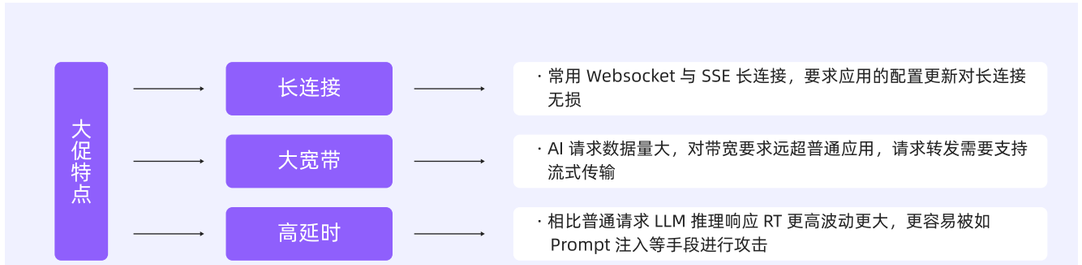
6.1.4 云原生网关

云原生网关是伴随 K8s 的广泛应用而诞生的一种创新网关，K8s 集群内外网络天然隔离的特性要求通过网关来将外部请求转发给集群内部服务，K8s 采用 Ingress/Gateway API 来统一网关的配置方式，同时 K8s 提供了弹性扩缩容来帮助用户解决应用容量调度问题，基于此用户对网关产生了新的诉求：期望网关既能有流量网关的特性来处理海量请求，又具备微服务网关的特

性来做服务发现与服务治理，同时要求网关也具备弹性扩缩容能力解决容量调度问题，能够让开发者能够专注于业务逻辑的实现，而无需担心底层架构的容量、维护和管理。

6.1.5 AI 网关

之前的几种类型网关处理的流量基本是以 HTTP、RPC 为主，多采用长连接提高传输性能，长连接采用 request-response 模式，目前在 AI 场景中有了新的变化，AI 采用的协议以 SSE/Websocket 为主，这两种协议虽然也是长连接，但相比之前的 request-response 模式，Server 可以主动发送数据给 Client，Websocket 更是一种全双工协议，主要在如语音类的实时通讯场景中使用。这种长连接对应用带来的一个巨大的潜在影响是，原来的无状态应用会变成有状态应用，应用层的配置变更不能中断长连接的传输，AI 流量特点简图如下：



在 AI 场景中，不仅流量有新的特点，而且对于网关也产生了新的诉求，最典型的是多模型代理诉求，背后的原因有以下4点：

- 企业需同时处理文本、图像、音频、3D 等多模态数据。研发、产品团队对推理能力强的模型需求多；客服、营销、平面设计等团队对图片大模型的场景需求多；工业设计、影视制作团队对音视频大模型的场景需求多。
- 企业业务覆盖多个垂直领域，需针对不同行业特性调用专用模型。尤其是供应链端的企业往往服务多个行业，可能会涉及多款垂直行业的大模型需求。
- 复杂任务协同场景，单一任务需要多个模型分工协作，以提升效果。多个大模型员工协同生成内容，才能起到最佳效果。
- 安全与效率双重要求场景，例如医疗机构的场景，处理患者数据使用专属私有模型分析，其他和患者无关的需求使用通用模型，避免敏感数据和非敏感数据在写入数据库混存。

除了多模型代理以外还有很多其他的诉求，例如智能路由、模型增强、安全防护、流式传输、无损变更等，由于篇幅有限这里就不再一一展开。目前对网关有一些更细粒度的分法，比如直接对接后端模型的称为 Model Gateway，面向终端应用的称为 AI Gateway，笔者认为其本质都是属于 AI 网关范畴。

读到这里你可能会问：这些 AI 网关的诉求是怎么来的呢？是从真实的实践场景中总结过来的还是自行 YY 的呢？所谓实践出真知，目前 Higress 在阿里云内部的 AI 场景中已经大规模生产落地，支持的核心业务见下图。

Higress 网关不仅在阿里云内部大规模生产落地，有对应的云产品，名称是 API 网关，也对外进行了开源。如果大家感兴趣可以访问项目地址：https://higress.ai/，了解更多内容。

Higress 的愿景是成为全球领先的 AI 网关，目前 Higress 已经启动了开源出海项目，也欢迎感兴趣的小伙伴一起参与共建。

6.1.6 API 网关

从流量网关，再到 AI 网关，都是 API 网关在不同软件架构下的网关形态。

在流量网关中，路由本身是一种 API，只是没有定义规范的请求和响应标准，通常被称为HTTP API，他是使用最简单、应用最广的 API。REST API 采用 OpenAPI 格式来规范请求和响应，在微服务架构下中对外提供服务时，应用较广。gRPC 和 Dubbo 都是高性能的远程调用框架，对于服务之间的高频调用，对带宽和 CPU 序列化开销大的场景，更加适用。

Websocket HTML5 标准中的全双工通信协议，在即时通讯应用的消息同步、游戏场景下的状态更新等实时场景下，更加适用。可以说支持 API 访问的都是 API 网关，API 网关是贯穿软件架构演进的各个阶段。

因此，唯一不变的是变化，在现代复杂的商业环境中，企业的业务形态与规模往往处于不断变化和扩大之中。这种动态发展对企业的信息系统提出了更高的要求，特别是在软件架构方面。为了应对不断变化的市场需求和业务扩展，软件架构必须进行相应的演进和优化。网关作为互联网流量的入口，其形态也在跟随软件架构持续演进迭代中。在 AI 应用爆发的当下，AI 网关正扮演着 AI 应用基础设施的角色。

6.2 AI 网关的定义、特点与应用场景

AI 网关是提供多模型流量调度，MCP 和 Agent 管理，智能路由和 AI 治理的下一代网关。

本质上，它是一种针对 AI 应用场景进行专门优化和能力扩展的 API 网关。AI 网关不仅完整继承了 API 网关的通用能力，如安全认证、路由转发、流量控制等，也演进出了大模型 Fallback、大模型负载均衡、Token 级别的精细化流量管控、语义化缓存、内容安全、联网搜索、MCP 协议转化和管理、工具精选和搜索效果优化等面向 AI 场景的能力。

6.2.1 AI 应用的流量特征

当我们尝试理解 AI 网关时，不妨先从 AI 应用的流量特征开始，因为 AI 网关的服务对象就是 AI 应用，所谓知其然而知其所以然。AI 应用的流量特征与我们所熟知的传统 Web 应用截然不同，主要体现在以下四个方面：

- **高延时**：模型推理涉及庞大的计算量，导致其响应时间远高于普通应用，从几十毫秒的量级跃升至数秒甚至数十秒。这种长耗时的特性，使得服务在面对类似慢速连接的恶意攻击时，变得异常脆弱，攻击者可以用极低的成本耗尽宝贵的后端计算资源。

- **大带宽与流式传输**：为了提升交互的即时感，AI 生成的较长内容（如文章、代码）通常不会等待全部生成完毕后再一次性返回，而是采用流式（Streaming）的方式，像打字机一样逐字或逐词地推送给用户。这对网关的流式处理能力和内存控制提出了极高的要求，传统的缓存转发模式很容易导致内存溢出。

- **长连接**：为了支持上下文连贯的多轮对话，AI 应用广泛采用 SSE (Server-Sent Events)或 WebSocket 等长连接协议。然而，许多传统网关在更新配置（如发布一条新路由）时，需要重启工作进程，这会强制切断所有已建立的长连接，导致用户的对话进程中断，仿佛正在交流的机器人突然失忆，体验极差。

- **API 驱动**：未来的 AI 原生应用将由无数个轻量化的 Agent 构成，它们通过调用各种原子化的 API（即工具）来完成查询信息、操作软件等复杂任务。这将导致应用内部和外部的 API 数量及调用量呈爆炸式增长，对 API 的设计、发布、监控、安全等全生命周期管理能力提出了前所未有的要求。

6.2.2 AI 网关的应用场景

根据不同的业务需求和部署位置，AI 网关通常出现在以下四种典型的场景中，扮演着各自独特的角色：

1. 模型服务提供商（MaaS）的接入层

对于提供基础大模型服务的厂商而言，AI 网关是其商业化服务的第一道防线。它部署在所有模型推理集群之前，作为流量的总入口，负责处理来自全球开发者和企业的海量请求。在此场景下，网关的核心职责是保障服务的性能、稳定和安全。例如，通过高效的负载均衡策略将请求智能分发给后端最合适的计算资源，通过精细化的多维度流量控制来防止服务被恶意流量或突发洪峰打垮，并为不同等级的客户提供差异化的服务质量（QoS）保障。

2. AI 应用的开发网关

对于广大的 AI 应用开发者（尤其是 SaaS 应用开发者）而言，他们通常不会自建模型，而是灵活地集成多家模型厂商的 API，以取长补短。AI 网关此时扮演着多模型智能调度中心的角色。它可以屏蔽不同厂商 API 的协议差异，提供一个统一、简洁的调用接口。更重要的是，当某个主用模型（如某个经过精调的垂直领域模型）响应缓慢或调用失败时，网关可以依据预设策略，自动重试或无缝切换到备用模型（Fallback，如某个通用的基础大模型），从而在不影响用户体验的前提下，极大地提升应用的稳定性和韧性。同时，它还能对所有模型的返回内容进行统一的安全合规过滤。

3. 企业内部的中央 AI 网关

在大型企业内部，当各个业务线都开始引入 AI 能力时，若各自为政，很快就会陷入前面提到的接入混乱、成本失控和安全风险的困境。此时，AI 网关就像一个面向 AI 时代的企业服务总线（ESB），成为所有内部系统访问内外部 AI 服务的统一、标准化的入口。这种集中化的模式带来了巨大的管理效益：

- **成本审计与分摊**：精确记录并分析各业务部门、各应用的 Token 消耗量，为成本控制、预算分配和内部结算提供清晰、可信的数据支持。

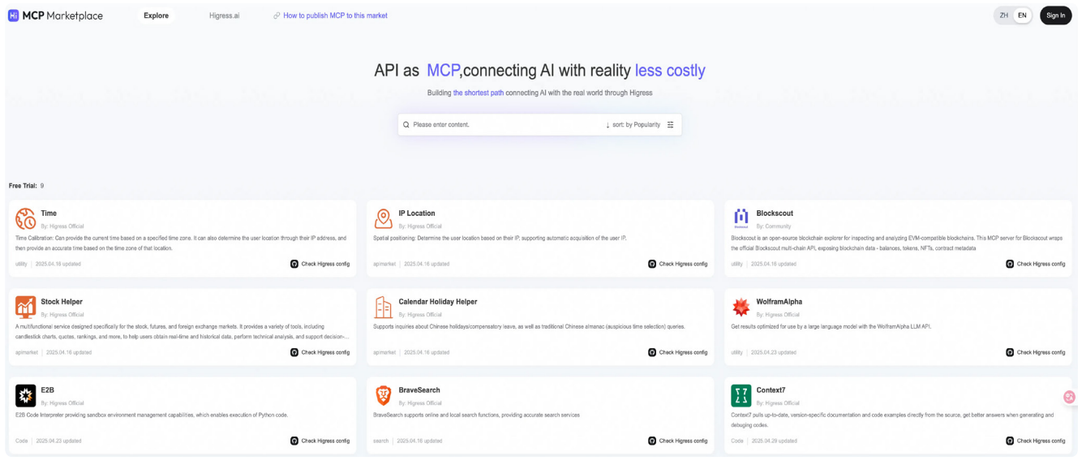
- **数据安全与合规**：建立全企业统一的内容安全策略，能够有效防止企业内部的敏感数据（如客户信息、财务报表）通过 prompt 被无意或恶意地发送给外部模型，构筑起一道关键的数据防泄露屏障。

- **资源复用与效率提升：**通过部署统一的语义化缓存，对全公司范围内的高频、相似问题直接返回缓存结果，不仅能有效降低模型调用成本，更能显著提升响应速度，改善员工和客户的体验。

4. MCP 工具生态的统一入口

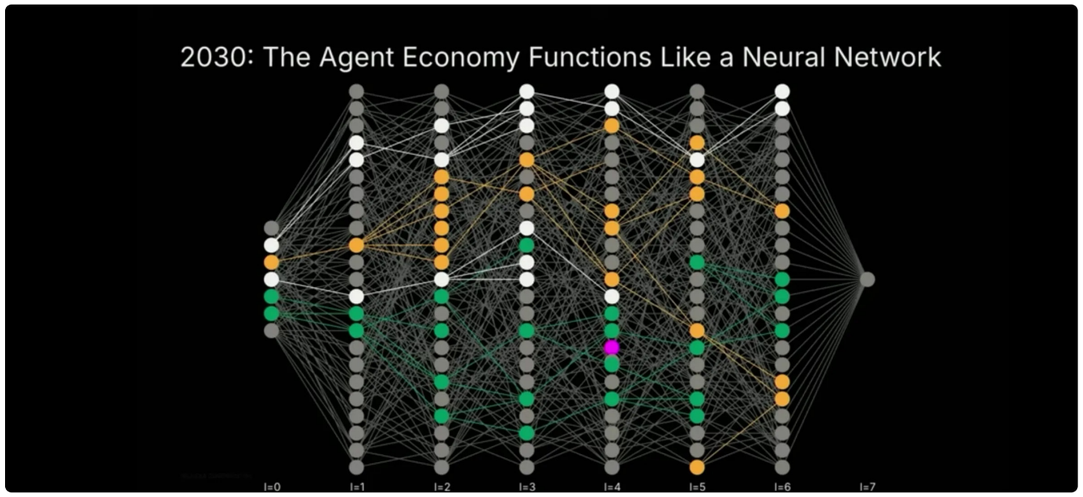
随着 AI Agent 的兴起，让 AI 能够调用外部工具（Tools）来完成查询订单、预订会议室等实际任务变得至关重要。AI 网关在此场景下，可以作为所有 MCP（Model Context Protocol）工具的统一收口和安全堡垒。所有对外部工具的调用请求都必须先经过 AI 网关，由网关进行集中的、标准化的安全管控，包括统一的认证鉴权、精细的速率限制、全面的审计日志等。这种方式避免了在每一个独立的工具服务上重复实现和维护复杂的安全逻辑，极大地简化了 MCP 工具生态的安全治理，为企业构建一个安全、可靠、可控的 AI Agent 体系提供了坚实的基础。例如 Higress 构建的开源 MCP 市场就是这样的一个样板间。

https://mcp.higress.ai/



5. 构建企业 AI 能力货币化的统一开放平台

任何企业的终极目标都是营收与利润，未来 AI 智能体应用会成为大家日常工作生活当中不可或缺的存在，在红杉风险投资举办的一场创业企业分享会中，就对未来 AI Agent 的前景做了展望，甚至红杉风投将其称之为 Agent 经济，并给了一张畅想图，从下面的图中大家可以感受到未来 Agent 货币化的广阔前景，那么 Agent 货币化的载体就是“AI 开放平台”，而这也是网关自身承载且擅长的领域。



图片来源：https://www.sequoiacap.com/article/ai-ascent-2025/

读到这里，你可能会产生这样的困惑，开放平台不是“API 网关”的职责吗？在上一个章节中简单说了下“API网关”，总结起来，API 是一种很泛化的定义。

Websocket/gRPC/Dubbo/HTTP 都可以被称之为 API，MCP、A2A 也可以被称之为 API。

ALIBABA CLOUD

6.3 AI 网关的核心能力和最佳实践

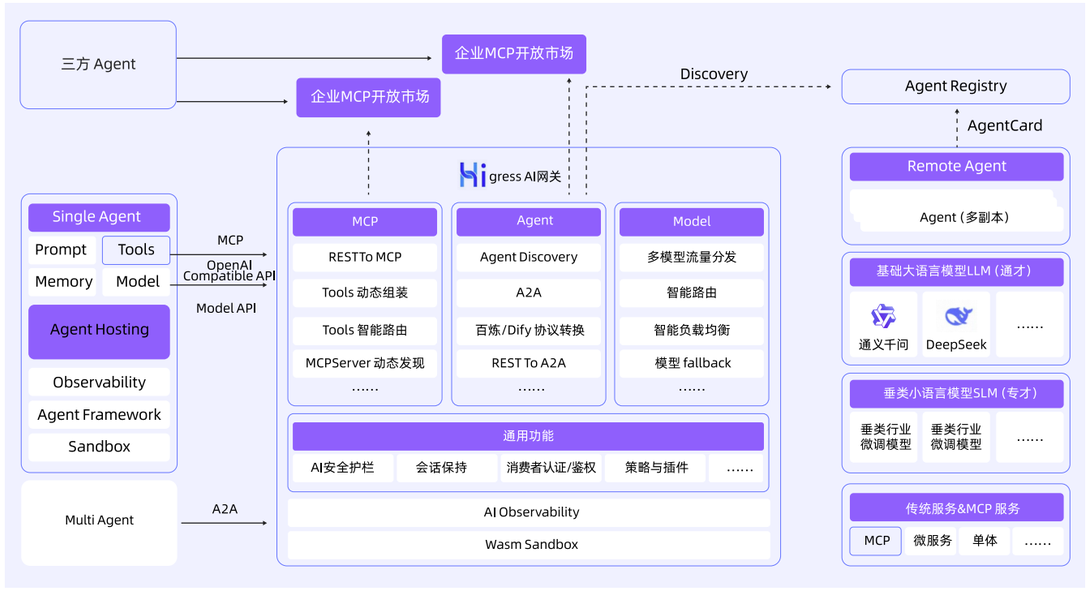
当前，大语言模型（LLM）正经历一个前所未有的百花齐放时代。从阿里云的 Qwen 系列、OpenAI 的 GPT 系列、Google 的 Gemini、Anthropic 的 Claude，再到 Llama 等众多开源模型，整个行业呈现出群雄逐鹿、蓬勃发展的多样性格局。

这种多样性不仅仅是模型的数量增多，更体现在能力上的分化与专精。有的模型以其强大的逻辑推理和代码生成能力著称，有的则在长文本处理和人文创意写作上表现优异，还有一些轻量级模型在特定任务上具备极高的性价比和响应速度。

在这样的背景下，Agent 的设计理念发生了深刻的演变。早期的 Agent 或许只依赖单一的强大模型来完成所有任务，但如今，为了实现更高效、更经济、更强大的功能，先进的 Agent 会根据任务类型的不同，在后台智能地调度和使用多个不同的模型。例如，一个复杂的报告生成任务可能会被拆解：用一个模型进行数据检索，调用另一个模型进行逻辑分析，最后再使用一个文笔优美的模型来润色文稿。

6.3.1 核心能力

当 Agent 从 Single 模式走向 Sub Agent、甚至 Multi Agent 时，AI 网关作为入口中间件，正发挥着更关键的应用基础设施作用：



1. 多模型代理：AI 网关是流量的统一入口，用来接收用户端请求，并负载均衡到后端模型，还能实现同一个接口对接多种模型，这个代理层不仅解决了调用不同模型 API 的复杂性，还通过动态路由实现了成本与性能的最佳平衡，更是提升了模型服务的可用性。

2. 多模型回退/容灾：依赖单一模型存在单点故障风险，API 的不稳定或性能下降都可能导致服务中断。AI 网关能同时对接多个模型，当单个模型出现调用失败、超时或返回质量不佳的结果时，可以自动 Fallover 到备选模型多模型，以确保服务的连续性和高可用性。

3. 消费者认证：当一个代理服务需要为多个用户或应用提供支持时，身份认证变得至关重要。通过 AI 网关清晰地识别每一个请求的来源，以便进行后续的计费、权限管理和个性化服务，确保按权限分类提供服务。

4. 内容安全防护：不同的模型拥有各自的安全策略，标准不一。在一个多模型系统中，必须建立一个统一、前置的内容安全防护层。AI 网关能通过内容安全插件，在请求送达模型之前和模型返回结果之后，对内容进行审查，过滤有害信息，确保整个应用输出的内容始终符合安全规范和合规性。

5. Token 限流：大模型调用按 Token 计费，且单位成本远高于 CPU 计费，因此有效控制成本是高优先级需求。AI 网关提供的 Token 限流机制可以从控制单个用户的调用频率和总量，以及对服务总流量进行调控两个维度进行管理，防止滥用导致费用激增，并保障服务的稳定性。

6. 语义缓存：AI 网关提供了扩展点，可接入 Redis 实现内容缓存。一能提高效率，如果相同的输入反复出现，缓存可以避免重复运行模型，从而加快响应速度，特别是在处理常见问题时。二是降低成本，大模型 API 计费因是否命中缓存，而有所不同，缓存机制可以减少模型调用次数，以节省计算资源。三是保持一致性，缓存可以确保相同输入产生相同输出，有助于测试和合规性场景。

7. 可观测性：在一个由多个开发平台、多个模型、多个组件构成的复杂系统中，统一的可观测性是运维和优化的基石。AI 网关能提供包括对每一次调用的详细日志记录（请求内容、选择的模型、响应结果、耗时）、关键性能指标（如延迟、Token 消耗、错误率）的监控，以及端到端的链路追踪。通过可观测性，企业可以快速定位问题、分析成本构成、洞察用户行为，并为模型的选择策略提供数据驱动的优化依据。

8. MCP 代理：面向 MCP Server，提供 MCP Server 代理、安全认证，以及统一观测、限流等治理能力，同时支持将 REST API 直接转化成 MCP Server，提供协议卸载能力，将 SSE 转换为 Streamable HTTP，避免无状态应用也要使用 SSE。

9. 工具的动态组装：当请求携带大量工具通过 AI 网关时，通过 Query 改写及Rerank 模型将大量工具进行压缩，再转发给 LLM，可大幅降低调用耗时，并在一定程度上增加工具选取的准确性。

10. 工具的智能路由：将用户注册在 AI 网关的大量 MCP Server、工具进行集合，以 MCP 或其他形态提供语义搜索能力，客户端只需要集成这个工具即可基于用户 Query，动态搜索出最符合需求的 N 个工具。

综上所述，不管是构建企业级的 Single Agent 亦或是 Multi Agent，多模型代理、消费者认证、多模型 Fallover、可观测等都已经成为 AI 应用的通用诉求，根据以往的软件演进经验，这些通用诉求的最佳载体就是网关，正如微服务网关统一解决了微服务的外部认证鉴权、服务发现、微服务容灾等，AI 网关也是帮助企业快速构建 AI 应用的最佳实践。

6.3.2 最佳实践

AI 网关经过持续的技术演进，其原子能力已经非常丰富，可以总结为面向 LLM、Agent、MCP 和 AI 开放平台四大类。通过服务开源社区用户和云上商业客户的 AI 网关落地，我们总结了8类常见的实践，多模型代理、消费者认证、内容安全防护、Token 限流、语义缓存、多模型容灾、多模型可观测和 AI 开放平台，看看这些实践是如何借助 AI 网关的能力去满足实际的业务需求的。

AI 网关经过持续的技术演进，其原子能力已经非常丰富，可以总结为面向 LLM、Agent、MCP 和 AI 开放平台四大类。通过服务开源社区用户和云上商业客户的 AI 网关落地，我们总结了8类常见的实践，多模型代理、消费者认证、内容安全防护、Token 限流、语义缓存、多模型容灾、多模型可观测和 AI 开放平台，看看这些实践是如何借助 AI 网关的能力去满足实际的业务需求的。

1. 多模型代理

需求场景

- 多模态业务整合场景，企业需同时处理文本、图像、音频、3D 等多模态数据。研发、产品团队对推理能力强的模型需求多；客服、营销、平面设计等团队对图片大模型的场景需求多；工业设计、影视制作团队对音视频大模型的场景需求多。
- 企业业务覆盖多个垂直领域，需针对不同行业特性调用专用模型。尤其是供应链端的企业往往服务多个行业，可能会涉及多款垂直行业的大模型需求。
- 复杂任务协同场景，单一任务需多个模型分工协作以提升效果。多个大模型员工协同生成内容才能起到最佳效果。
- 安全与效率双重要求场景，例如医疗机构的场景，处理患者数据使用专属私有模型分析，其他和患者无关的需求使用通用模型，避免敏感数据和非敏感数据在写入数据库混存。

多模型代理的核心价值

- **业务需求适配：**根据业务复杂性或性能要求选择不同模型。
- **数据隐私与合规性：**在处理敏感数据时，可能需要切换到符合特定法规的模型，确保数据处理的安全性。
- **性能优化：**根据实时性能需求，可能会切换到更快的模型以减少延迟。
- **成本与性能平衡：**根据预算动态选择性价比最优的模型
- **领域特定需求：**针对特定领域（如法律、医学），可能需要切换到在相关领域微调过的模型，以提高推理准确性。
- **容灾与故障转移：**主模型服务异常时快速切换备用模型。

2. 消费者认证

需求场景

多租户模型服务分租场景：企业为不同部门或团队提供共享的大模型服务时，会通过 API Key 区分租户，确保数据隔离和权限管控。具体要求包括：

- 为每个租户分配独立 API Key，控制其调用权限和资源配额度，例如部门 A 的调用资源配额是每天每人20次，部门 B 的调用资源配额是每天每30次。
- 支持租户自定义模型参数（如温度系数、输出长度），但需通过网关校验权限。

企业内部权限分级管控：企业内部不同角色需差异化访问模型能力。具体要求包括：

- 基于 RBAC（基于角色的访问控制）限制敏感功能（如模型微调、数据导出）。
- 出于成本考虑，多模态大模型只供设计部门调用。
- 记录操作日志并关联用户身份，满足内部审计需求。例如，金融企业限制风险评估模型仅限风控部门调用，防止普通员工滥用。

消费者认证的核心价值

- **身份可信：**确保请求方为注册/授权用户或系统。
- **风险拦截：**防止恶意攻击、非法调用与资源滥用。
- **合规保障：**满足数据安全法规及企业审计要求。
- **成本控制：**基于鉴权实现精准计费与 API 配额管理。

3. 内容安全防护

需求场景

- **金融行业敏感数据处理：**审核用户输入的金融交易指令、投资咨询内容，防范欺诈、洗钱等违规行为。对模型生成的财务报告、风险评估结果进行合规性校验。
- **医疗健康信息交互：**电子病历生成内容，防止泄露患者隐私（如身份证号、诊断记录），确保 AI 生成的医疗建议符合相关法规。通过多模态大模型识别医疗影像中的敏感信息，并结合合规规则库进行自动化拦截。
- **社交媒体与 UGC 内容管理：**实时审核用户发布的图文、视频内容，拦截涉黄、暴恐、虚假信息。对 AI 生成的推荐内容（如短视频标题、评论）进行合规性检查。
- **政务服务平台交互：**审核公众提交的政务咨询内容，防止恶意攻击或敏感信息传播，确保 AI 生成的政策解读、办事指南符合相关法规。
- **电商与直播平台风控：**审核商品描述、直播弹幕内容，拦截虚假宣传、违禁品信息，对 AI 生成的营销文案进行广告法、合规性检查。

内容安全防护的核心价

- **防止攻击：**验证输入可以阻止恶意提示注入，防止模型生成有害内容。
- **维护模型完整性：**避免输入操纵模型，导致错误或偏见输出。
- **用户安全：**确保输出没有有害或误导性内容，保护用户免受不良影响。

- **内容适度：**过滤掉不适当的内容，如仇恨言论或不雅语言，特别是在公共应用中。
- **法律合规：**确保输出符合法律和伦理标准，尤其在医疗或金融领域。

4. Token 限流

需求场景

虽然企业内部使用，不会频繁存在并发的需求，但通过设置限流能力，可以更经济的配置硬件资源。例如一家10000人的企业，不需要配置同时支持10000人上线的硬件资源，只需要配置7000人的硬件资源，超出部分进行限流，避免资源闲置。其他需求包括：

- **提升资源管理：**大模型对计算资源的消耗不可控，限流可以防止系统过载，确保所有用户都能获得稳定性能，尤其在高峰期。
- **指定用户分层：**可以基于 ConsumerId 或者 API Key 进行 Token 限流。
- **防止恶意使用：**通过限制 Token 数量来减少垃圾请求或攻击，以免受到资损。

Token 限流的核心价值

- **成本管理：**LLM 的费用通常基于 Token 数量计算，限流帮助用户避免超支。例如，服务提供商可能按 Token 使用量提供不同定价层。
- **资源管理：**LLM 需要大量计算资源，限流防止系统过载，确保所有用户都能获得稳定性能，尤其在高峰期。
- **用户分层：**可以基于 ConsumerId 或者 API Key 进行 Token 限流。
- **防止恶意使用：**通过限制 Token 数量来减少垃圾请求或攻击。

5. 语义缓存

需求场景

多大模型 API 服务定价分为每百万输入 tokens X 元（缓存命中）/ Y 元（缓存未命中），X 远低于 Y，以通义系列为例，X 仅为 Y 的40%，通过在内存数据库中缓存大模型响应，并以网关插件的形式来改善推理的延迟和成本。在网关层自动缓存对应用户的历史对话，在后续对话中自动填充到上下文，从而实现大模型对上下文语义的理解。例如：

- **高频重复性查询场景：**客服系统、智能助手等场景中，用户常提出重复问题（如“如何重置密码”“退款流程”），通过缓存常见问题的回答，避免重复调用模型，降低调用成本。
- **固定上下文多次调用场景：**法律文件分析（如合同条款解读）、教育教材解析（如知识点问答）等场景，需对同一长文本多次提问。通过缓存上下文，避免重复传输和处理冗余数据，提升响应速度，降低调用成本。

- 复杂计算结果复用场景：数据分析与生成场景（如财报摘要、科研报告生成），对相同数据集的多次分析结果缓存，避免重复计算。
- RAG（检索增强生成）场景中：缓存知识库检索结果（如企业内部 FAQ），加速后续相似查询的响应。

语义缓存的核心价值

- **提高效率：**如果相同的输入反复出现，缓存可以避免重复运行模型，从而加快响应速度，特别是在处理常见问题时。
- **降低成本：**减少模型调用次数可以节省计算资源，尤其对大型模型来说成本较高。
- **保持一致性：**缓存确保相同输入产生相同输出，有助于测试和合规性场景。

6. 多模型容灾

需求场景

- **模型自身特性引发的异常：**大模型生成结果存在概率性波动，导致存在随机性输出不稳定的情况；发布新版本导致的流量有损。
- **用户使用不规范导致的异常：**使用者请求参数不符合 API 规范，导致连接超时或中断，或者输入包含恶意构造的提示词，触发模型安全防护机制，返回空结果或错误码。
- **资源与性能限制：**请求频次过高 ，触发限流策略，导致服务不可用，长请求占用过多内存，导致后续请求被阻塞，最终导致超时。
- **依赖服务故障 ：**外部 API，例如 RAG 检索的数据库不可用，导致模型无法获取必要上下文。

模型容灾的核心价值

当主 LLM 服务因为各种原因出现异常，不能提供服务时，网关侧可以快速将请求 Fallback到配置的其他 LLM 服务，虽然可能推理质量有所下降，但是保证了业务的持续性，争取了排查主 LLM 服务的时间。

7. 多模型可观测

需求场景

可观测常见于成本控制和稳定性场景。由于大模型应用的资源消耗比 Web 应用更加敏感和脆弱，因此成本控制对可观测的需求更为强烈，如果缺少完备的可观测能力，异常掉用可能会耗费几万甚至几十万的资损。除了：

模型可观测的价值

AI 网关通过支持在应用、网关、后端 LLM 服务上开启 OT 服务来进行全链路的跟踪，通过 TraceId 来串联各个地方的日志、请求参数等信息。除了提供 QPS、RT、错误率等传统观测指标，AI 网关还能集成更多元化的监控指标：

- 基于 Consumer 的 Token 消耗统计。
- 基于模型的 Token 消耗统计。
- 限流指标:每单位时间内有多少次请求因为限流被拦截，限流消费者统计（是哪些消费者在被限流）。
- 缓存命中情况。
- 安全统计：风险类型统计、风险消费者统计。

8. AI 开放平台

需求场景

当企业借助 Higress AI 网关完成模型与工具的统一接入后，难点便从“连通”转向“协作与产品化”。工具提供方、Agent 开发者、安全合规、财务运营等角色各自为政，产生了能力目录分散、上架流程不一致，权限与配额难统一，成本难计量与分摊，内容安全也难以落实到每一次调用等一系列问题。企业自然需要一个面向多角色、体验友好、功能完备的上层开放平台。

从 0 开始构建，可能需要：

- 开发一套完整的门户系统（> 3个月）：从 UI/UX 设计到前后端开发。
- 实现复杂的开发者与应用管理：注册、审批、RBAC 权限、凭证管理与轮转。
- 设计繁琐的订阅与授权流程：如何让开发者自助订阅？如何将授权关系安全地同步到网关？
- 构建面向运营和开发者的可观测性：从网关和 MCP Registry 拉取原始数据，进行二次
- 开发，实现按模型、按消费者的多维度成本与计量分析。

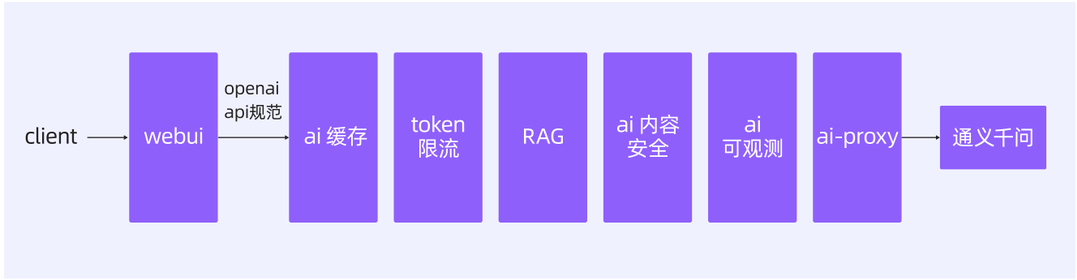
AI 开放平台的核心价值

- **AI 开放平台管理后台 (for 管理员/运营)：**在这里将底层的模型服务、MCP Server、Agent 等多样化的 AI 能力，以 API 的形式轻松打包成标准化的“AI 产品”，并配上完善的文档、示例，最终一键发布到门户。
- **AI 开放平台门户 (for 开发者)：**门户是面向内外开发者的“店面”。开发者可以在此完成开发者注册、创建消费者、获取凭证、浏览和订阅 AI 产品、在线测试，并清晰地监控自己的调用状态和成本。

- **AI 网关：**作为 Higress 社区的子项目，Higress AI 网关承载所有 AI 调用的认证、安全、流控、协议转换以及可观测性等能力。
- **Nacos：**Nacos 作为 MCP Registry 为门户提供全面的 MCP Server 元信息托管，版本管理，服务发现，密钥托管等能力。通过动态服务发现和动态配置变更实现系统灵活扩展和变更，通过 Nacos MCP Router 智能路由简化多 MCP 服务管理和调用。在日常开发测试的场景中，通过动态 Prompt 变更实现高效的 MCP 服务调试，在生产环境中通过多版本管理实现灰度发布，通过加密存储和动态配置变更提升安全能力。

6.4 使用 AI 网关快速构建 AI 应用

ChatGPT-Next-Web <https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web> 是一个开源的前端项目，用于提供大模型聊天窗口，支持接入多种大模型，本文基于 Higress、通义千问以及 ChatGPT-Next-Web，演示如何逐步搭建一个体系完整的 AI 应用，该应用的架构如图所示：



在构建 AI 应用的过程中会用到 Higress AI 网关提供的一些列插件能力，为了方便大家了解，这里再贴下 Higerss 的插件能力图谱，如下：

ALIBABA CLOUD

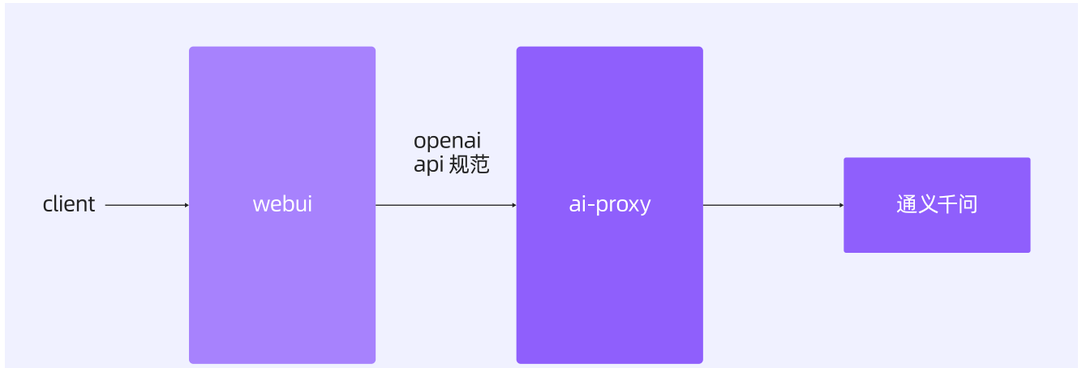
6.3.1 核心能力

AI 代理插件实现了基于 OpenAI API 契约的 AI 代理功能。Higress AI 代理插件目前支持的模型有：通义千问、DeepSeek、OpenAI、Anthropic Claude、Azure OpenAI、月之暗面、智谱 AI、百川智能、零一万物、Ollama、Groq。插件的详细描述见社区文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/aiproxy/README.md

应用架构

首先，我们先通过网关快速部署一个可以进行对话的聊天应用，其架构如下图所示：



LLM 服务使用通义千问，服务类型为 DNS。路由及服务创建完成后如下图所示：

服务名称	健康检查状态	服务地址	端口	服务来源	FQDN
chatgpt-next-web	健康	172.16.0.233:3000	-	固定地址	chatgpt-next-web.static
qwen	健康	dashscope.aliyuncs.com	443	DNS 域名	qwen.dns

路由名称 / 发布状态	规则
llm 已发布	http://*/api/openai/v1/chat/completions/* → qwen
chatgpt-next-web 已发布	http://*/* → chatgpt-next-web

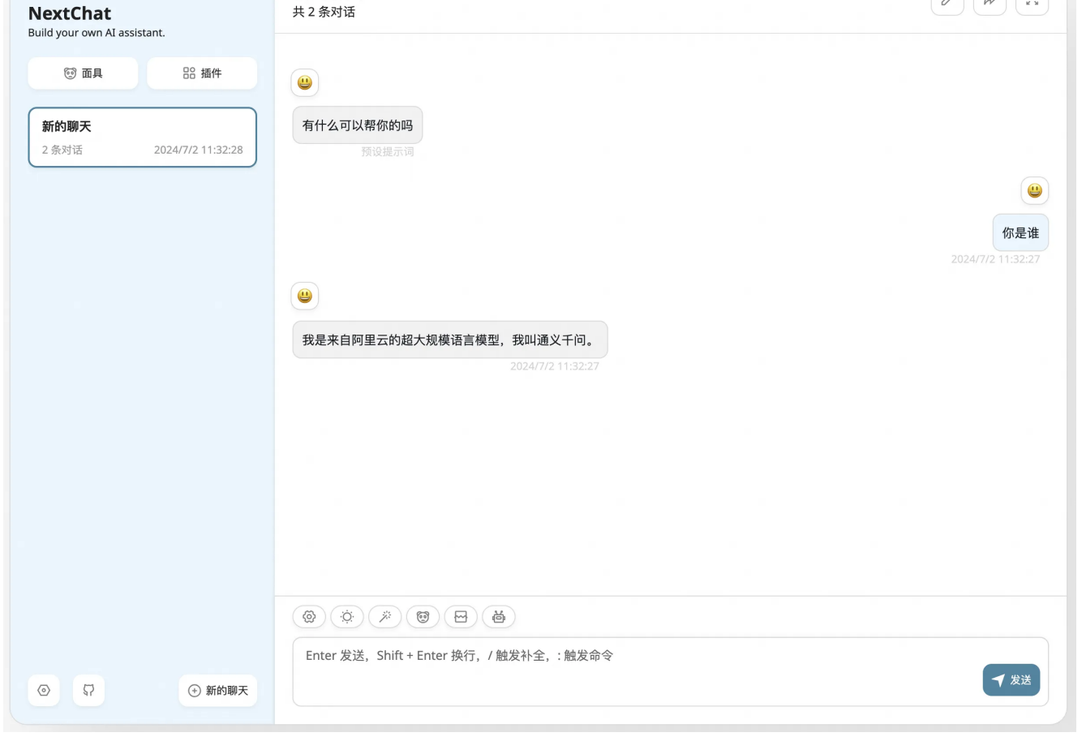
插件配置

置路由级插件规则，选择在 LLM 路由下生效，配置如下：

YAML | 复制代码

```yaml
provider:
  type: qwen
  apiTokens:
    - sk-xxxxxxxxxxxxxxxxxxxxxx
  timeout: 1200000
  modelMapping:
    'gpt-3.5-turbo': qwen-turbo
    'gpt-4': qwen-max
    '*': qwen-max
```

插件效果



6.4.2 AI 可观测插件

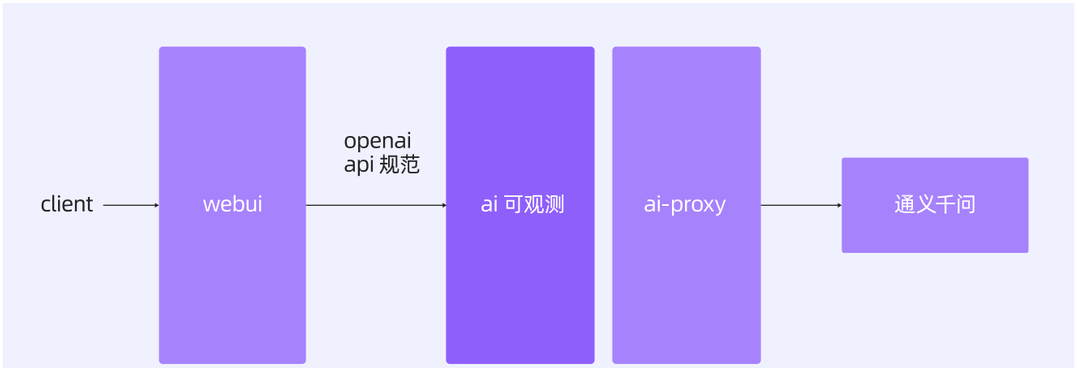
提供 AI 可观测基础能力，包括 Metric、Log 和 Trace，其后需接 ai-proxy 插件，如果不接 ai-proxy 插件的话，则需要用户进行相应配置才可生效。具体插件的详细描述见社区文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/aistatistics/README.md

应用架构

现在，我们已经有了基础的对话功能，作为一款网关产品，我们希望在网关这个统一的入口处对各个服务、路由的请求情况进行观测。考虑到 LLM 请求主要以 Token 为观测目标，网关提供了对 Token 的观测机制，包含路由级、服务级、模型级的 Token 用量观测。

现在，我们改变上文的应用架构，插入可观测插件，改造后如下图所示：



插件配置

依然是选择在 LLM 这条路由上生效，插件配置如下：

```yaml
enable: true
```


插件效果

6.4.3 AI 内容安全插件

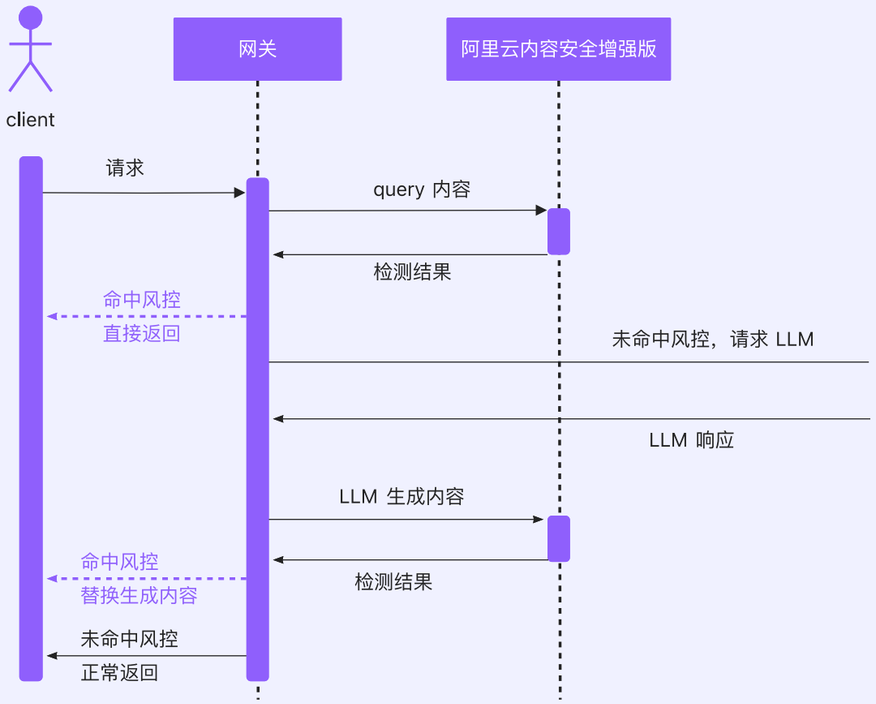
通过对接阿里云内容安全检测大模型的输入输出，保障 AI 应用内容合法合规。插件的详细描述见社区文档：

https://github.com/alibaba/higress/blob/main/plugins/wasmgo/extensions/ai-security-guard/README.md

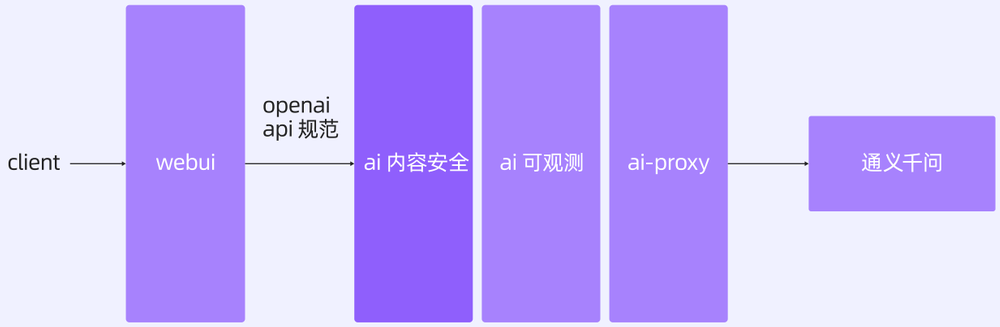
应用架构

大模型通常是通过学习互联网上广泛可用的数据来训练的，它们有可能在过程中学习到并复现有害内容或不良言论，因此，当大模型未经过适当的过滤和监控就生成回应时，它们可能产生包含有害语言、误导信息、歧视性言论甚至是违反法律法规的内容。正是因为这种潜在的风险，大模型中的内容安全就显得异常重要。

基于 AI 内容安全插件，通过简单的配置即可对接阿里云内容安全 <https://help.aliyun.com/document_detail/28417.html> ，为大模型问答的合规性保驾护航。

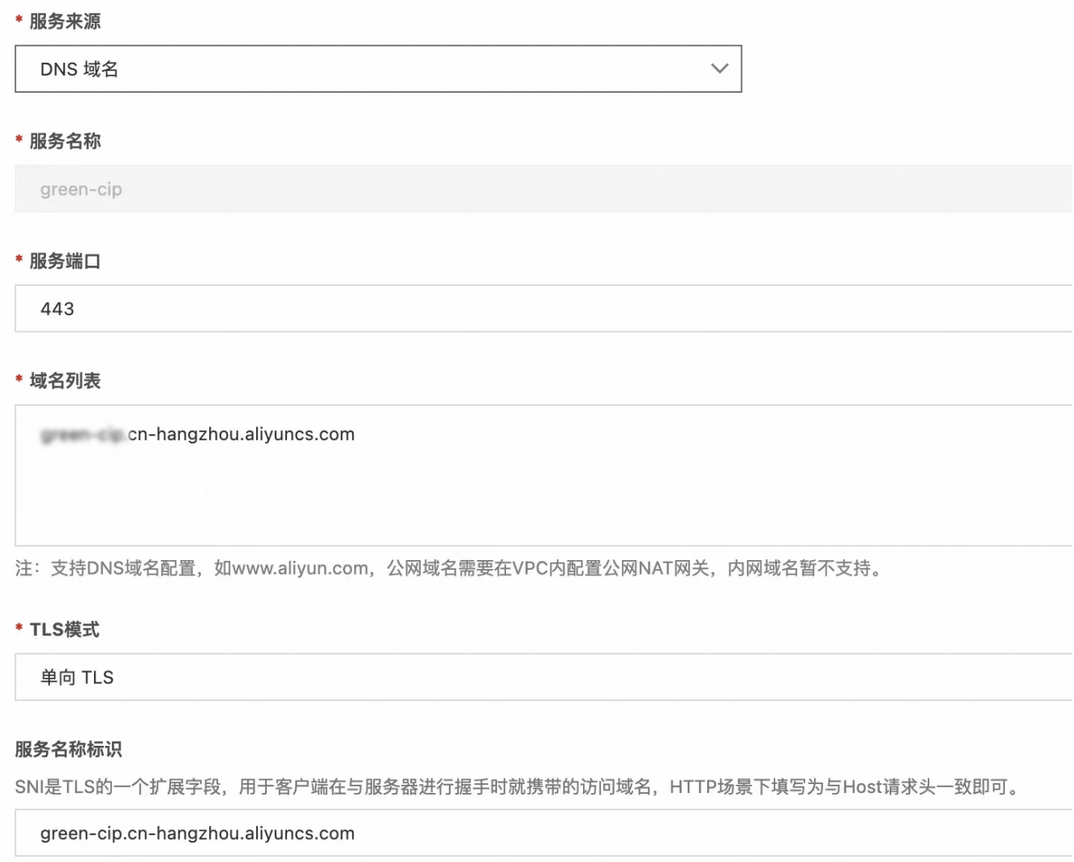


配置 AI 内容安全插件后，应用架构如下图所示：



插件配置

首先需要在网关配置内容安全的服务：

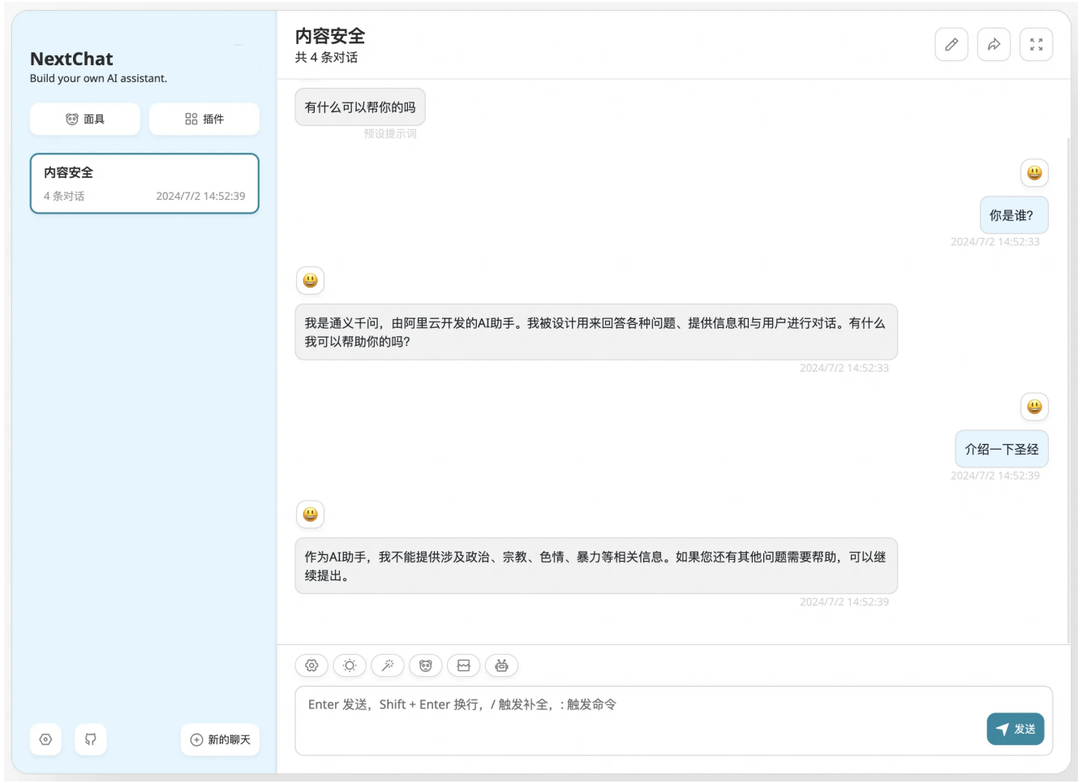


配置服务后，开启内容安全插件，选择对 LLM 路由生效：

YAML | 复制代码

```yaml
serviceSource: dns
serviceName: green-cip
servicePort: 443
domain: green-cip.cn-hangzhou.aliyuncs.com
ak: xxxxxxxxxxxxxxxx
sk: xxxxxxxxxxxxxxxx
```

插件效果



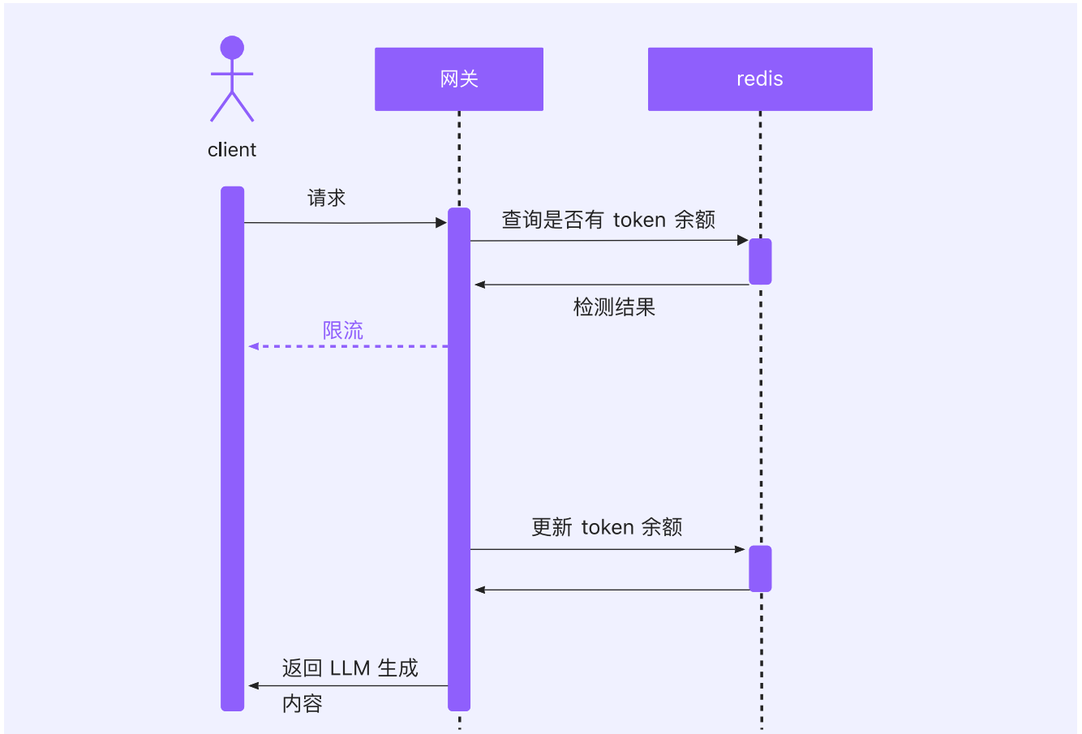
登录阿里云内容安全控制台，可以查看每条请求的审计记录：

6.4.4 AI Token 限流插件

插件实现了基于特定键值实现 Token 限流，键值来源可以是 URL 参数、HTTP 请求头、客户端 IP 地址、Consumer 名称、Cookie 中 key 名称。其借助 Redis 实现全局的 Token 限流。插件的详细描述见社区文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/ai-token-ratelimit/README.md

应用架构



创建一个 Redis 服务并且在网关进行配置：

服务名称	健康检查状态	服务地址	端口	服务来源	FQDN
redis	健康	172.16.0.233:6379	-	固定地址	redis.static

之后添加 AI Token 限流插件，应用架构为：

插件配置

YAML 复制代码

```yaml
rule_name: default_rule
rule_items:
  - limit_by_per_ip: from-remote-addr
    limit_keys:
      - key: 0.0.0.0/0
        token_per_minute: 100
redis:
  service_name: redis.static
  service_port: 6379
  username: xxxxxx
  password: xxxxxx
rejected_code: 429
rejected_msg: 您的请求频率过高，请稍后再试。
```

以上插件配置效果为每个 IP 地址每分钟内只能使用100个 Token，当超过 Token 限制时，返回 429，响应 body 为“您的请求频率过高，请稍后再试。”

插件效果

6.4.5 AI 缓存

AI 缓存插件能够缓存每个请求的响应，当有相同请求到来时，可以直接返回存储在 Redis 中的大模型的生成内容，社区文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/ai-cache/README.md

应用架构

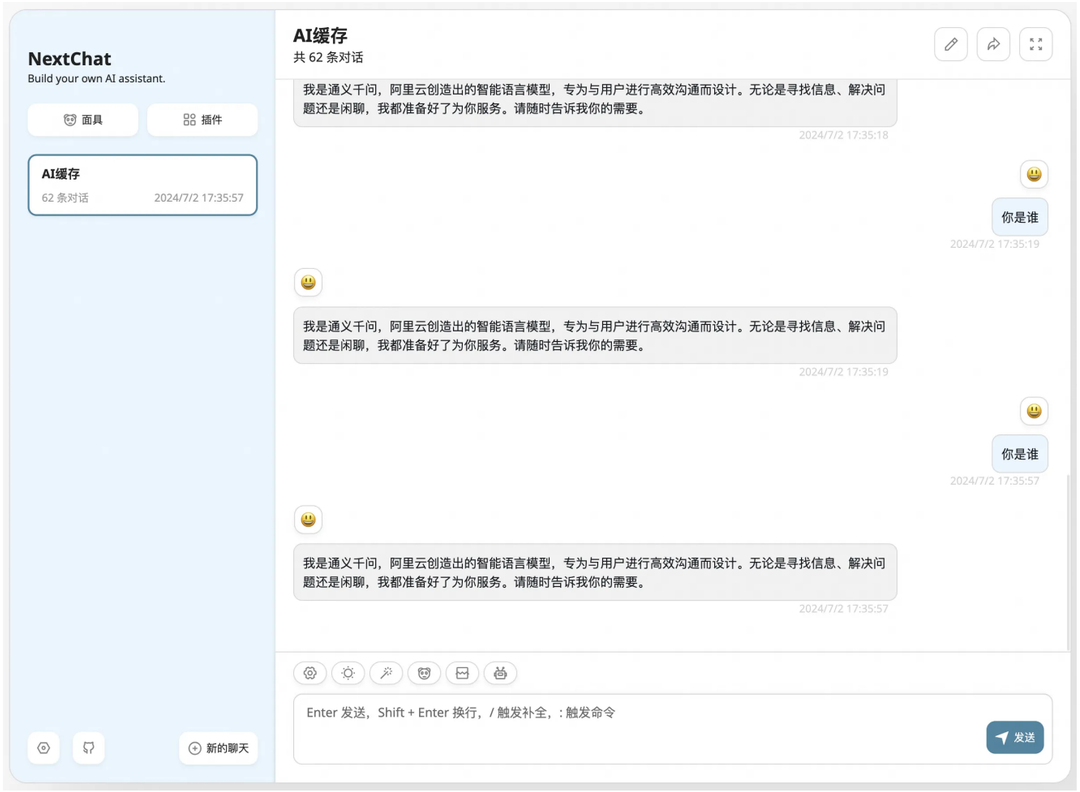
添加 AI 缓存插件后，应用架构为：

插件配置

YAML | 复制代码

```yaml
redis:
  serviceName: redis.static
  servicePort: 6379
  timeout: 2000
  username: xxxxxx
  password: xxxxxx
```

插件效果



6.4.6 AI RAG

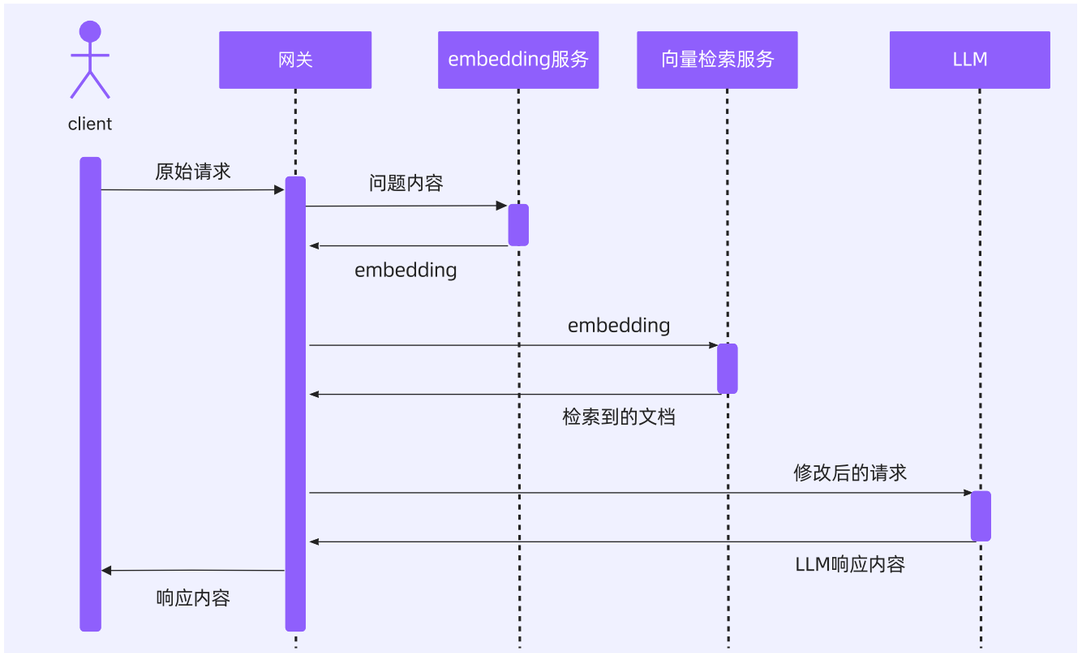
通过对接阿里云向量检索服务实现，可以实现 LLM-RAG，插件的详细文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/ai-rag/README.md

应用架构

大模型具有一个显著的局限性，那就是它们的知识截止到模型被训练的数据。一旦训练完成，模型就无法获取或学习新的信息。此外，大型语言模型的训练数据虽然浩如烟海，但仍然有可能缺少某些领域的信息，或者对某些主题的覆盖不够深入，针对这些细领域的查询可能会产生不够精确或缺乏深度的结果。检索增强生成（RAG）技术能够利用检索系统从大规模的数据库中找到相关信息，然后将这些信息提供给文本生成模型以帮助生成更精确、更丰富、更符合实际情况的文本。

Higress 通过对接阿里云向量检索服务 DashVector 能够快速实现 RAG 功能：



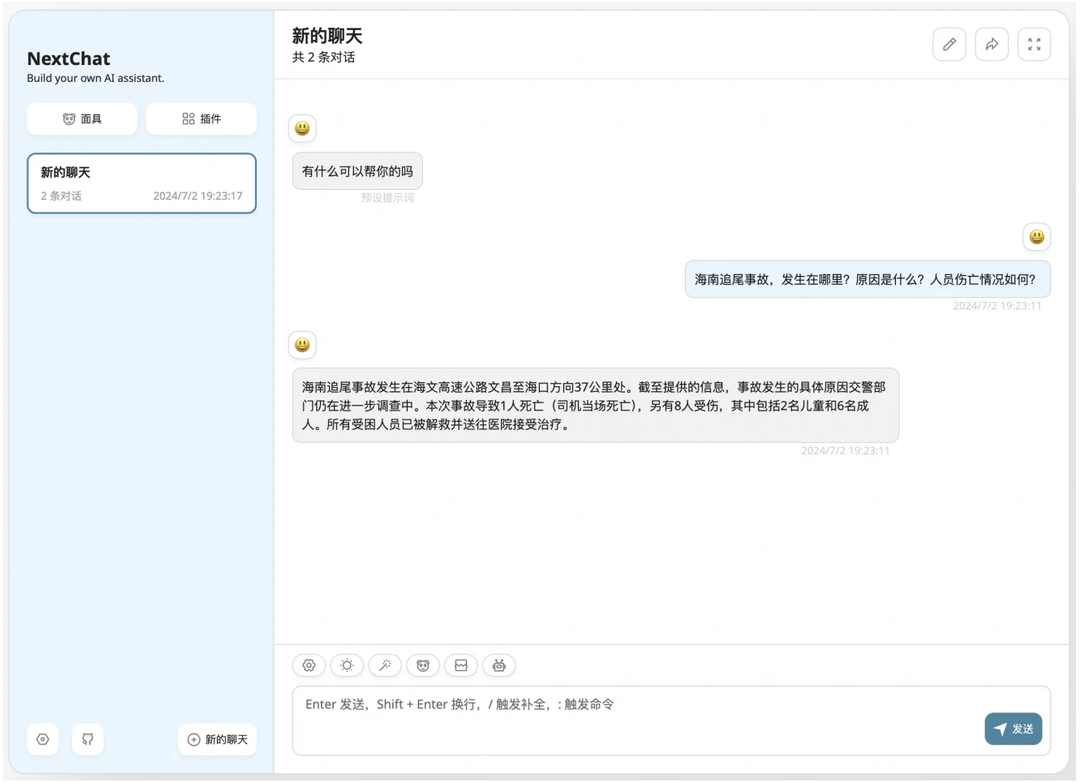
添加 RAG 插件后，应用架构如下图所示：

插件配置

插件需要配置百炼中 DashScope SDK 和向量检索服务 DashVector 的相关信息：

```yaml
dashscope:
    apiKey: sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx
    serviceName: qwen
    servicePort: 443
    domain: dashscope.aliyuncs.com
dashvector:
    apiKey: sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx
    serviceName: dashvector
    servicePort: 443
    domain: vrs-cn-xxxxxxxxxxxxxx.dashvector.cn-hangzhou.aliyuncs.com
    collection: xxxxxxxxxxxxxx
```

插件效果



6.4.7 其他

除了以上插件，我们还提供了对 Prompt 进行修改的插件以及对请求/响应进行智能转换的插件。

Prompt工程相关插件

Prompt 插件包括 Prompt模板 以及 Prompt 装饰器：

- Prompt 模板
- Prompt 装饰器

Prompt 模板允许用户在网关定义一系列 LLM 请求的模板，使用者通过指定模板中的参数对 LLM 进行访问，配置示例如下：

```yaml
templates:
- name: "developer-chat"
  template:
    model: gpt-3.5-turbo
    messages:
    - role: system
      content: "你是一个 {{program}} 专家，你平时使用的编程语言为 {{language}}"
    - role: user
      content: "帮我写一个 {{program}} 程序，你的返回结果里面应该只包含python代码"
```

请求 Body 示例如下：

```yaml
{
  "template": "developer-chat",
  "properties": {
    "program": "冒泡排序",
    "language": "python"
  }
}
```

Prompt 装饰器允许用户在网关定义对 Pompt 的修改操作，包括在原始请求之前和之后插入 message，配置示例如下，请求 Body 与 OpenAI 的请求一致。

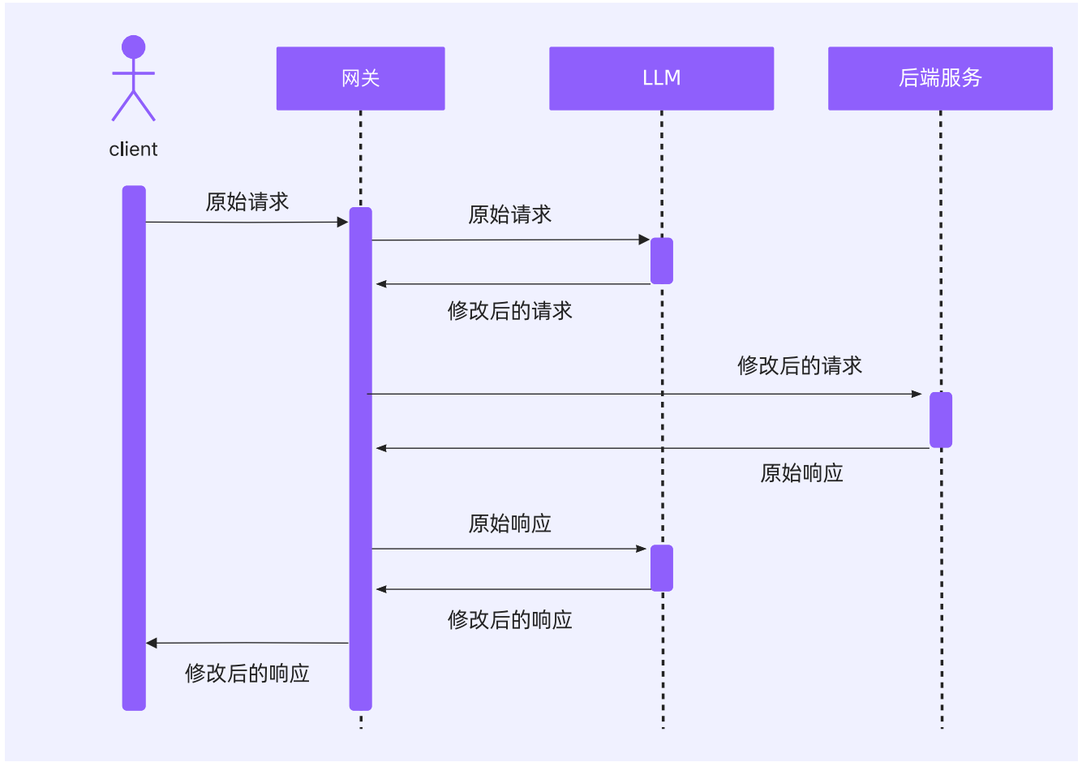
```yaml
prepend:
- role: system
  content: "请使用英语回答问题."
append:
- role: user
  content: "每次回答完问题，尝试进行反问"
```

AI 请求/响应智能转换

AI 请求响应转换插件，可通过 LLM 对请求/响应的 Header 以及 Body 进行修改。插件详细文档：

https://github.com/alibaba/higress/blob/main/plugins/wasm-go/extensions/ai-transformer/README.md

请求响应转换插件支持对请求/响应进行智能转换，其工作流程如下图所示（示例中后端服务为 HTTPBin）：



Prompt 模板允许用户在网关定义一系列 LLM 请求的模板，使用者通过指定模板中的参数对 LLM 进行访问，配置示例如下：

插件配置

```yaml
response:
    enable: true
    prompt: "帮我修改以下HTTP应答信息，要求：1. content-type修改为application/json；2. body由xml转化为json；3. 移除content-length。"
provider:
    serviceName: qwen
    domain: dashscope.aliyuncs.com
    apiKey: sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

此插件可用于修改经过网关的请求/响应内容，例如将 XML 格式的响应修改为 json 格式。

插件效果

访问原始的 HTTPBin 的 /xml 接口，结果为：

YAML 复制代码

```yaml
<?xml version='1.0' encoding='us-ascii'?>

<!--  A SAMPLE set of slides  -->

<slideshow
  title="Sample Slide Show"
  date="Date of publication"
  author="Yours Truly"
  >

  <!-- TITLE SLIDE -->
  <slide type="all">
    <title>Wake up to WonderWidgets!</title>

  </slide>


  <!-- OVERVIEW -->
  <slide type="all">
    <title>Overview</title>

    <item>Why <em>WonderWidgets</em> are great</item>

    <item/>
    <item>Who <em>buys</em> WonderWidgets</item>

  </slide>


</slideshow>
```

使用以上配置，通过网关访问 HTTPBin 的 /xml 接口，结果为：

YAML 复制代码

```yaml
{
  "slideshow": {
    "title": "Sample Slide Show",
    "date": "Date of publication",
    "author": "Yours Truly",
    "slides": [
      {
        "type": "all",
        "title": "Wake up to WonderWidgets!"
      },
      {
        "type": "all",
        "title": "Overview",
        "items": [
          "Why <em>WonderWidgets</em> are great",
          "",
          "Who <em>buys</em> WonderWidgets"
        ]
      }
    ]
  }
}
```

ALIBABA CLOUD

6.5 API 和 Agent 的货币化

AI 应用正在越过“聊天即产品”的早期阶段，进入以任务完成为目标的 Agent 时代。我们可以看到业界 Agent 经济的一些趋势：

- **行业进展不均衡：**通用型 Agent 率先成熟，行业垂直 Agent 加速追赶，价值闭环集中在“降本、提效、增体验”三件事上。
- **技术拐点临近：**强化学习与复杂推理让 Agent 从“Prompt 驱动”走向“专家型”，稳定性和可解释性同步提升。
- **商业模型迁移：**从 SaaS 的功能订阅走向 RaaS（Result as a Service）结果付费，但“如何度量结果并据此定价”仍是货币化难点，短期以混合定价更稳妥。
- **落地方法明确：**优先攻坚企业核心流程的真痛点，量化 ROA/ROI，把准确率拉到可用阈值，才能形成持续使用与复购。
- **现金流优先区：**客服、营销、数据分析等流程清晰的场景率先验证；金融、医疗等高价值垂类仍需技术、组织与生态协同突破。

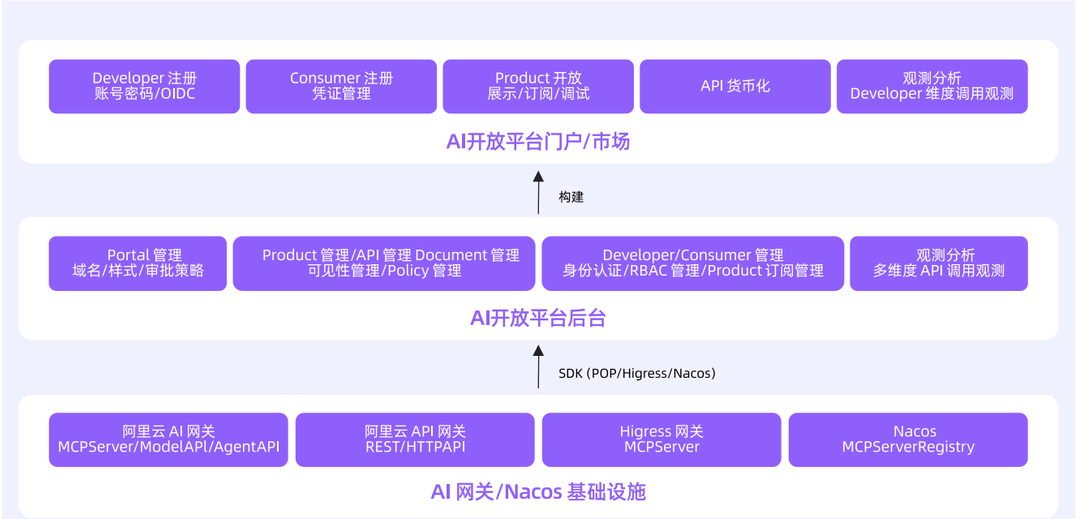
在企业加速数据驱动转型的背景下，生成式 AI 与 AI Agent 快速普及，将 AI 嵌入现有应用已成为主流落地路径，AI Agent 也被视为 GenAI 商业化的关键能力之一。在这个趋势下，企业基于企业原有业务结构与私有数据，参与并构建私有的 Agent 市场，可以进一步实现规模化的 AI 创新、开发、治理与货币化，把 Agent 转化为可计价、可审计的“数字劳动力”，可以持续放大企业经营洞察与增长效应。

Agent 不仅要理解意图、规划步骤，还要以安全可控的方式访问工具和企业数据，能在权限与合规边界内执行真实操作。要让这样的能力从试验室走向规模化分发，AI Agent 开放平台必须提供统一入口，将开发、上架、订阅、计费与治理贯通起来，让 Agent 能像云服务与应用一样被标准化消费。

阻碍企业规模化创新的根因，在于能力与接口的碎片化。工具调用、检索与工作流散落在不同团队与厂商的封装里，参数规范、错误语义与调用生命周期各说各话，导致复用性差、可靠性不稳。与此同时，计量口径不统一、成本不可见，使得对外报价、内部对账与生态分成无从谈起。再叠加跨云迁移的高成本与治理缺位（权限、审计、SLA、合规难一体化），任何尝试构建可持续的商业闭环都会被运营摩擦吞噬。

6.5.1 AI 开放平台

破解之道是以 AI 网关为中枢，协议化承载模型、工具、数据与工作流，把“能用什么、怎么用、怎么计费与如何被治理”一并纳入平台底座。AI 网关负责抽象并统一上游推理与下游工具，提供稳定的路由、编排与策略面；协议层对工具与资源进行自描述，收敛参数、错误与权限语义，消弭供应商差异；在此基础上，以 API 货币化为抓手，构建可计量、可计费、可分成、可审计的商业框架，最终形成 Agent 开放平台或 Agent 市场，建立“供给–分发–变现–治理”的闭环。



上图所示是开源项目 HiMarket AI 开放平台的三层架构，底座是 AI 网关/Nacos 基础设施，承载 Model API/Agent API 与 MCP Server 的统一接入、注册与发现，打通REST/HTTP 与 AI 推理流量，提供高可用路由、鉴权、限流与重试，并将计量埋点前置到网关侧，形成稳定、可控、可观测的执行面。

其上是 AI 开放平台后台，负责 Portal 管理（域名、样式、审批策略）、API 与文档管理、可见性与策略管理、身份认证与 RBAC、订阅与审计，以及多维 API 调用观测，作为企业级后台把开发、运营与合规织成闭环。

HiMarket 开源地址：

https://github.com/higress-group/himarket

6.5.2 AI 网关

从上图中我们看到，AI 网关是 AI 开放平台的底层基础设施。

AI 网关在入口实现身份鉴别、租户隔离与最小权限授权，在数据路径上完成脱敏、驻留与加密，在推理路径上进行模型路由、降级与并行编排，并以统一错误模型与重试策略提升鲁棒性。对于工具与资源，网关鼓励以标准 Schema 自描述输入输出、标注幂等与副作用等级，支持流式与异步调用、长文档的资源引用与分页传输，从而既降低上下文成本，又提升可复现性与可调度性。

6.5.3 API 货币化

API 货币化是市场成立的前提。

首先让一切可被度量：请求次数、输入输出 token、推理时长、并行度、外部工具直通成本都需被精准采集与归因，形成统一账单与成本画像，完成 AI 资产的分账诉求。

随后基于这些指标设计价格计划：按量与订阅并存，支持企业合约、试用额度与超额计费；对复杂工作流可提供“每次运行”或“每任务完成”的计价模式。对于分成引擎来说，则将平台费、开发者收益与第三方工具直通费透明拆分，实现可持续的生态激励。

6.5.4 API 开发者门户

面向开发者，Agent 开放平台提供自助式上架到变现的全链路体验。开发者在门户完成能力注册、参数与权限声明、版本与依赖管理、测试与基准评测，随后配置价格与分成策略，提交上线审核。平台回馈用量、收入与质量指标，并提供问题排查、回放与 A/B 路由工具，帮助持续优化可靠性与单次任务成本。这样，开发者的能力沉淀为可交易的“Agent 商品”，摆脱一次性交付的天花板。

6.5.5 Agent 市场

面向企业客户，Agent 市场提供统一目录与受控分发。企业可在同一控制台选择通用或垂直 Agent，绑定自身数据源与权限策略，配置配额、预算与费控阈值，获得透明的 SLA、审计轨迹与合规模块。对于有特殊安全诉求的客户，平台可支持私有化部署或私域市场形态，让采购、集成与运维遵循既有的 IT 治理流程，同时保留跨云与多模型的可移植性。

1. 治理是 Agent 市场可持续的底线工程

平台需要将内容安全、越权与注入检测、异常用量风控与反作弊、数据出境与驻留策略、可撤回与删除权等内化为默认能力，以策略即代码的方式统一配置与审计。通过可观测性与分布式追踪，将每一次工具调用、模型选择与策略判定记录为可回放的事实，为问题定位、成本优化

与合规稽核提供依据。

2. 跨云与可移植性关乎长期弹性与议价能力

通过协议化的工具与资源层，以及模型无关的路由策略，平台可以在不同模型与云环境之间进行性能与成本基准，对关键工作负载实施多活或按需切换。在同等能力下，选择更具性价比或更合规的供应商无需重写上层 Agent，只需调整路由与策略，真正做到“能力一次接入，处处可用”。

3. 随着供给与需求的双侧累积，Agent 市场会形成正向飞轮

更多高质量工具与数据接入，催生更强的复合型 Agent；更好的体验提升转化与复购，带来更高分成与开发者收入；收益驱动更多供给入场，市场规模与品类扩展进一步增强平台护城河。平台可在此基础上引入评测榜单、企业认证与行业模板，降低选择成本，鼓励“拿来即用”的场景化组合。

API 和 Agent 的货币化绝不是“加个计费按钮”，而是一套以 AI 网关为中枢、以协议为通用语言、以市场为分发载体的系统工程。它把分散在各处的模型、工具与数据，重塑为可发现、可计价、可治理的商品与服务：平台方获得新的营收曲线与网络效应，开发者获得持续变现通道，企业获得统一、安全、可移植的智能入口。随着生态成熟与最佳实践沉淀，这样的开放平台会像城市的主干道，连接起智能供给与业务价值的每一处场景。

AI 应用运行时

AI Runtime

CHAPTER

7.1 AI 应用运行时的演进趋势

7.1.1 从云原生到 AI 原生

信息技术的发展史，是一部计算范式不断演进的历史。从大型机时代的集中式计算，到客户端-服务器架构的分布式计算，再到过去十年由微服务和容器化技术定义的云原生时代，每一次范式的跃迁，都旨在解决前一个时代的根本性矛盾，并释放出新的生产力。如今，我们正站在第四次计算浪潮的黎明 AI 原生时代。

云原生时代的核心是应用。我们围绕应用构建了以 Kubernetes 为事实标准的容器编排系统，以阿里云函数计算为代表的无服务器计算（Serverless/FaaS），以及一套完整的 CI/CD、可观测性和服务治理理论。这些技术的共同目标是：将单体应用拆解为更小、更独立、更具弹性的微服务，并高效地在云基础设施上进行部署、扩展和管理。这个范式取得了巨大的成功，但其核心假设仍然是：计算任务是由人类预先编写的、逻辑确定的代码驱动的。

然而，LLM 的崛起，催生了一种全新的计算实体——自主智能体（Autonomous Agent）。Agent 的行为不再由确定性的代码逻辑严格限定，而是由一个宏大的目标和一个或多个模型所驱动。它能够自主地进行规划、推理、决策，并调用外部工具来与数字世界和物理世界进行交互，以达成指定目标。一个 AI 软件工程师 Agent 的工作流，不再是执行一段固定的 main() 函数，而是一个动态的、长周期的、充满不确定性的“思考-行动”循环。

这种根本性的转变，使得我们开始重新审视云原生时代构建的基础设施。为运行持久化应用而设计的虚拟机和传统容器，因其沉重的资源占用、缓慢的启动速度和高昂的闲置成本，无法适应 Agent 所需的极致弹性和成本效益。而为处理确定性、无状态、短周期的 HTTP 请求而设计的传统 Serverless 架构，也无法承载一个需要数小时上下文记忆、拥有完整文件系统、并能与浏览器进行复杂交互的 Agent。我们需要为 AI 应用而生的原生运行时。

7.1.2 Agentic AI 应用的典型场景

从为了更清晰地理解 AI 原生应用的需求，我们以云上企业客户落地 AI 业务场景为例，并尝试拆解这些典型应用的业务行为。

场景一：交互式智能内容创作助手

- **场景描述：**市场营销科技公司的经理在一个富文本编辑器中输入初稿，AI 助手可以实时地进行润色、扩写、总结。它还能进一步执行指令，如“帮我配一张科技感的头图”、“将这段翻译成英文”等。
- **应用行为拆解：**
  - 多轮对话：整个创作过程是一个持续的多轮人机交互会话。
  - 上下文记忆：AI 必须记住之前的文稿版本、用户的修改指令和偏好，才能提供连贯的创作建议。
  - 模型自托管：助手生成图片、润色文本等任务需要调用不同的模型。例如，生成符合公司过往风格的图片，可能需要依赖对公司素材进行微调后的垂类模型，并将其托管在如 ComfyUI 这样的服务上。
  - 异构任务流：这个过程混合了多种任务。文本润色前的内容预处理，然后交给 LLM，而配图则需要调用文生图的扩散模型。
  - 流式响应：为了更好的用户体验，AI 的回答，特别是长文本，需要像打字机一样逐字流式输出。

场景二：个性化 AI 客服

- **场景描述：**某电商平台的“主动式导购 Agent”。当一个用户在某个昂贵的商品页面停留超过 3 分钟，并反复查看评论和规格时，AI Agent 不是被动等待提问，而是主动发起对话：“您好！我是您的专属 AI 导购……”。
- **应用行为拆解：**
  - 事件驱动：Agent 的启动是由复杂的业务事件（如用户行为日志、定时器）触发的，而不是简单的 API 请求。
  - 企业数据：在发起对话前，Agent 需要瞬间拉取并整合多个数据源：用户的历史订单（CRM）、商品知识库（向量数据库）、实时库存（ERP）等。
  - 实时在线：Agent 需要全天候在线，随时准备响应触发事件。

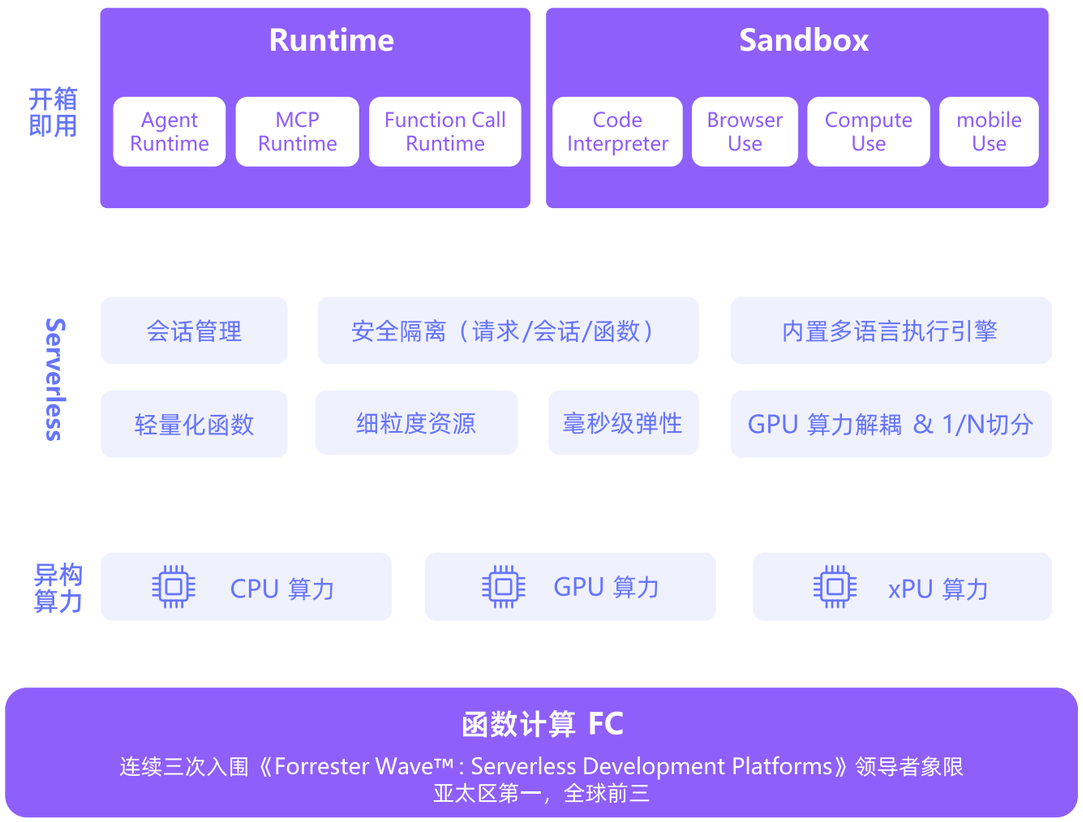
场景三：通用 Agent 平台 + 病毒式传播的 AIGC 创意应用

- **场景描述：**某科技企业是国内一家头部的基础大模型服务公司，利用自己的大模型结合面向 C 端的 Agent 平台开展全栈开发，让平台用户可以通过提示词写出一个完整的项目工程，并且可以一键部署和分享出去。
- **应用行为拆解：**
  - Agent 智能体：通过和用户对话，根据用户输入，以休闲益智类游戏为例，通过 LLM 生成游戏项目代码。
  - 代码执行验证：Agent 智能体通过 LLM 生成代码，然后需要一个 Code Interpreter Sandbox 来执行验证。

- 部署与分享：LLM 生成的游戏项目完成验证后，用户可以快速部署为一个独立服务，然后通过分享对外发布。
- 脉冲式流量：应用可能因为网红的推荐而流量激增，并发请求在几分钟内从 10 个飙升到 10000 个。
- 大文件处理：应用需要处理用户上传的高清照片进行游戏互动，并生成高清结果图，涉及较大的数据负载（Payload）传递。

7.1.3 AI 原生应用运行时的核心能力

从上述真实实践中，我们可以清晰地勾勒出 AI 应用的共同画像：它们是会话式的、工具增强的、事件驱动的、异构算力的，并且需要极致弹性与精益成本的。这要求运行时基础设施具备以下核心能力：



1、面向会话的状态管理与安全隔离

Agent 的核心概念是会话（Session），它是持久的、有状态的。Agent 在其生命周期中积累的上下文信息，包括内存中的变量、本地文件系统中的代码和数据等，是其能够进行多步复杂推理的基础。因此，基础设施必须为每个 Agent 会话提供一个完全隔离且状态持久的运行时。

由于 Agent 执行的代码很大一部分是由 LLM 动态生成的，其行为具有不可预测性，必须被视为不可信的第三方代码。传统的基于操作系统的容器隔离共享同一个内核，这为内核漏洞利用和容器逃逸等攻击留下了风险。因此，硬件虚拟化级别的强隔离（例如 MicroVM）是必不可少的，以确保最基本的运行时安全边界。运行时需要提供会话管理和资源调度，以实现会话维度的运行时隔离和存储隔离，确保数据安全，并实现强隔离、快启动。

2、大规模实时弹性与精益成本管理

Agent 应用的负载模式通常是高度动态和不确定的。例如，一个集成到热门应用中的 AI 助理，需要在流量高峰的几秒钟内，从数百个并发会话扩展到数千甚至数万个。同时，许多 Agent 应用场景是交互式的，用户需要近乎实时的响应。这就要求 Agent 的运行环境必须能够瞬时启动，将冷启动时间控制在毫秒级。传统的虚拟机启动需要数分钟，容器冷启动也通常在数秒到数十秒之间，都难以满足需求。

此外，Agent 的工作模式并非持续的高强度计算。在漫长的空闲或等待时间里，传统基础设施的计费仍在持续，导致巨大的资源浪费。理想的基础设施，必须能够将成本与 Agent 的真实工作量严格挂钩。平台需要通过精细的运行时状态监控和调度引擎，精确区分 Agent 或工具的“忙时（Active）”和“闲时（Idle）”状态，提供面向“忙闲时”的计费模型，真正实现为价值付费。

3、异构算力与标准化工具连接能力

AI 应用通常需要精细化调度 CPU、GPU 等多种异构资源。CPU 提供毫秒级弹性，而 GPU 提供秒级弹性能力。业务侧需要能够方便地组合不同类型的算力，平台侧则需要提供开箱即用的算力使用能力，实现快速交付和极致的成本效益。

同时，AI 智能体、AI 工具和各种沙箱环境之间需要标准、安全的连接能力。协议层需要原生支持以 MCP、A2A 等标准协议连接智能载体，原生支持流式请求响应，并结合 AI 网关提供灵活、安全的访问鉴权能力，方便各种组件的集成。此外，平台还需具备原生异步化处理能力，自动应对大规模的异步请求，并提供平台错误处理能力，以支持 Agent 及工具常见的长时任务处理方式。

7.2 模型运行时

模型是 AI 应用的核心基石。当前企业在模型部署上面临两种技术路径的权衡：模型即服务（MaaS）虽免运维却受限于数据合规与定制化瓶颈；自托管（Self-hosted）方案虽可控却深陷 GPU 利用率低下、弹性不足、运维复杂的困局。经过大规模生产实践验证，Serverless 运行时成为模型的最优解。

7.2.1 企业模型使用的核心痛点

案例1：景区 AI 生图的资源浪费困局
某 4A 景区设计师希望为游客照片生成国风特效，使用 ComfyUI 部署绘图模型。但景区流量波动极大：节假日单日处理 10 万张图，平日仅千张。传统虚拟机方案被迫包月预留峰值 GPU 资源，导致 90+% 算力闲置，月成本超5 万元。更致命的是，突发流量时扩容 >5 分钟，游客排队超时投诉激增。

案例2：儿童阅读 App 的冷启动灾难
某儿童阅读平台用 CosyVoice 模型生成故事语音。其 2万+ 绘本中，热门内容日均调用 10 万次，而长尾绘本月均不足 10 次。自建容器集群无法为低频模型保持实例存活，家长点击冷门绘本时需等待 60+s 冷启动，体验断裂导致用户流失。

案例3：智能家居企业的定制化困境
某智能家居企业用 Qwen 模型分析家庭安防视频。需适配不同硬件终端（摄像头/门锁/电视），每个场景需定制化微调模型。自建 IaaS 平台每次部署新模型需配置 GPU 环境、压测验证，迭代周期长达3天，严重拖慢产品创新速度。

当 AI 从实验室走向万千真实企业场景，传统方案在成本、弹性和效率上的短板被急剧放大，而 Serverless 模型运行时以无服务器理念重塑 AI 生产关系：

- **对开发者：**告别“GPU 环境配置-集群管理-模型部署”的非核心工作，专注场景创新；
- **对企业：**破解“不用时浪费，用时又不足”的算力悖论，使规模化不再伴随成本失控；
- **对产业：**当景区设计师、儿童产品经理、智能家居工程师都能轻松用好领域模型，AI 应用才会真正繁荣。

7.2.2 Serverless 模型运行时的核心能力

1、异构算力和 1/N 卡切分使用
在算力层，Serverless 模型运行时通过 GPU 虚拟化技术将单张 GPU 显卡划分为 N个独立计算单元，每个实例具备隔离的显存空间与算力资源，不同实例通过 GPU 分时复用技术实现并行推理，以 16GB 显存的 Tesla 卡系列为例，Serverless 模型运行时具备将单张卡切分为 16个实例，每个实例均拥有 1GB 显存和 1/16 算力，使 ≤1GB 的小参数模型可运行在单个实例，实例和实例间互不影响，实现资源隔离和数据隔离，进而大幅简化开发范式，减少稀缺 GPU 资源的碎片化浪费。

Serverless 模型运行时通过池化技术，将 CPU/GPU/XPU 统一纳管到一个资源池，开发者可根据模型特性按需配置算力类型 - 语音识别等轻量任务分配 CPU，图像生成等算力密集型任务分配 GPU 碎片，大语言模型等显存密集型任务分配 GPU 整卡或多卡。某服装企业实测显示，Stable Diffusion 服务采用 1/8 卡虚拟化后，GPU 利用率从 18% 提升至 89%，成本降幅达 83.2%，逼近理论极限值 87.2%。这种碎片化供给模式，从根本上重构了算力经济模型。

2、负载感知调度和毫秒级闲置唤醒
在调度层， Serverless 模型运行时通过负载感知调度系统实时监测请求队列深度、GPU 显存占用率、实例健康状态等多维指标，基于池化技术构建三级响应机制：请求优先分配至活跃实例；当资源吃紧时，毫秒级唤醒闲置实例；仅在极端流量下触发冷启动。其核心技术突破在于利用 CRIU（用户空间检查点/恢复）技术冻结显存状态，并将显存数据临时置换至内存，并在新的请求调度前实现毫秒级/秒级状态恢复，较传统虚机/容器方案提速百倍。某语音类应用接入后，长尾模型冷启动延迟从 47 秒压缩至 0.8 秒，流量高峰期的 RT 抖动减少 80%，配套的成本模型使闲置实例仅按 15% 标准计费，实现弹性与经济的平衡。

3、集成加速框架和开发调试工具链
在开发层，Serverless 模型运行时预集成加速框架深度优化模型运行效率：vLLM框架的Paged Attention 技术通过显存分页管理提升3倍吞吐量；SGLang 的 RadixAttention 实现注意力机制并行编译，降低 60% 推理延迟；TensorRT-LLM 的量化融合策略提升2倍能效比。

开发工具链提供 DevPod 交互式环境，开发环节实现白屏化操作和实时反馈，集成在线 IDE 如 VSCode/JupyterLab/SSH 终端，开发者在云端环境具备比本地环境更高的生产效率；生产部署环节实现革命性简化 ——上传模型文件后，系统在 30 秒内自动生成 Dockerfile、构建推理服务、输出OpenAPI 文档及 SDK。某智能家居企业模型迭代周期因此从 3 天缩短至 30 分钟。企业级运维能力涵盖金丝雀发布（支持 5%/10%/20% 阶梯放量）、动态弹性扩展（实例数量根据定时

策略或水位策略在 0~N 间自适应调整）、全观测体系（实时追踪 50+ 指标如 GPU 显存利用率、SM 利用率）等关键特性。

这三层技术并非孤立存在，而是形成深度协同的模型运行时能力增强：GPU 碎片化技术提供原子级算力单元，为智能调度奠定资源基础；负载感知引擎通过毫秒级实例弹性，将碎片化算力转化为即时服务能力；开发工具链则构建自动化流水线，使技术红利直达开发者工作台。某 4A 景区实践表明，当三项技术叠加应用时，AI 生图服务峰值吞吐量提升 12 倍，同时成本降低 78%。这种协同效应标志着 AI 模型托管从手工作坊时代迈入工业自动化时代，将企业从算力枷锁中解放，重归业务创新的本质。

7.2.3 Serverless 模型运行时——AI 大脑的终极载体

从 Serverless 到 Serverless AI，本质是从资源效率优化迈向智能生产力释放。而 Serverless 模型运行时作为承载 AI 大脑的核心基座，通过异构算力革命、智能调度进化、开发范式升级三重突破，让企业真正实现：所想即所得的模型服务，无需担忧算力枷锁，专注智能本身，灵活扩展和不断创新。

7.3 智能体运行时

7.3.1 AI 应用形态的持续演进

1、“请求-响应”模式：无状态的事务性 AI 任务

在 AI 应用的早期阶段，其交互模式很大程度上继承了传统 Web 服务和微服务的“请求-响应”模型。其核心是执行一个独立的、原子性的任务，具有显著的无状态性。

这种模式下，AI 更像一个高效、自动化的工具，其无状态的特点与 Serverless FaaS 架构非常契合。因为计算层是无状态的，平台可以近乎无限地水平扩展，通过并行启动成百上千个独立的、可互换的函数实例来应对流量洪峰，实现了极高的弹性和容错性。

2、“对话”模式：有状态的协作式 Agentic AI 应用

AI 应用的关注点从单纯驱动 LLM 完成指令，转移到了如何利用外部工具和知识库构建更加智能的交互式 智能体（Agent），一个任务需要经历多轮交互才能最终完成，具有显著的会话特征。

从事务性到对话式的转变，应用的用户界面发生了很大的变化，会话上下文贯穿整个交互的生命周期。在 Agent 应用中，会话状态的持久化及会话执行上下文共享，成为了解决 Agent 应用性能的先决条件，应用的架构优先级发生了根本性变化，会话及状态管理成为了架构关注的第一要素。

7.3.2 Agent 运行时的核心架构目标

1、围绕会话请求和资源调度模型，维持长时运行的状态延续

Agent 运行时的架构设计，以会话（Session）作为不可分割的原子单元，构建一个原生支持状态持久化的大规模实时弹性系统。这个目标与云原生时代常见的“请求-响应”模型形成了鲜明的对比。后者的弹性，本质上是一种无状态弹性，通过快速复制无状态的计算实例来应对流量变化。然而，在面对需要记忆和上下文共享的 Agent 应用时存在较大的适配负担，状态被迫成为需要被外部化管理的包袱。

我们认为，会话本身，连同其内部状态，是实现下一代弹性的核心。因此平台设计的出发点不再是“如何高效地处理一个瞬时请求”，而是“如何将一个完整的、有状态的、长周期的 Agent 会话作为一个整体，进行高效、安全、经济的生命周期管理”。然后将 Agent 会话（包含其代码、内存、文件系统）封装成一个可冻结、可恢复、可迁移的独立单元。基于这种能力，能够将会话的逻辑生命周期与物理计算资源的分配彻底解耦。这种解耦是实现大规模实时弹性的关键。

2、实现面向 Active-Idle 资源管理，解决长时运行的成本困境

Active-Idle 资源管理，简单来说就是基于应用“忙-闲”时的资源管理，结合 Agent 运行特征，我们可以将一个会话所需的资源分解为两个部分，要突破成本困局，本质是要提供一种基于会话生命周期的动态资源解耦方案。

- **状态资源：**包括持久化文件系统和冻结的内存快照。这些资源的特点是存储成本低廉。
- **计算资源：**包括活跃的 vCPU 和内存。这些资源是昂贵的，也是传统模型中造成浪费的主要来源。

Active-Idle 模型的核心，就是根据 Agent 的实时活动，动态地、透明地为会话“挂载”或“卸载”昂贵的计算资源，同时始终保持其廉价的状态资源在线。

7.3.3 现有架构支撑 Agent 运行时对比分析

结合 Agent 运行时需求，本节将对目前几种主流解决有状态 Serverless 架构模式进行系统性的对比分析。它们在性能、成本、复杂性和可扩展性等方面存在显著的差异。

技术方案	状态外置适配无服务器架构	传统尽力会话亲和	平台原生状态抽象
状态范围	请求级，外部持久化	会话级，内存中（脆弱）	实体级，平台持久化
弹性效率	极高（按请求独立扩展）	低（破坏水平扩展）	高（按实体独立扩展）
系统容错性	高（函数本身无状态）	低（实例故障导致状态丢失）	高（状态由平台持久化保证）
业务性能	中（受网络延迟影响）	高（内存访问）	中（持久化状态访问有开销）
开发复杂度	中（需手动管理状态读写和一致性）	高（路由复杂，状态易丢失，调试困难）	低（状态管理由平台抽象）
成本模型	按请求付费 + 状态存储成本	按请求付费（但通常需预置实例以保证状态）	按操作/状态存储付费
理想场景	状态需求简单、对延迟不敏感的应用	迁移需要粘性会话的遗留应用	事件溯源、复杂工作流编排

分析与建议：

- **业务状态外置：**架构的优点是简单、清晰，保持了计算层的完全无状态，但实际改造过程中常常面临两个常见的架构适配难题而常常被决绝：状态解耦编程模型复杂度高，状态外置引入性能和成本问题。
- **传统会话亲和：**应被视为一种过渡性的手段。亲和性无法得到保证，属于尽力亲和；同时在会话维度共享实例，无法提供隔离能力，无论是系统升级，还是业务异常，调度不确定导致应用容错无法保障。
- **内置状态抽象：**架构理念先进，但需要依赖特定产品能力，同时需要适配状态内置的编程模型，更适合需要可靠状态管理和复杂工作流编排的场景，并不适合作为编程模型自由度较大的 Agent 运行时。

7.3.4 为 AI 原生的“会话式”Serverless 运行时

这种架构认识到，AI 应用的会话既需要状态持久化，又需要高性能的本地计算和会话维度隔离能力。在这种趋势下，Serverless 业界领导者阿里云和亚马逊云科技（AWS）不约而同给出了自己的答案：

- 阿里云凭借在大规模、高并发等极致业务场景的实践，依托 Serverless 产品函数计算突破架构边界，原生提供会话管理能力，支持为每一个用户会话动态配置一个专用的、持久化实例，支持会话维度的动态挂载实现存储隔离，会话绑定的实例可以持续存活长达 8 小时，未来最长可达 24 小时，并能够保持请求的优雅结束。

- AWS 凭借其先发优势和对市场的定义能力，发布了 AWS Bedrock AgentCore 产品作为构建 Agentic AI 应用的底层基础设施；其核心架构创新在于，它为每一个用户会话动态配置一个专用的、持久化的实例，该实例可以持续存活长达 8 小时。

这种“一个会话，一个独立运行时”的模型解决了传统 Serverless 的所有状态难题：

- **原生会话状态保持：**将“会话（Session）”提升为状态、请求和资源管理的“一等公民”，在一个会话的生命周期内，会话上下文、中间计算结果和临时文件都可以直接存储在实例的内存和本地文件系统中。这消除了因状态与外部系统交互而产生的网络延迟，为 Agent 的多步推理提供了性能保障。只有第一次调用可能经历冷启动。后续的所有交互都在同一个“热”的上下文中执行，响应时间稳定在毫秒级。

- **灵活可靠的安全隔离：**运行时底层计算实例，基于 MicroVM 提供了硬件辅助的虚拟化隔离。每个实例都有自己独立的内核和内存空间，从根本上杜绝了传统容器逃逸和旁道攻击的风险。同时平台提供了会话级别的强隔离选项，每个会话可以选择运行在独立的 MicroVM 实例中。这对于处理敏感数据、执行不可信代码的多租户 Agent 应用，有效防止了会话间的数据泄露。

- **毫秒级的弹性速度：**根据 Firecracker 官方数据，运行时底层基于的 MicroVM 冷启动时间可以稳定在 ~100 毫秒。每个 MicroVM 的内存开销仅为 5MB 左右，使得可以在一台物理服务器上高密度地运行数千个独立的 MicroVM 实例，解决会话粒度的实例弹性问题。

这种架构可以被称为 Agent 原生“会话式”Serverless 架构，既保留了 Serverless 的免运维、按需扩展的优势，又提供了传统长连接服务才有的状态保持和高性能，在此基础上同时兼顾了安全性。同时，平台负责解决了 Serverless 架构有状态环境中生命周期管理、业务升级和成本等一系列难题：

- **会话生命周期管理：**计算实例的生命周期不再是单个请求，而是整个会话。例如，Runtime 的会话有明确的生命周期状态： Active （活动）、 Idle （空闲）、 Expired （过期）、 Deleted （删除） 。平台需要提供 API 允许开发者主动管理（如提前终止）这些长生命周期的会话，便于业务集成，将会话管理的权利交给用户，提升用户使用的灵活性。

- **优雅升级与灰度发布：**当更新一个正在处理有状态会话的函数时，不能简单地用新版本替换所有实例。平台需要支持优雅升级：新的会话请求被路由到新版本的实例，而持有旧会话的实例则继续服务，直到其关联的会话自然结束，从而实现平滑过渡。

- **匹配运行特征的成本模型：**通过实施以会话为中心的 Active-Idle 资源管理模型，平台能够让开发者无需关心底层资源的复杂管理，就能享受到永远在线的会话体验和按真实计算付费的经济效益，从而解决开发者和企业构建 Agent 的成本困局。

7.4 工具与云沙箱

7.4.1 AI Agent 与工具：从概念到能力

白皮书第5章完整介绍了 Agent 是如何借助外部工具来执行具体的任务，从而与外部世界交互，最大限度地释放 LLM 的潜力。这一节我们将聊聊工具的运行时与云沙箱。

由于 AI 工具已经不再是简单的 API 调用，而是演变为需要复杂运行时才能管理和安全保障的执行环境。因此，我们先在下表格提炼了 Agent 常用工具的类型，并进一步剖析了它们背后对底层运行时的核心诉求，这为我们理解 AI 原生应用的新型执行范式奠定了基础。

工具	描述	核心运行时需求
MCP （模型上下文协议）	实现互操作性，允许 LLM 通过结构化 API 动态理解并调用外部工具。	稳健的 API 服务与运行时紧密集成。运行时具备轻量，弹性能力。
File Search （文件搜索/检索）	在预定义的私有文档语料库中执行语义检索，实现检索增强生成（RAG），提高回答的准确性。	高效的向量检索服务与运行时紧密集成。运行时需要具备上下文缓存，提升检索性能。
Image Generation （图像生成）	赋予模型文生图和程序化图像处理的能力，将自然语言描述转化为视觉内容。	高性能计算、高效稳定的 API 调用能力。运行时具备异构算力。
Code Interpreter （代码解释器）	为模型提供一个有状态的、沙箱化的多语言执行环境（Python、NodeJs、Go等），用以执行复杂的计算和程序化任务。这将模型的能力从语言推理扩展到逻辑分析领域，使其能够进行量化数据分析、算法执行和动态数据可视化等操作。	强隔离与安全、持久化会话、丰富的科学计算环境。

Browser Use （浏览器使用）	赋予模型以编程方式控制 Web 浏览器实例（通常是无头模式，headless mode）的能力，以执行复杂的网页自动化任务。这超越了简单的信息搜索，涵盖了导航、表单提交、与动态网页元素交互，以及直接从渲染后的网页内容中提取信息。	强隔离、会话管理、可观测性与人机协同能力。
Computer Use （计算机使用）	使模型能作为一个视觉代理 (visual agent)，直接与计算机的图形用户界面 (GUI) 进行交互。它通过模拟人类的键鼠操作来操控缺少程序化API的桌面应用和遗留系统，从而实现端到端的数字化工作流自动化。	强隔离、具备 GUI 的完整操作系统、会话管理、可观测性。
Mobile Use （移动端使用）	赋予操作系统更强的语义理解和跨应用操作能力，使手机能够自主完成复杂任务，将手机转变为一个能够真正理解用户需求的“智能助手”。	专用、持久化的运行时环境，能够进行跨应用和 OS 级的操作，并需要支持会话管理以保留对话上下文和记忆。

从上述分析可以看出，像代码解释器、浏览器使用和计算机使用这类复杂工具，其核心需求已超越了传统单次、无状态的 API 调用。它们需要一个有状态的、沙箱化的环境。为满足这一需求，云计算基础设施正在演进，而我们常说的 AI Sandbox，正是一个被严格控制的隔离环境。它允许 Agent 在其中安全地执行代码、与应用交互和访问资源，同时有效防止对主机系统或敏感数据的危害。AI Sandbox 的核心价值在于，它为创新提供了安全保障，是推动 Agent 技术走向成熟和商业化应用不可或缺的基础设施。

7.4.2 复杂工具运行时的核心诉求

Agent 的出现，对底层的运行时环境提出了三大根本性的、前所未有的技术挑战。这些挑战解释了为何传统基础设施无法高效、安全地支持复杂的 AI 工具运行时。

- **隔离与安全（Isolation & Security）**：这是首要且不容妥协的要求。执行由 LLM 动态生成且不可信的代码，引入了巨大的安全漏洞，例如沙箱逃逸、代码注入和权限提升等。传统的软件沙箱技术在应对这种动态、复杂且可能具有对抗性的 AI 工作负载时，正变得力不从心，漏洞频发证明了其不足。Agent 对运行时隔离性的要求达到了前所未有的高度。
- **状态管理与成本（State Management & Cost）**：Agent 的工作模式是对话式和持续性的，这需要每个沙箱都拥有一个持久的、有状态的会话来维持上下文、记忆和交互环境。这种特性与传统无状态的 FaaS（函数即服务）设计产生了根本性冲突。若为每一个潜在的用户会话都预置一个长期运行的虚拟机（VM），将导致惊人的闲置资源成本和巨大的运维负担。
- **可扩展性与运维（Scalability & Operations）**：Agent 应用的流量往往是突发且不可预测的。基础设施必须能够瞬时、动态地扩展以应对峰值，并在空闲时迅速缩减以节约成本。然而，从零开始构建并维护一个既安全隔离又具备弹性伸缩能力的沙箱环境，需要极其专业的 DevOps 知识和大量的人力投入，这对大多数开发团队而言都是一个难以逾越的障碍。

上述挑战共同指向一个结论：Agent 的兴起催生了一种独特的云工作负载类型，它不完全符合传统 IaaS 的模式，也对现有的 FaaS 平台在状态管理和隔离性上提出更高的要求，使其兼具虚拟机级的强隔离、状态化管理与 Serverless 的极致弹性与经济性。

7.4.3 Serverless 作为 AI Sandbox 的理想基座

一个成熟的 Serverless 平台，特别是以 FaaS 如函数计算为代表的技术形态，通过其底层架构的演进，已成为解决上述所有核心诉求的理想起点，从底层硬件到上层应用提供了一套无缝集成的、端到端的解决方案。

1、计算隔离，硬件级与内核级双重保障

安全性是 AI Sandbox 的基石，其保障始于最底层的物理硬件。例如，阿里云函数计算采用的“神龙裸金属 + MicroVM 安全容器”架构，构建了一个从硬件到应用运行时的端到端、纵深防御安全体系。神龙架构通过自研的 MOC 芯片将虚拟化功能从主 CPU 卸载，实现了在共享硬件上的租户间“硬隔离”。在此基础上，函数计算为每个函数实例提供独立的 MicroVM 安全容器，使其运行在拥有独立客户机内核的微型虚拟机中。这确保了 AI Agent 生成和执行的代码被完全封装在自身的内核空间里，其影响范围被严格限制在单个、即用即毁的实例内部。这种双重安全模型共同构建了坚固的壁垒。

2、会话管理，原生支持有状态应用

传统的 Serverless 架构被设计为处理无状态的、短暂的事件，而 AI Sandbox 本质上是有状态的。若采用将函数实例数固定为1的“静态预留”变通方案，虽然可以模拟会话粘性，但很快会暴露其致命缺陷：管理成本高昂、破坏自动伸缩能力、丧失按需付费优势等。

而以函数计算为代表的 AI 原生 Serverless 架构通过原生能力解决了这一矛盾。核心能力是 强会话亲和性（Session Affinity）、会话物理隔离（Session Isolation）以及会话管理接口（Session Management Interface）。

- **强会话亲和性：**是一个智能路由层，它确保在一次会话的整个生命周期中，所有来自同一客户端的请求都会被精确地路由到同一个函数实例上，从而为每个用户会话分配了一个固定的函数实例，保证了交互的连续性和上下文的完整性。平台提供了多种灵活的亲和方式，包括专为 Agent 场景设计的 MCP SSE 亲和，以及适用于 Web 场景的 HeaderField 亲和和 Cookie 亲和。
- **会话物理隔离：**解决了多租户环境下的安全问题。在启用该能力后，平台会为每一个会话独占一个完整的函数实例，将单实例会话并发度强制设置为 1。这意味着租户间的内存、临时文件和进程空间在物理上（基于神龙裸金属）和逻辑上都是分离的，从而提供了金融级别的多租户安全保障。
- **会话管理接口：**将这些强大的底层能力通过简洁的控制台配置和 API 接口开放给开发者。开发者无需编写复杂的外部编排和调度逻辑，平台在后台自动处理了会话生命周期与实例生命周期的精确映射和管理，极大地降低了开发门槛。

这三种能力协同工作，共同构建了一个全新的 Serverless 原语：“按需生成的、隔离的、有状态的运行时环境”，完美解决了 AI Agent 对有状态环境的根本需求。

3、存储隔离，解决状态持久化难题

一个完整的 AI Sandbox 解决方案，不仅需要解决计算层的状态管理，还必须应对存储层的持久化和隔离挑战。函数计算通过创新的技术和完善的最佳实践，为 Agent 的两类核心存储需求，本地临时存储和持久化共享存储，提供了端到端的解决方案。

本地临时存储，利用快照技术实现极速恢复

对于需要快速恢复本地环境状态的会话，函数计算引入了基于快照的运行时环境恢复机制。当一个函数实例进入空闲时，系统会自动为其完整的本地磁盘状态创建一个快照，并在下次请求到达时，从该快照“秒级唤醒”一个全新的实例。这种机制巧妙地平衡了成本与性能，为用户提供了持久化会话的体验，而其计费模式却依然遵循 Serverless 的按需使用原则。

持久化共享存储，提供会话级别的数据沙箱

在物理共享的存储资源上实现租户之间数据的严格隔离是一个严峻的挑战。

- Serverless 平台将会话亲和与会话隔离能力从计算层延伸至存储层，通过“会话级存储亲和”（Session-level Storage Stickiness）机制将会话与一个持久化的、归属于特定租户的存储子目录进行强绑定，从而在共享存储之上构建了一个独立的、目录级别的“数据沙箱”，从文件系统层面杜绝了会话间数据交叉的可能性。
- 在会话级存储亲和的基础上，函数计算还提供了基于 POSIX 标准的多租户安全实践框架，给用户提供了一套可落地的多租户存储安全最佳实践。包括：
  - UID 隔离：为每个租户/会话颁发唯一的 POSIX 用户 ID (UID)，确保每个租户或会话在文件系统层面就有了自己独立的“身份”。
  - SecurityContext 继承：确保用户进程在创建文件时主动继承挂载目录的权限，保证挂载目录的权限只为其所有者，从源头上防止了权限混乱。
  - 主进程 UID 切换：Agent 的代码本身就运行在受限的、非 root 的用户身份下，极大地缩小潜在的攻击面。
  - 目录配额：借助存储平台的目录级配额能力，为不同租户/会话的目录设置配额，有效防止某个恶意或行为异常的租户耗尽。
  - UID 复用：确保 UID 回收后，文件内容被清理后才能被再次使用。

4、对极致存储性能的持续探索

对于需要极致 I/O 性能的 Agent 应用，例如大规模代码编译或高频读写海量小文件的场景，函数计算正与高性能存储团队合作提供更高性能的原生挂载能力，支持在会话粒度上为每一个独立的 Sandbox 动态挂载专属的高性能存储盘，以满足最严苛的性能需求。

至此，我们阐述了 Serverless 之所以能作为模型、智能体、工具的理想运行时的原因，下一节我们将从成本角度讲讲 Serverless 的差异化优势。

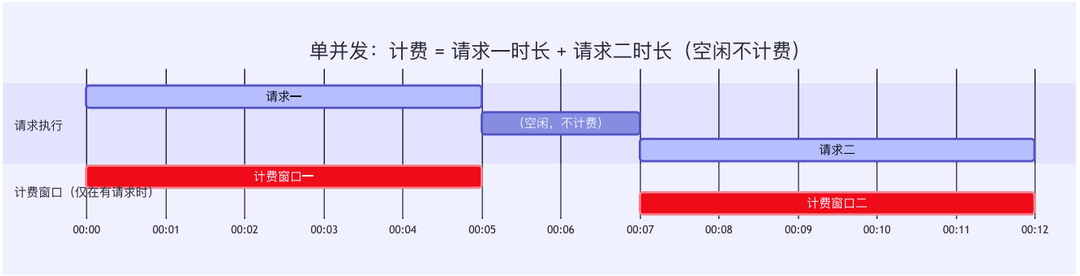
7.5 AI 应用运行时的降本路线

在云计算的发展过程中，计费方式往往是开发者最直观的感知。最初，用户需要直接购买资源，按小时计费；后来，Serverless 函数计算将计费粒度细化到按请求执行的毫秒级。很多开发者第一次接触一款云产品时，关注的往往不是架构，而是账单。因为账单背后映射的，正是云厂商在资源抽象、调度方式、安全隔离与开发体验上的关键选择。

Serverless 函数计算的演进史，其实也是一部计费方式的演化史。透过计费这一窗口，我们可以一管窥全豹，清晰地看到背后产品形态在技术与体验上的深刻变化，以及技术架构随应用场景不断演化的能力。

7.5.1 阶段一：从资源租用到按请求计费

在 Serverless 函数计算发展的最初阶段，最大突破点在于计费方式的根本转变：用户不再像租用虚拟机一样，为实例的持续运行付费，而是只在函数被真正调用、执行时支付费用。换句话说，在没有请求执行的时间段，用户无需承担任何闲置成本，这一阶段的创新，让“只为代码运行时刻付费”成为 Serverless 的立身之本，也迅速降低了开发者的使用门槛。如下图所示。



支撑这种计费模式的关键技术包括：

- **精准识别请求边界：**请求的生命周期就是计费的生命周期，平台必须在微秒/毫秒级准确地识别“开始”和“结束”，保证账单公平与精确。
- **按请求分配独占资源：**每个请求都获得确定的 CPU/内存资源，避免资源竞争导致性能抖动，从而保障账单的可控性。
- **低延时大并发的冷启动能力：**实例不常驻，而是按需启动。平台必须优化冷启动延时，在大规模并发场景下快速分配资源，同时在空闲时立即回收，避免浪费。

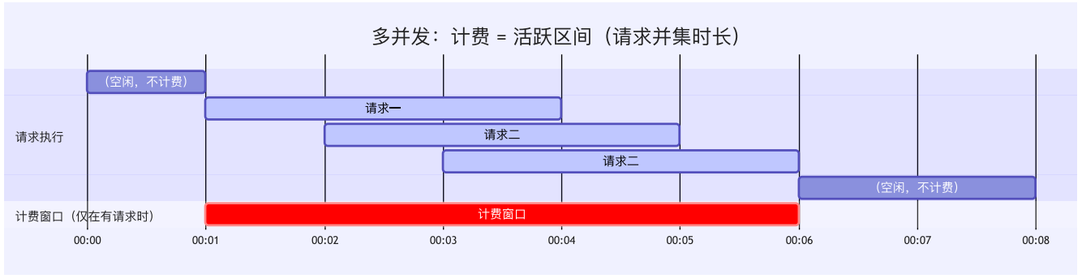
- **1ms 完成活跃/闲置状态转化：**在无请求时通过冻结函数实例的 CPU 调度，转成闲置状态，确保不再消耗时间片，请求来到时候，实时转成活跃状态，允许CPU调度，这是实现毫秒级精确计费和公平性的保障。

这一阶段让函数计算真正区别于虚拟机和容器租用模式，奠定了按请求计费的核心心智模型。

7.5.2 阶段二：多并发 + 毫秒级计费 —— 面向 Web 应用的优化

随着函数计算逐渐普及，除了事件驱动触发外，Web Server 等 I/O 型场景也开始采用 Serverless 架构。如果继续采用单请求独占计费，对比传统多并发的服务模型，原有单实例单并发模型的成本很难接受，因此进入了第二阶段的演化。

核心变化是：突破单并发限制，按函数实例的活跃时间段计费，并将粒度精细化到 1ms，从而支撑 Web 应用、API 服务等主流场景，如下图所示。



支撑这一演化的关键技术包括：

- **识别活跃时间段作为计费边界：**从“单请求时长”转变为“活跃区间”，只要实例内有请求在执行，即视为活跃计费，不管并发多少请求。
- **引入 Custom Runtime / Container Runtime：**支持用户平滑迁移主流 Web 框架（如 Express、Flask、Spring Boot），这些框架天然支持多并发，能够降低成本并收敛数据库连接数，减少连接暴涨带来的风险。
- **缩短计费粒度：**从 100ms 到 1ms：大多数 Web 请求延时低于 100ms，如果仍按 100ms 粒度计费，用户成本过高。精细化到 1ms，使账单更公平。
- **极致优化平台全链路延迟：**Web 应用对端到端延迟极其敏感，平台必须在鉴权、路由、调度、转发等环节做性能优化，避免平台开销成为主要瓶颈。

- **这一阶段的价值在于：**从“为单个请求买单”转变为“为活跃区间买单”，辅以更精细的粒度和运行时灵活性，让函数计算从事件驱动扩展到主流 Web/API 服务场景。

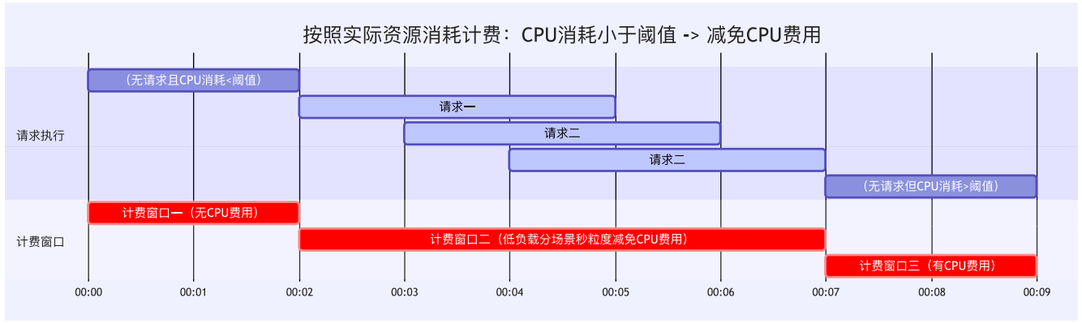
7.5.3 阶段三：按实际资源消耗计费 —— AI 时代的价值计费

AI 应用具有长会话、强交互、低延迟的特点：

- 模型对话需要保持上下文；
- 语音/流式生成需要实时响应；
- 会话中可能包含多种工具调用与后台任务。

这类应用往往是 稀疏型负载：大多数时间处于低负载，仅维持长连接和上下文。传统请求边界=活跃，闲置时冻结 CPU 的机制不再适配：如果一律计为活跃，用户在“低价值”的保活状态下将付出过高成本。

因此，第三阶段的核心转变是：在识别请求边界的基础上，引入按实际资源消耗动态区分活跃/闲置的计费模型。低负载状态下减免 CPU 费用，同时仍然允许 AI 应用运行后台任务。



支撑这种演化的关键技术包括：

- **支持会话亲和性**
  - 引入会话亲和性机制，使得同一会话的请求路由到同一个实例，避免上下文丢失。
  - 用户可通过配置 IdleTimeout 主动控制会话保留时间。
  - 平台针对 AI 应用提供强亲和及会话隔离能力，确保会话绑定实例的有效存活时间。
- **按实际资源消耗判断活跃/闲置**
  - 在过去“有请求=活跃”的基础上，引入根据资源利用率感知活跃/闲置的机制。
  - 如果 CPU 使用超过阈值，则记为“活跃”并计算CPU费用；如果只是心跳/轻量保活，CPU 使用极低，则记为闲置，免去CPU费用，仅收内存/网络成本。
- **执行期间低负载的减免机制**
  - 在有请求执行时，函数计算以秒为周期采样，如果 CPU 使用低于阈值，自动减免该周期的 CPU 费用。
  - 在 MCP、WebSocket 等典型低负载场景默认启用，平台主动让利，避免“在线=计费”的粗暴逻辑。
- **支持不冻结，允许后台任务持续运行**
  - 在 AI 场景中，冻结会导致长连接中断、缓存失效，恢复代价高。
  - 函数计算支持不冻结模式，允许请求结束后继续运行后台任务，如缓存预热、索引更新、回调处理。
  - 这类任务的费用仍然根据实际资源消耗判定为活跃或闲置，差异化计费。

第三阶段的价值在于：从“为活跃区间买单”进一步演化为“按资源消耗分层计费”，账单更好地对齐到有效计算，避免因长连接或低负载保活而产生额外成本，让 Serverless 真正适配 AI 时代的长会话与强交互负载。

7.5.4 函数计算的演化方向是把产品形态与用户价值更紧密地对齐

阿里云函数计算的计费方式经历了三个阶段：

- **阶段一：**按请求计费 —— 降低门槛，让用户只为调用付费；
- **阶段二：**活跃区间计费 —— 扩展场景，让 Web/API 应用也能高效低成本运行；
- **阶段三：**按资源消耗计费 —— 贴近价值，让 AI 应用在长会话与低负载下也能公平付费。

在 AI 时代，函数计算一直坚持走向让开发者只关心业务逻辑，云厂商自动完成一切资源管理与调度的愿景，最终让计算像水、电一样随时可得、按实际使用价值付费。

CHAPTER

8.1 AI 可观测的挑战与应对方案

随着 AI 应用被广泛部署，并呈现出日益复杂的趋势，深入洞察其内部行为显得尤为重要。AI 可观测也应运而生。通过可观测的方法论以及配套的产品工具，可以帮助开发者监控性能、调试问题，并提升内容输出的可用性、安全性与可靠性。

8.1.1 什么是 AI 可观测

AI 可观测是一系列能够让工程师全面洞察基于大型语言模型构建的应用的实践与工具。它超越了传统的监控，不仅追踪系统性能，更深入探究模型“在做什么”以及“为什么这么做”。这种全面的方法对于确保 AI 应用的健康、性能和安全至关重要。借助适当的可观测工具和产品，团队可以观测从响应质量、延迟到可能预示着错误或滥用的异常行为等一切情况。

AI 应用具有非确定性，即便是相同的提示，在不同次的运行中也可能产生不同的输出。如果没有可观测，在发生幻觉虚假错误等严重问题时根本无从查起。相反，一个好用的可观测工具会记录每一次的提示与响应、追踪使用模式，并标记异常。这为工程师提供了非常宝贵的洞察，以修复不准确之处、优化性能并修复安全风险。简而言之，可观测是确保 AI 应用高效稳定安全运行所必不可少的基础能力。

8.1.2 可观测 vs. 监控：从“是什么”到“为什么”

在 AI 应用的背景下，区分“可观测”与“监控”非常重要。

- **监控：关注“什么”**

监控通常聚焦于“发生了什么”。它追踪关键性能指标（KPIs）和系统健康状况，以确保服务正常运行且响应迅速。例如，监控会记录 API 响应时间、错误率、请求吞吐量和 Token 使用量等指标。一旦某个指标超出阈值（比如延迟飙升或错误率增加），监控工具会发出警报或在仪表盘上显示，以便工程师迅速作出反应。监控回答的问题是：“AI 应用当前是否有问题？”

- **可观测性：探究“为什么”**

可观测则深入挖掘这些指标背后的“为什么”。一个可观测解决方案，不仅会像监控一样采集性能指标，还会将这些指标与系统内丰富的日志、事件和链路信息（Trace）关联起来。这意味着 AI 应用的每一次请求，都可以被弯针的追溯，包括调用工具的出入参、发送给大模型的具体提示词、任何中间步骤（如调用数据库或其他 API），以及最终收到的输出。所以，当具体发生错误的时候，可观测工具就可以提供极其丰富的排查数据和上下文，帮助定位问题的根因。例如，如果一个聊天机器人的回答不正确且耗时过长，监控可能只会显示“错误率上升”或“延迟过高”，但可观测则能揭示为什么：可能是一次长时间的 RAG 请求最终得到了错误的召回，导致模型给出不准确的答案。

总结来说，监控告诉你“出问题了”。可观测则帮助你理解系统内部究竟发生了什么以及如何修复。 两者都是必需的，监控提供早期预警，而可观测提供深度诊断。对于 AI 应用而言，后者所提供的全面洞察尤为关键。

8.1.3 AI 可观测应对的核心挑战

AI 应用面临着一系列传统软件所没有的独特挑战，总结来讲有3大类：

- **性能与可靠性问题：**大模型是资源密集型的，延迟峰值和瓶颈时有发生。可观测将所有组件的数据关联起来，使工程师能够精确定位延迟的根源，是模型本身、外部 API 调用还是数据库查询。它还能追踪多步骤流程中的每一步，简化了复杂系统中的调试过程。
- **成本问题：**许多大模型服务按 Token 使用量收费，若无控制，成本可能意外飙升。可观测工具追踪每个请求的 Token 数、每日总用量等指标，当使用量出现异常高峰时发出警报，帮助团队在收到天价账单前优化提示或设置限制。
- **质量问题：**大模型的可能输出从训练数据中继承偏见或有害内容，也很有可能产生幻觉，导致输出的内容完全不符合预期，可观测通过提供评估等工具，针对采集的 AI 应用执行过程中各个阶段的输入输出，检测是否含有不当、不准确和危险的内容，通过自动分析和评分帮助工程师及时采取行动。

8.1.4 AI 可观测解决方案的关键能力

为应对上述挑战，一个高效的 AI 可观测解决方案应具备以下几项关键能力：

- **端到端全链路追踪：**提供端到端的日志采集和链路追踪，可视化展示请求在整个 AI 应用中的执行路径。支持对历史对话的灵活查询与筛选，以便于调试和改进。
- **全栈可观测：**包含应用、AI 网关、推理引擎可观测3个维度，观测内容有实时追踪响应延迟、请求吞吐量、Token 消耗，错误率和资源使用情况（如 CPU、内存、API 令牌），并能在指标异常时触发警报，帮助团队在影响用户前快速响应，同时有效监控成本。
- **自动化评估功能：**通过引入评估 Agent，对应用和模型的输入输出进行自动化的评估，检测幻觉、不一致性或答案质量下降等问题。有效的工具通常会集成评估模板，方便工程师快速的对常见的质量和安全问题进行评估。

第8章的第2-5节，我们将就端到端的链路和日志采集、全栈可观测进行展开，自动化评估在第9章中详细阐述。

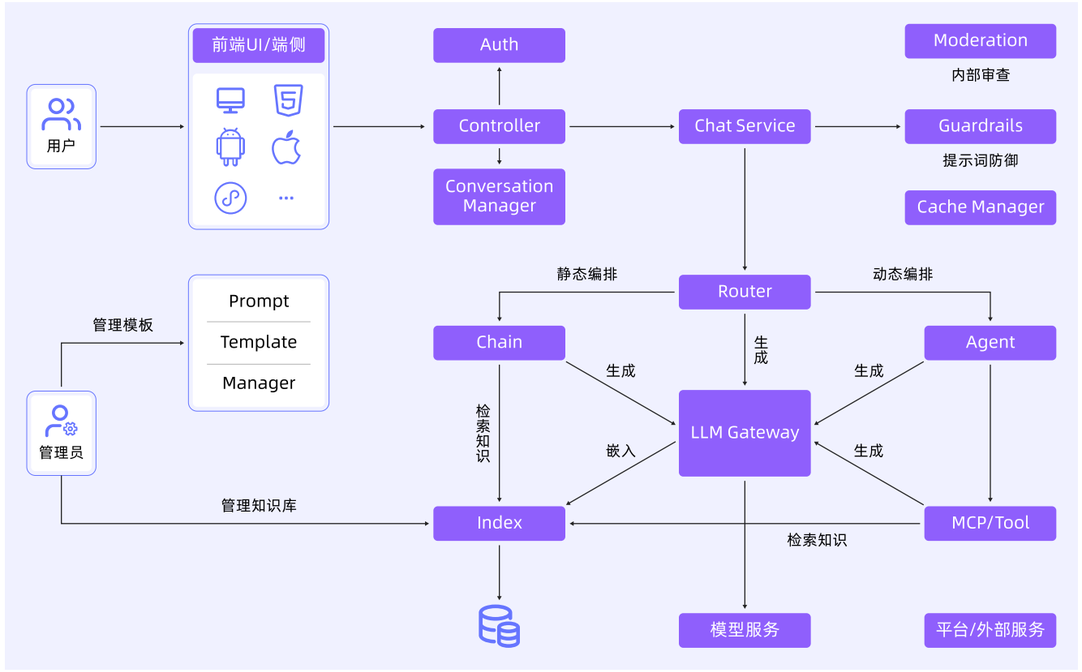
ALIBABA CLOUD

8.2 端到端全链路追踪

一个典型的 LLM 应用架构可能包含用户终端、认证模块、会话管理、对话服务、大模型路由、流程编排等。此外，模型推理应用还可能对接不同的大模型服务，依赖外部工具完成具体操作，通过向量数据库提供长期记忆，通过缓存降低 LLM 重复调用成本等。

为了应对 AI 应用链路的复杂性，保障系统服务 SLA 和终端用户体验，我们需要具备如下可观测能力：

- **标准化的数据语义规范：**比如规范 LLM Trace/Metrics 领域化语义，记录 Input、Output、Prompt、Token、TTFT、TPOT 等关键信息。
- **低成本高质量的数据采集：**比如通过 OpenTelemetry Agent 实现 LLM 模型应用与服务的无侵入数据采集。
- **端到端全链路追踪：**基于 OpenTelemetry W3C 协议，实现用户终端、AI 网关、模型应用、外部工具、模型服务的全链路 LLM Trace 串联，实现问题快速发现、定界与根因定位。



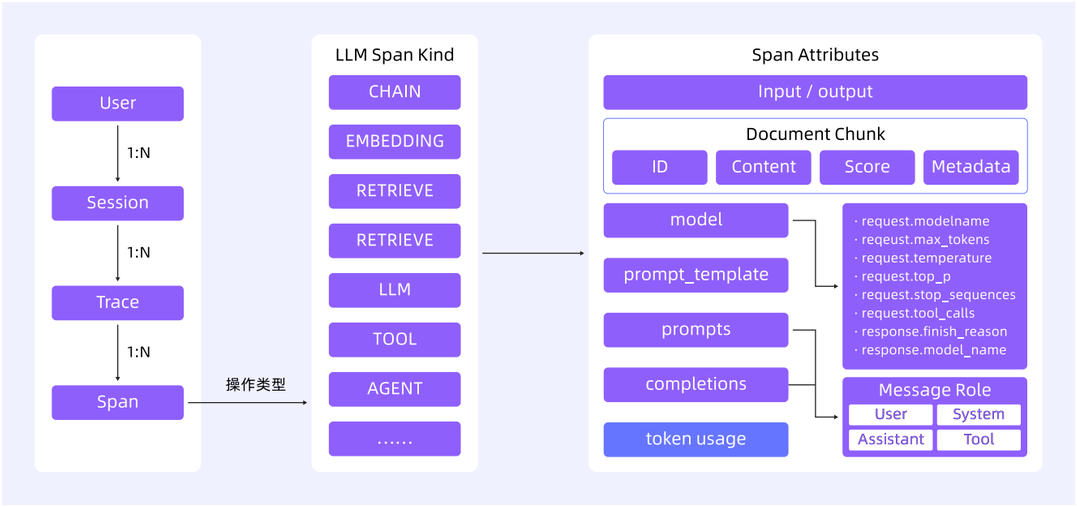
本节将介绍端到端全链路追踪的实现方式和核心技术。

8.2.1 端到端全链路追踪的实现方式

我们可以基于 OpenTelemetry 标准，通过 Trace 领域化语义增强、低成本高质量数据采集、标准化协议透传等方面的工作，最终实现了从用户终端到模型推理层的完整调用链路追踪。

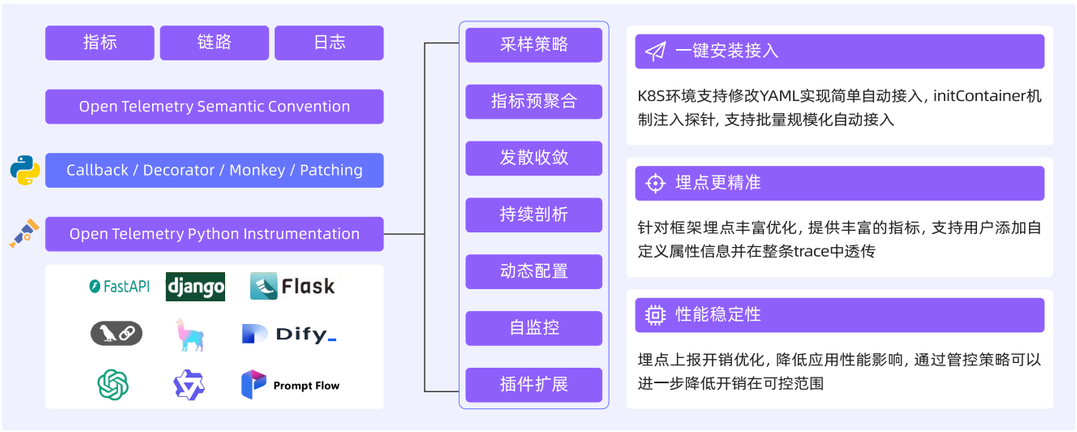
- **面向 AI 应用的领域化 Trace 语义**

以会话串联用户交互问答过程，以 Trace 承载应用全链路交互节点，定义领域化的操作语义、标准化存储以及可视化关键内容，比如 Input、Output、Prompt、Token、TTFT、TPOT 等字段，可用于模型性能分析、Token 成本分析等场景。



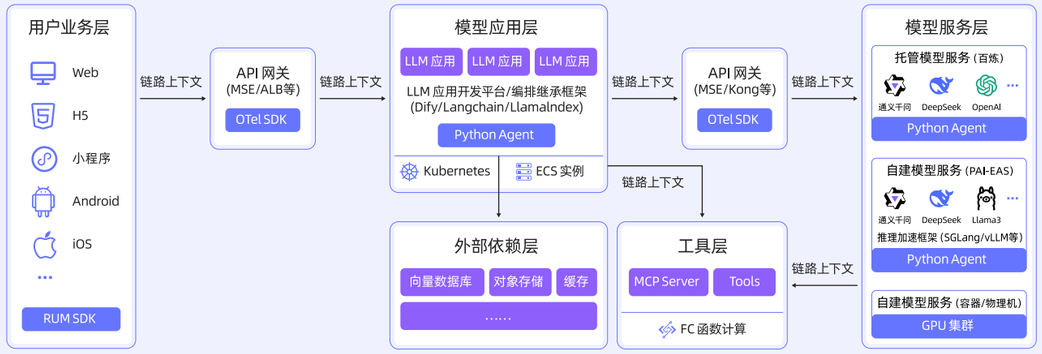
- **基于 OpenTelemetry 的高质量数据采集**

兼容 OpenTelemetry Python/Java Agent 等多客户端接入，增强大模型领域语义规范与数据采集，提供多种性能诊断数据，全方位自监控保障稳定高可用。



- **标准化协议**

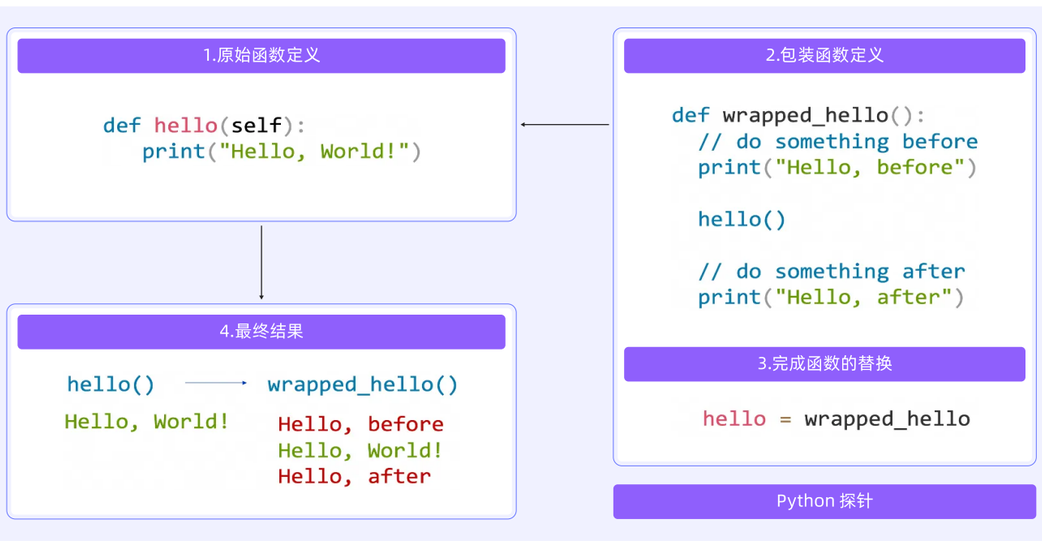
兼容 OpenTelemetry W3C 协议，实现跨语言、跨组件链路透传。



8.2.2 核心技术路径

1、链路插桩技术

- **Python 探针：**Python 语言提供一种装饰器模式，底层采用了一种 Monkey Patch 的机制，允许程序在运行时改变一个模块或者类的属性和方法，实现无侵入的可观测代码插桩逻辑。
  - 采用 monkey patch 实现无侵入埋点，支持LlamaIndex/Dify/LangChain等主流框架。
  - 自动采集 LLM 调用参数、Token 数、TTFT/TPOT 等指标。

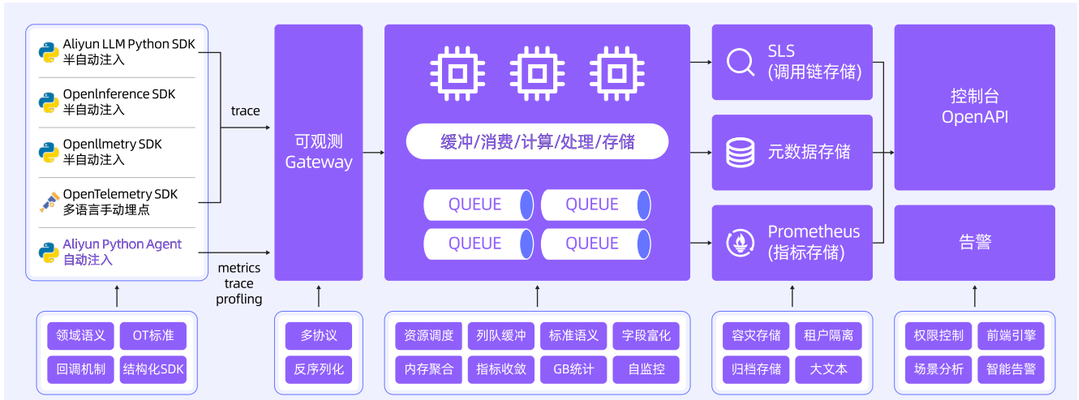


- **Java探针：**
  - 通过字节码增强技术拦截 Spring Boot 应用的模型调用链。
  - 支持 Dubbo/RPC 调用与 LLM 调用的关联分析。

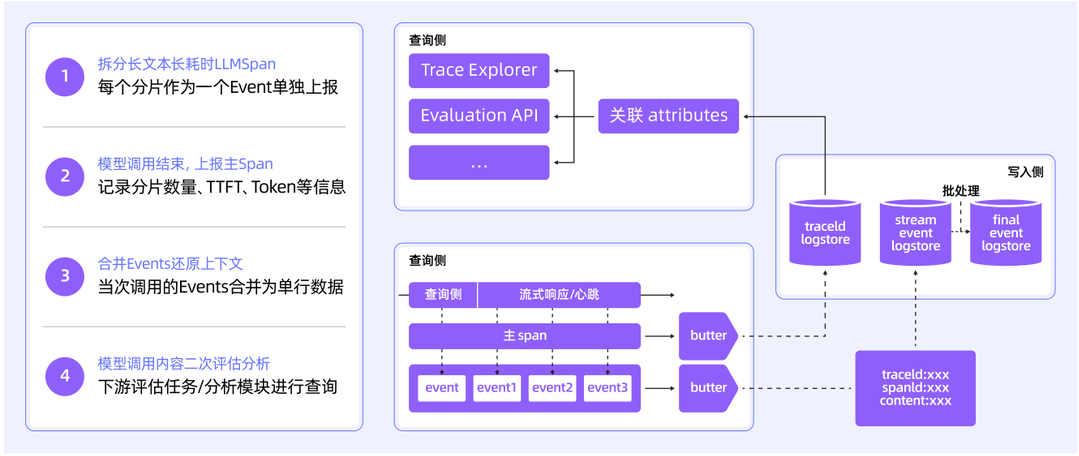
- **Go 探针：**
  - 通过编译时插桩技术自动针对常用的框架进行埋点，注入可观测数据采集的逻辑。
- **多语言兼容：**
  - 其他语言通过 OpenTelemetry 开源框架支持。

2、链路采集与加工

- **数据采集策略：**统一的数据处理链路，开放的可观测数据标准生态，线性化处理架构，海量数据稳定吞吐，标准化可观测模型和存储，构建观测数据全景。
  - Trace 直连上报：适用于用户终端、后端应用等场景。
  - 日志转 Trace：适用于网关等组件，通过日志解析生成链路数据（需包含 TraceId 上下文）。

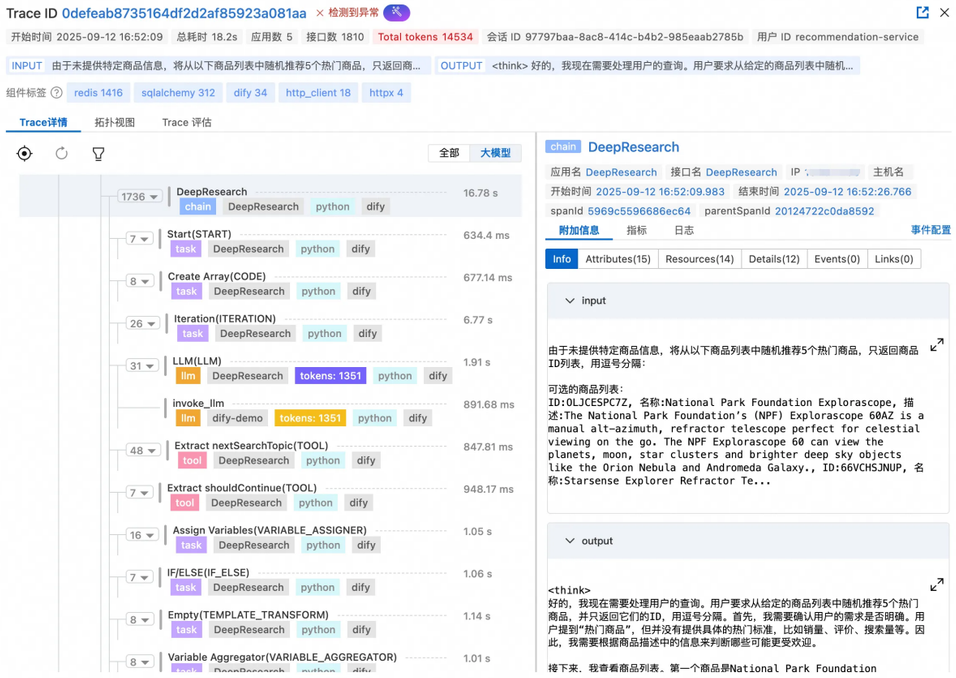


- **流式场景优化：**随着 SSE 协议在大模型推理场景的广泛应用，针对流式数据采用客户端分段采集+服务端合并还原，以平衡客户端侧性能与数据分析易用性。
  - 拆分长 Span 为 Event 分片，记录 TTFT/Token 数等元数据。
  - 模型调用结束时合并 Events 还原完整上下文。



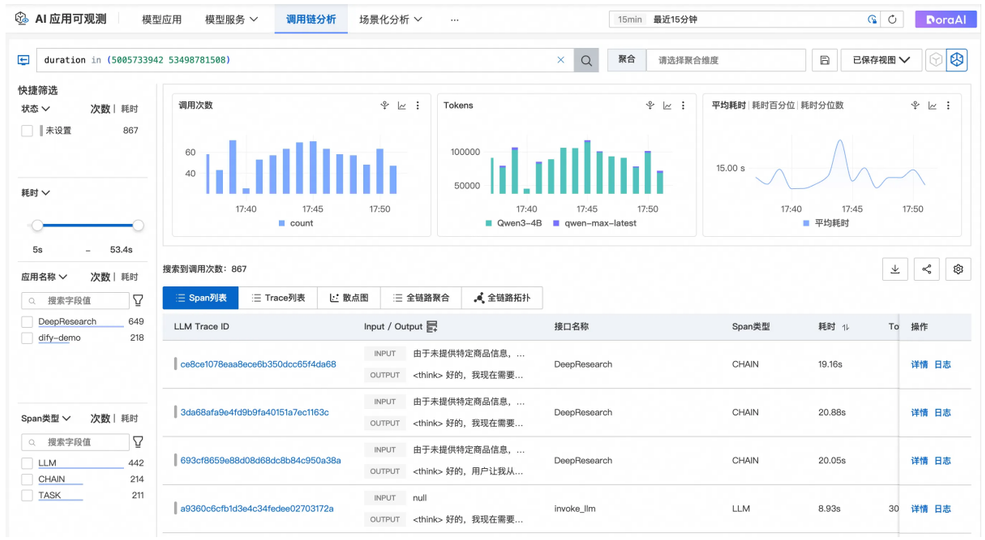
3、LLM Trace 查询与分析

- **全链路透视**：通过 Trace ID 串联用户请求路径，展示 LLM 调用输入/输出内容、Prompt 模板、模型参数。



- **智能诊断**：Trace 数据本身也可以基于大模型实现智能诊断，自动定位错慢瓶颈根因，并提供优化建议。

- 高级过滤：按状态（Success/Error）、耗时（>2s）、Span类型（LLM/MCP）筛选。



ALIBABA CLOUD

8.3 全栈可观测：应用可观测

上一节，我们讲了端到端全链路追踪的实现方式和核心技术路径，这样我们就构建了实现全栈可观测的基建。接下去3节，将分别从应用、AI 网关、推理引擎3个维度分享全栈可观测的场景、能力和实践。

8.3.1 AI 原生应用开发的痛点

在开发 AI Agent 与 MCP 的实际应用中，开发者经常遇到的关键痛点：

- **工具选择盲区：**Agent 在面对复杂任务时，可能选择了不合适的工具来完成任务，导致执行效率低下或结果不准确。而且开发者缺乏可视化手段来分析 Agent 的决策过程和工具选择逻辑。
- **错误排查困难：**工具调用时参数格式错误、类型不匹配或必填参数缺失，错误信息往往隐藏在复杂的调用链中，难以快速定位问题根源。
- **Token 消耗黑洞：**Agent 与 MCP 工具的多轮交互容易产生意想不到的大量token消耗，缺乏实时的成本监控，可能导致预算超支。而且无法清晰了解不同 MCP 工具的 Token 消耗占比，难以对成本进行精准优化。
- **循环调用陷阱：**Agent 可能陷入反复调用相同工具的循环中，缺乏合适手段及时发现和中断这种异常行为模式，造成资源浪费。

8.3.2 AI 原生应用可观测需要具备哪些能力

我们认为，AI 应用运行过程中接入全链路可观测，应具备以下能力：

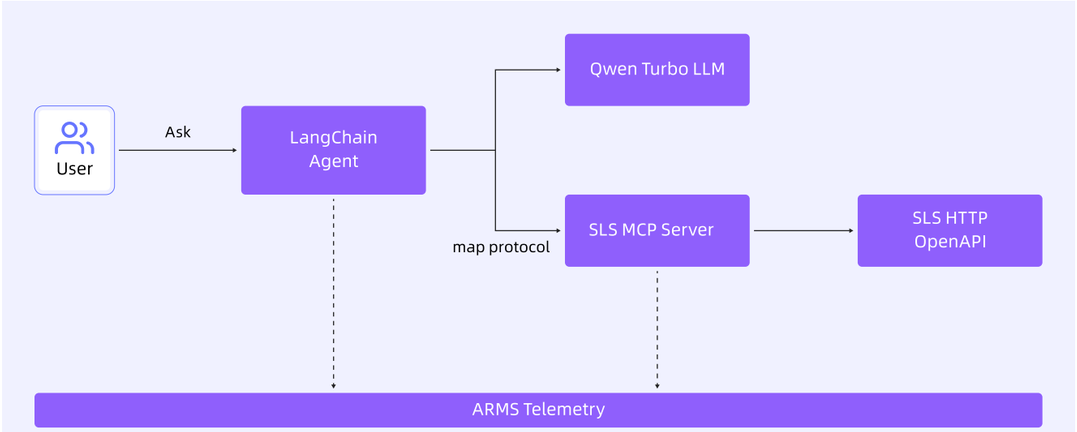
- **零代码接入：**无需修改任何代码，开箱即用的完整的 AI 应用监控能力。
- **可视化工具选择过程：**深度集成 MCP 协议，可视化 AI Agent 的工具选择和调用过程。
- **精准故障定位：**当系统出现异常时，可通过链路信息快速锁定问题根源。
- **Token 成本分析：**提供大模型 Token 使用量的精确监控和成本关联数据。
- **端到端链路追踪：**从用户查询到 Agent、MCP 调用的完整调用链路展示，快速定位异常调用模式。

8.3.3 演示场景架构

下面将构建一个智能日志分析助手，展示如何监控基于 LangChain 的 agent 以及其调用的 MCP 服务器。

演示中，我们使用目前非常流行的 LangChain 框架构建了一个 Agent，用户会向 Agent 发起提问。Agent 通过 Qwen Turbo 大模型的能力，自主规划解决问题的行动计划，并分步骤执行。在执行过程中 Agent 会通过 MCP 协议访问一个 MCP 服务器，本文以 SLS MCP 服务器为例，SLS MCP 服务器通过 SLS HTTP OpenAPI 访问日志服务的接口，完成日志分析请求。

请求中 LangChain Agent 与 MCP 服务器产生的观测数据会自动采集到可观测平台之中（本文以阿里云应用实时监控服务 ARMS 为例），我们可以对采集到的数据做进一步的分析。



8.3.4 场景演示

1、启动 SLS MCP 服务器

在终端中执行以下命令，在本地启动阿里云可观测 MCP 服务器：

```
aliyun-instrument python -m mcp_server_aliyun_observability \
 --transport sse \
 --access-key-id <你的阿里云AccessKey ID> \
 --access-key-secret <你的阿里云AccessKey Secret>
```

ALIBABA CLOUD

2、启动 Langchain Agent 程序

在这个简单的 demo 程序中，我们创建了一个 Agent，我们向 Agent 提了两个问题，该 Agent 会使用 SLS MCP 服务来处理我们的日志分析请求。

程序将会输出如下结果：

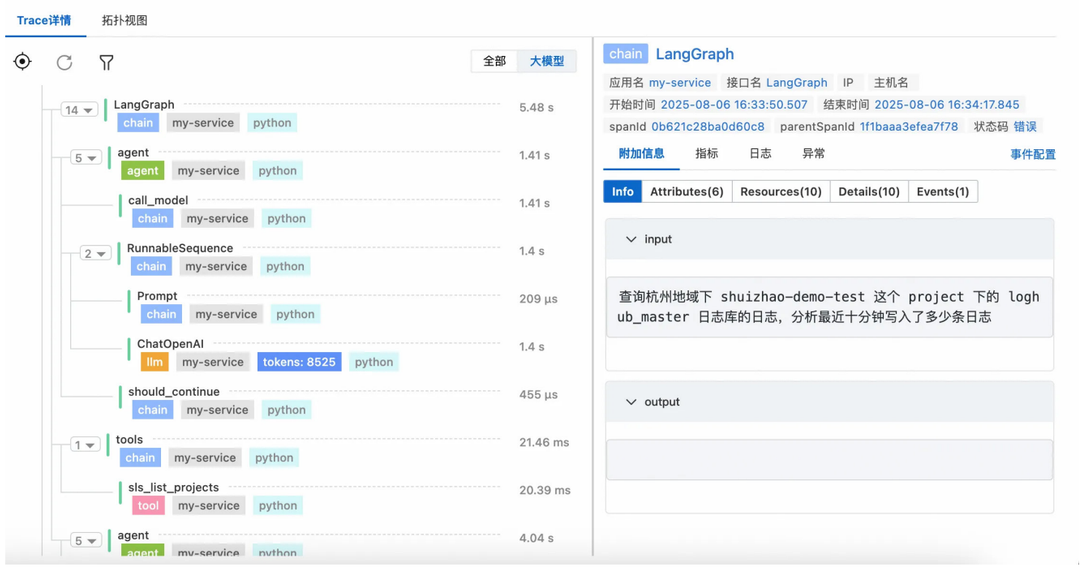
```
我要查询我在杭州地域的project的列表，取前10个


以下是您在杭州地域（cn-hangzhou）的前10个项目列表：

1. sls-aysls-pub-cn-hangzhou-b-xxxx
2. e2e-test-xxxx
3. workspace-xxxx
4. ws-xxxx (描述: Create by o11y)
5. clickbench8 (描述: clickbench8)
6. clickbench16 (描述: clickbench16)
7. clickbench (描述: clickbench)
8. job-e2e-project-xxxx
9. job-e2e-project-xxxx
10. job-e2e-project-xxxx
如果需要查询更多项目，您可以提供项目的关键词进行模糊搜索。


查询杭州地域下 shuizhao-demo-test 这个 project 下的 loghub_master 日志库的日志，分析最近十分钟写入了多少条日志

在最近的十分钟内，`shuizhao-demo-test` 项目下的 `loghub_master` 日志库共写入了 **8641 条日志**。
```

上面的演示比较简单，只是展示 MCP 与 AI Agent 集成的效果，更多MCP tools可以参考 alibabacloud-observability-mcp-server。

3、Agent观测

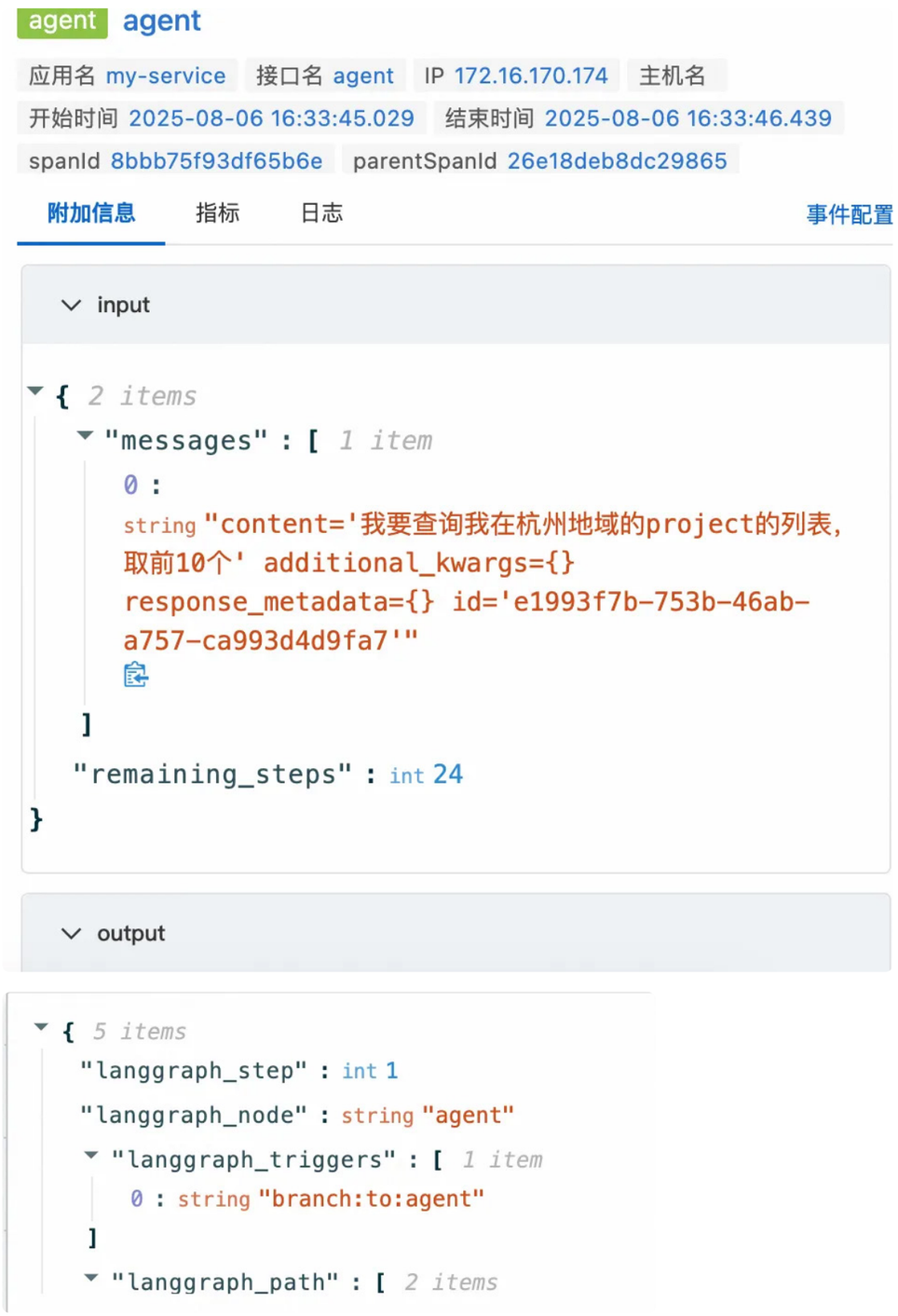
打开可观测平台（本文以阿里云 ARMS 为例），找到接口名称为 LangGraph 的 Span 列表。

打开其中的一条 Trace 详情：

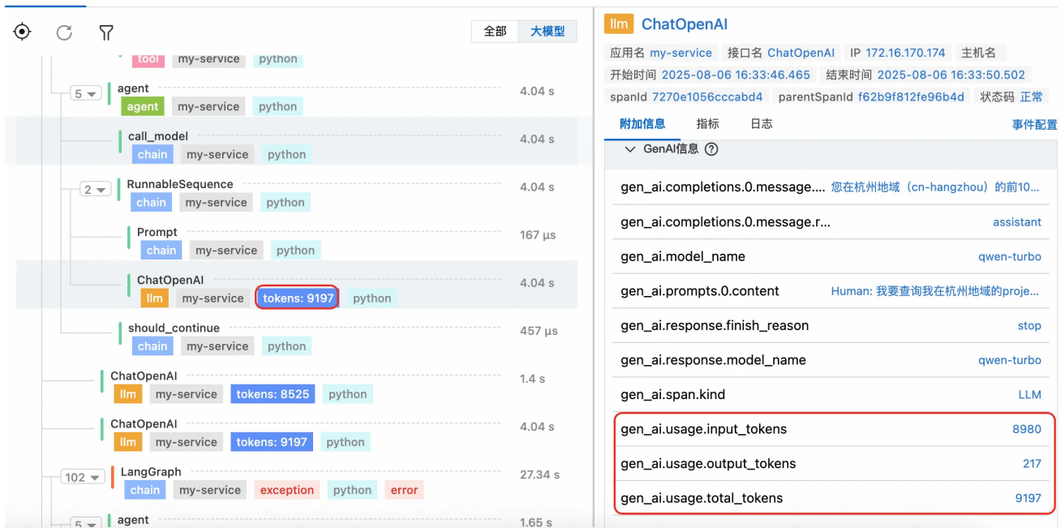


可以看到基于 LangChain 框架构建的 Agent 的详细执行过程，包括调用 LLM 和 MCP Server 以及最终的输入和输出：

针对这个 Agent，可以看到 Agent 执行的每一步的信息以及在 LangGraph 中是第几步：

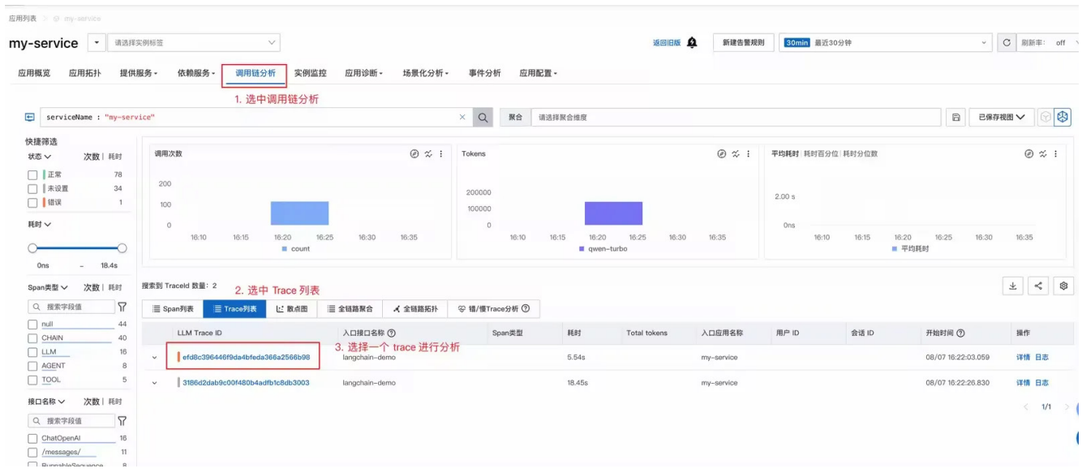


针对 LLM 的调用，也详细记录了 Token 的消耗，包括 Input Token和 Output Token，这样可以有效的发现 Token 的异常使用。

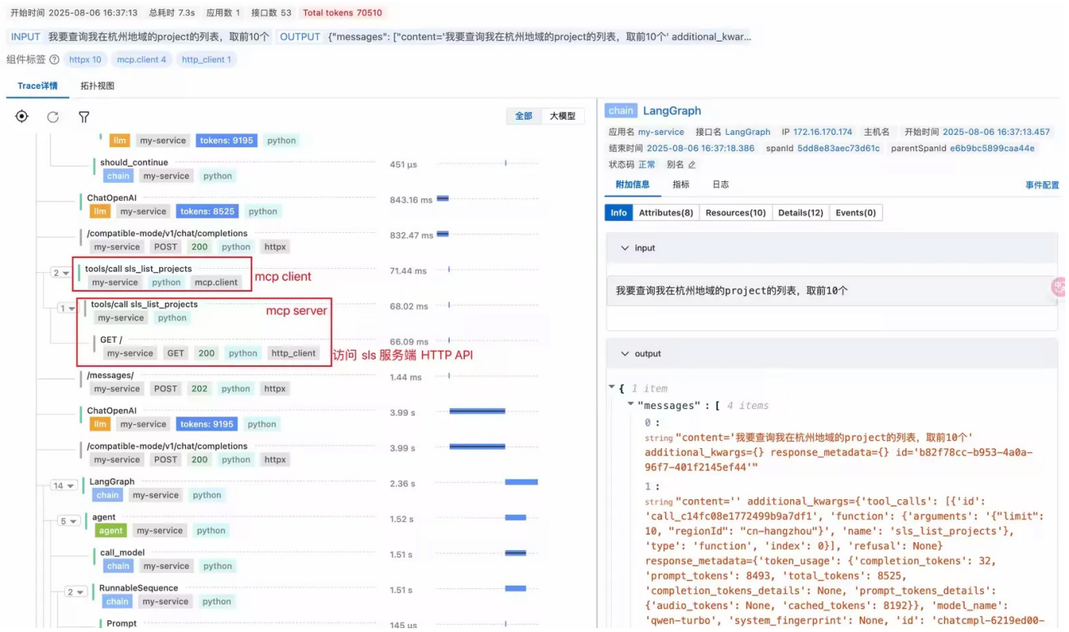


4、MCP 观测

现在让我们看看如何监控整个 MCP 调用链路。同样在控制台找到对应的应用，选中调用链分析 Trace 列表，选择一个 Trace 进行调用分析。

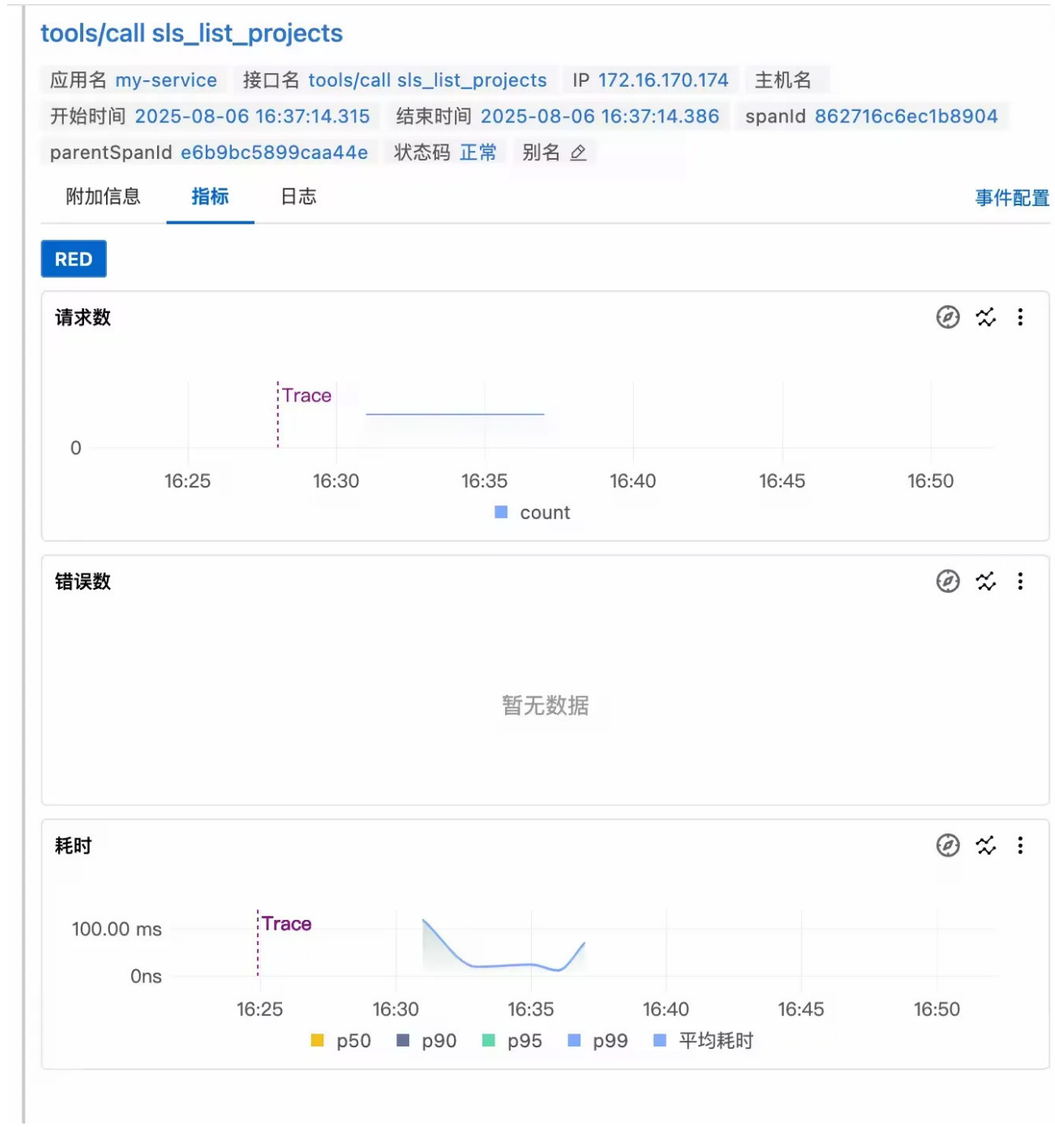


这里会展示一个完整的调用过程追踪信息。我们可以看到 AI Agent 调用了 MCP Client 来执行对日志服务 API 的访问，MCP Client 请求被发送到 MCP Server，MCP Server 通过 HTTP API 访问日志服务，总耗时 66ms。



进一步点击该 Span，我们可以看到更多详细信息，比如服务地址、请求协议、请求参数、返回 Body 大小等。

切换到指标选项卡，还可以查看该请求相关联的指标信息，可以进一步做吞吐量、延迟等指标的分析。



本节展示了通过常见的 AI Agent 框架搭建 AI 原生应用遇到的典型问题以及使用可观测工具如何解决上述问题，最后通过一个基于 LangChain 的 Agent 调用本地 MCP Server 的例子展示了如何使用应用可观测能力 Agent 和 MCP 调用的全流程进行全栈监控。下一节，我们将讲述 AI 网关的可观测。

8.4 全栈可观测：AI 网关可观测

AI 网关作为统一接入与治理中枢，承载着模型路由、鉴权、限流、缓存等关键能力。其运行状态直接影响 AI 应用的质量与效率。构建围绕 AI 网关的可观测体系，实现对请求流量、资源消耗、安全风险与治理策略的全面监控，是保障 AI 能力可靠输出的核心基础。

8.4.1 观测场景：AI 组件的多维可观测需求

在AI 组件的可观测性并非单一维度的性能监控，而是涵盖性能、资源、安全、成本、治理等多个层面的综合能力体系。以下从5个典型观测场景出发，全面揭示企业在使用 AI 网关过程中所面临的可观测需求。

1、性能与稳定性监控：保障 AI 应用的高可用

AI 应用的用户体验高度依赖于模型响应的及时性与稳定性。然而，大模型服务普遍存在响应延迟高、成功率波动大等问题。例如，商用模型在高峰期可能因限流导致请求失败，自建模型可能因 GPU 资源不足而出现超时。因此，企业需要实时掌握 AI 服务的性能表现。

通过 AI 网关，可观测性系统可采集以下关键性能指标：

- **QPS（Queries Per Second）：**每秒处理的请求数量，反映系统负载。
- **请求成功率：**成功返回结果的请求占比，用于评估服务健康度。
- **响应时间（RT）：**包括非流式响应的整体延迟和流式响应的首包延迟（Time to First Token），直接影响用户体验。
- **流式与非流式请求分布：**区分不同调用模式的性能特征。

这些指标不仅支持实时监控，还可用于设置告警规则。例如，当某 API 的请求成功率连续5分钟低于95%时，自动触发告警通知运维团队介入排查。

2、资源消耗与成本分析：实现精细化成本管控

大模型调用的成本主要由输入和输出 Token 数量决定，而不同模型的单价差异巨大。例如，GPT-4 Turbo 的 Token 价格远高于 Qwen-Max 或自建 Llama3 模型。若缺乏有效的资源监控机制，企业极易因个别应用的高频调用导致成本失控。

AI 网关通过统一代理所有模型请求，能够精确统计每个 API、每个消费者（Consumer）、每个模型的 Token 消耗情况：

- **Token 消耗数/s：**实时监控每秒输入、输出及总 Token 消耗速率。
- **按模型维度的 Token 使用统计：**识别高成本模型的使用频率，辅助模型选型优化。
- **按消费者维度的 Token 使用统计：**实现“谁使用、谁负责”的成本分摊机制，支持预算控制与配额管理。

此外，结合缓存命中率数据，企业还可评估缓存策略对成本的节省效果。例如，若某知识问答 API 的缓存命中率达到60%，意味着近三分之二的请求无需调用昂贵的大模型，显著降低整体成本。

3、安全与合规审计：防范数据泄露与内容风险

AI 应用的 SaaS 化特性带来了数据安全与合规风险。企业在调用外部模型时，可能无意中将敏感信息（如客户数据、内部文档）传入模型，存在泄露风险。同时，模型生成的内容也可能包含涉政、色情、违法等违规信息，影响企业声誉。

AI 网关内置内容安全审核能力，可在请求和响应两个方向进行双重防护。可观测系统需记录以下安全相关数据：

- **内容安全拦截日志：**记录被拦截的请求内容、风险类型（如涉政、敏感词）、消费者身份等。
- **风险类型统计：**按风险类别（政治、暴力、广告等）进行聚合分析，识别高频风险点。
- **风险消费者统计：**定位频繁触发安全规则的应用或用户，便于针对性治理。

这些数据不仅用于实时防护，还可作为合规审计的重要依据，满足 GDPR、网络安全法等法规要求。

4、治理策略执行追踪：确保限流、缓存、Fallback 有效落地

AI 网关的核心价值之一是提供统一的治理能力，包括 Token 级限流、语义缓存、多模型 Fallback 等。然而，这些策略是否真正生效，需要可观测性系统提供透明化的执行轨迹。

典型观测需求包括：

- **限流统计：**记录被限流的请求数量，分析限流触发原因（如某消费者突增流量）。
- **缓存命中情况：**展示缓存命中与未命中的请求数，评估缓存策略有效性。
- **Fallback执行路径：**当主模型不可用时，记录请求是否成功降级至备用模型（如从 DeepSeek API fallback 到自建 Qwen），并统计 fallback 成功率。

通过这些数据，企业可验证治理策略的实际效果，并持续优化配置。例如，若发现某 API 的 fallback 失败率较高，可能需增加备用模型的副本数或优化健康检查机制。

5、多租户与权限治理：实现调用者的精细化管理

在企业内部，多个团队或业务线可能共享同一AI网关资源。若缺乏有效的身份鉴权与访问控制机制，可能导致资源滥用、权限越界等问题。

AI 网关支持基于 API Key 或 JWT 的消费者鉴权，并为每个调用方分配独立身份。可观测系统需支持：

- **消费者身份识别**：记录每次请求的 Consumer ID，实现调用者可追溯。
- **消费者级指标统计**：按消费者维度展示 QPS、Token 消耗、错误率等指标。
- **异常消费者检测**：识别频繁触发限流、发送高风险请求的消费者，支持自动封禁或告警。

这种多租户视角的观测能力，有助于企业建立资源配额+行为审计的治理体系，确保 AI 资源的公平、合理使用。

8.4.2 观测实践：基于 AI 网关的可观测体系构建

要实现上述观测场景，企业需构建一套完整的可观测性实践体系。以下以 AI 网关为核心，介绍从数据采集、可视化监控到智能分析的全流程实践方法。

1、观测数据：统一日志与指标

AI 网关作为所有AI请求的统一入口，具备天然的数据采集优势。其可观测性体系依赖于两大核心组件：

- **指标采集（Metrics）**：通过集成 Prometheus、云厂商监控等监控系统，定期上报 QPS、RT、成功率、Token 消耗等聚合指标。
- **日志采集（Logging）**：利用日志系统，将每一条 AI 请求的完整上下文（包括请求头、Prompt、Response、状态码、模型名称、消费者ID、缓存命中情况等）写入日志库。

2、可视化监控：多维度仪表盘

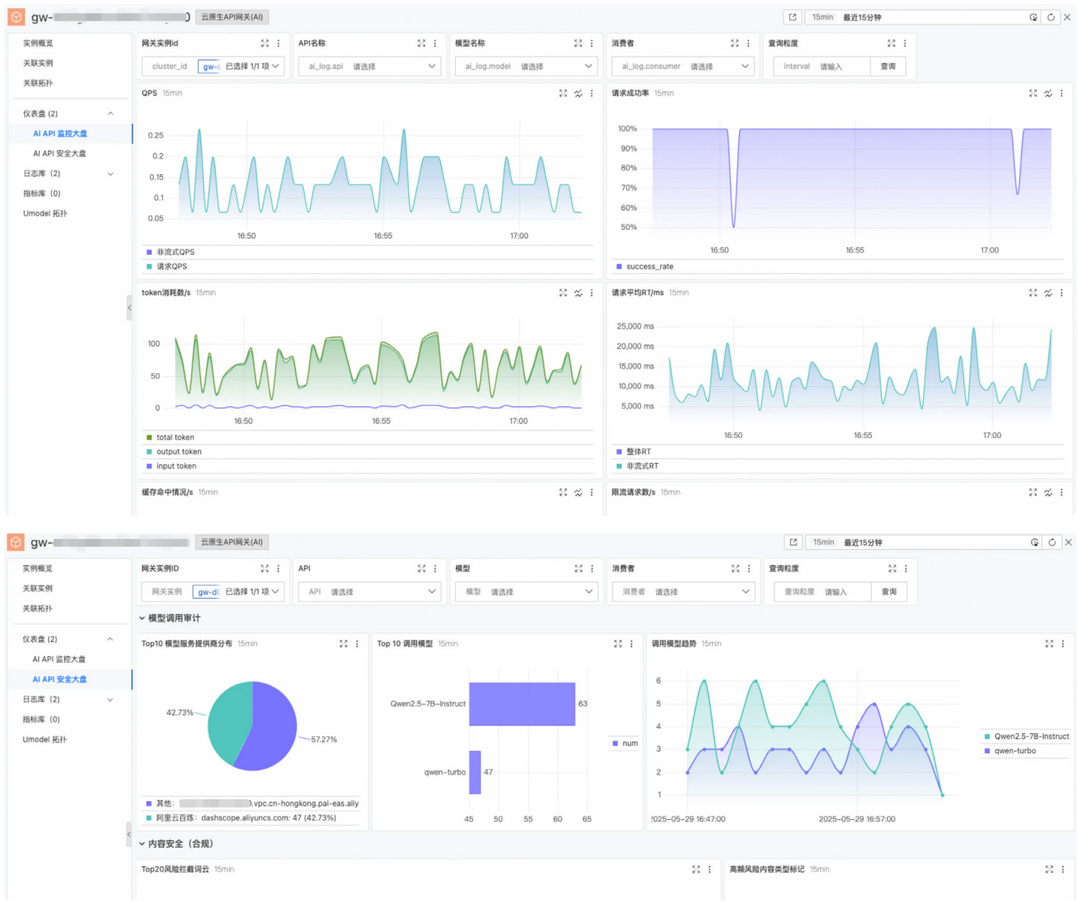
在 AI 组件的可观测体系建设中，可视化监控不仅是数据呈现的终端，更是决策支持的核心工具。通过构建多维度、分层级、可下钻的监控仪表盘，企业可实现从宏观态势感知到微观问题定位的闭环管理。

- **分层设计，聚焦核心场景**

仪表盘应遵循“总览–明细”分层结构：顶层展示全局关键指标（如整体 QPS、成功率、Token 消耗趋势），帮助快速掌握系统健康度；下层按网关实例、API、消费者等维度展开，支持精细化分析。

- **多维聚合，支撑业务视角**

指标需支持按模型、区域、消费者、时间粒度等多维度交叉分析，满足运维、成本、安全等不同角色的观测需求，实现技术指标与业务价值的对齐。



3、深度分析：基于日志查询与 SQL 分析

对于复杂问题排查，使用日志查询分析功能进行灵活检索。例如：

- 查询某消费者在过去1小时内的所有失败请求。
- 分析缓存命中对响应时间的影响。
- 统计高风险请求的来源。

通过 SQL 分析，企业可挖掘隐藏在海量日志中的模式与异常，实现从被动响应到主动洞察的转变。

4、智能告警与自动化响应

可观测性最终目标是提升系统稳定性。企业应基于关键指标设置智能告警规则，例如：

- 请求成功率 < 95% 持续5分钟 → 触发 P1 级告警。
- Token 消耗速率突增200% → 触发 P2 级告警。
- 连续10次 fallback 失败 → 触发运维介入。

告警可通过短信、邮件、钉钉机器人等方式通知相关人员，并可联动自动化脚本进行初步处置（如临时扩容、切换主备模型）。

5、成本优化与治理闭环

可观测数据不仅用于监控，更应驱动治理决策。企业可定期生成AI资源使用报告，包含：

- 各业务线的 Token 消耗排名。
- 高成本模型使用占比。
- 缓存节省成本估算。
- 安全事件统计。

基于报告，制定优化策略：

- 对高消耗 API 实施配额限制。
- 推动低价值场景使用低成本模型。
- 优化 Prompt 设计以减少输入 Token。
- 扩大缓存覆盖范围。

最终形成“监控 → 分析 → 优化 → 再监控”的治理闭环。

AI 网关作为 AI 组件可观测体系的核心载体，正在重新定义企业对 AI 服务的管理方式。它不仅解决了多模型集成、安全合规、性能稳定等基础问题，更通过全面的可观测能力，为企业提供了数据驱动的决策支持。下一节，我们将围绕推理引擎的可观测进行展开。

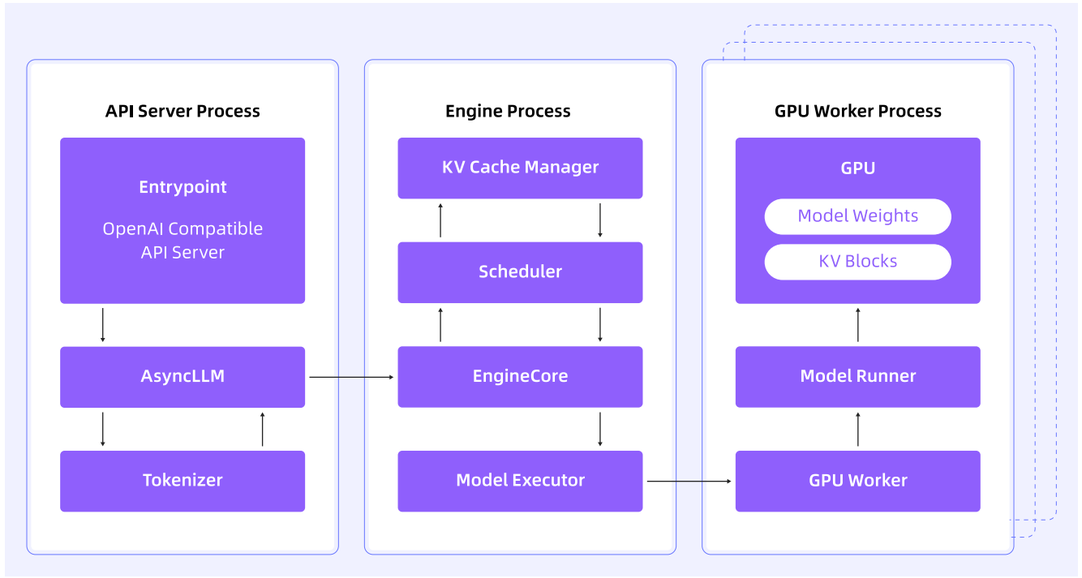
ALIBABA CLOUD

8.5 全栈可观测：推理引擎可观测

LLM 推理引擎是为了运行 LLM 而设计的专用软件，它接收用户的提示词并生成连贯的、人类可读的文本或代码响应。它管理着一系列复杂的流程，包括将输入转换为 LLM 能够理解的格式，运行模型逐个生成后续 Token，并将生成的 Token 转换回人类可读的输出。它是 AI 服务基础架构中的关键组件，是 AI 算法和软硬件系统之间的桥梁，其主要功能包括优化 LLM 的性能以确保快速准确的推理、管理 GPU 内存等硬件资源、提供分布式和可扩展能力等。常见的推理引擎有 vLLM、SGLang 等。

8.5.1 推理引擎需要观测什么

这一节我们以 vLLM 为例，介绍推理可观测。vLLM 是一款开源 LLM 推理和服务引擎，最初由加州大学伯克利分校的 Sky Computing Lab 开发，它采用了一种名为 Paged Attention 的全新内存分配算法。vLLM 是一个快速易用的 LLM 推理和服务库，现已发展成为一个由社区驱动的项目，得到了学术界和工业界的共同贡献。vLLM 的整体架构如下图所示：



图片来源：https://www.ubicloud.com/blog/life-of-an-inference-request-vllm-v1

vLLM 由 API Server、Engine、GPU Worker组成，它们可以运行在一个进程内，也可以多进程或者多机器分布式部署。其中 API Server 负责 HTTP 服务并对接收到的请求进行分词（Tokenization）。Engine 负责处理核心的 KV Cache 管理、调度等。GPU Worker则负责在GPU 上运行 LLM，生成推理的结果。

整个流程始于 API Server，它接收推理请求，并将请求传递给 AsyncLLM，后者对请求进行 Tokenization 并将其发送到 EngineCore。在 EngineCore 中，调度器会批量处理请求，并将它们发送到 GPU Worker 上执行。调度器里面有运行队列和等待队列，并通过算法让 GPU 的利用率最大化。Model Executor 与 Model Runner 协作，在 GPU 上加载 LLM 并通过 Attention layers 运行 Forward passes。最后，LLM 生成的 Tokens 将通过 HTTP 服务 传输回客户端。每个组件在这个请求执行的生命周期中都扮演着独特的角色，它们协同起来让 vLLM 达到了先进的服务性能。

我们看到 vLLM 是包含多个组件的复杂系统，其可观测性对于帮助开发人员和工程师监控性能、检测问题并有效地优化系统非常重要，尤其是在企业环境和云上。我们将推理引擎中一些常见的可观测项整理成了下面的表格，并挑选了一些比较有用的指标进行介绍。

我们可以在 Trace 中为 API Server、模型输入输出和推理过程分别创建 Span，将重要的可观测项以 Attribute 的方式记录到 Span 中。对于 API Server，我们可以对 HTTP 服务进行观测，记录请求的路径、来源IP端口、以及 HTTP 协议中的一些细节，也可以记录 HTTP 服务的整体耗时等。对于 Engine，我们可以记录推理请求的入参和出参，入参包括模型名称、提示词、Temperature、Top_p 等，出参包括生成的内容等。

我们也可以对请求执行过程进行观测。我们把每个请求执行的过程划分为 Prefill 和 Decode 两个阶段，分别记录这两个阶段的耗时。在 Prefill 阶段，推理引擎处理请求的提示词，计算 Transformer Attention，并将每个Token和每个层的注意力机制的键 (K) 值 (V) 张量存储到 KV 缓存块中。Prefill 运算量较大，Prefill 完成后才能生成第一个 Token。在 Decode 阶段，推理引擎为请求生成下一个输出 Token 或中间层的输出张量。它会重用已处理 Token 的 KV Cache，并根据最新的 Token 计算 Query (Q) 张量。Decode 阶段的性能决定了生成 Token 的速度。除了 Prefill 和 Decode，请求也可能在调度器中等待被执行，等待的时间也会影响用户体验，可以记录这个时间。

除此以外，对于推理引擎的一些内部状态，可以单独上报到 Metrics 系统中。这些指标总数可能有数十项，可以用来判断推理引擎中各组件的运行情况。比如请求数量超出了推理引擎的处理能力，则会在调度器中进行排队等待，影响用户体验。再比如 KV Cache 使用率过高，可能导致推理性能下降。

以下是我们罗列的 API Server、模型输入输出、推理过程、推理引擎状态4个维度常见的观测项和含义。

	观测项	含义	示例
API Server	路径	请求的路径。	/v1/completions
	状态码	请求的处理状态。	200/400
	客户端	客户端的类别。	Chrome/HTTP Client
	耗时	请求的处理时间。	1 s
模型输入输出	提示词	用户对模型的输入。	杭州哪里好玩?
	响应方式	是否以流的方式发送生成的 Token。	Stream/None Stream
	模型名称	被使用的 LLM 名称。	Qwen/Qwen2.5-0.5B
	最大 Token 数	模型生成的 Token 数量的上限，数量越大生成的时间越长。	1000
	温度	控制生成文本随机性和多样性的关键参数。温度越低，模型倾向于选择最高概率词，输出确定性高。温度越高，低概率词被选中几率增加，输出创意性强但可能不连贯，适合头脑风暴或诗歌创作。	0 0.5 0.9
	Top-K	模型在生成每个词时，仅保留概率最高的前K个候选词，并从中随机采样。例如，当K=3时，模型会从概率排名前三的词中随机选择一个作为输出。	3 5
	Top-P	根据概率累积阈值P动态选择候选词。例如，当P=0.9时，模型会按概率从高到低累加，直到总和≥0.9，仅保留这部分词进行随机采样。	0.5 0.9
	N	指定从生成的文本中返回的建议答案数量。	1 3

推理过程	E2E 时间	推理总时间，从接受到请求开始计算，直到推理完成。	1 s
	首 Token 时间	从向模型输入 Prompt 开始到模型生成第一个输出 token 所花费的时间。反映模型的初始响应速度，对于实时交互式应用非常重要，较低的 TTFT 可以提高用户体验，使用户感觉模型响应迅速，约等于等待时间 + Prefill 时间。	1 s
	Prefill 时间	Prefill 阶段所耗费的时间。	1 s
	Decode 时间	Decode 阶段所耗费的时间。	1 s
	等待时间	请求在调度器中等待的时间。	1 s
	Token 间隔时间	模型在输出阶段 (Decode 阶段) 每个输出 token 的延时，衡量模型的生成速度。	1 s
推理引擎状态	运行中请求数	正在推理的请求数量。	2
	等待请求数	等待被调度去执行的请求数量。	2
	被抢占请求数	当 KV Cache 不足时，请求被调度出 GPU，释放对应 GPU 内存，以便为其它请求腾出 KV Cache 空间。	5
	KV Cache 使用率	KV Cache 使用率，较高的使用率可能导致性能下降。	90%
	总提示词 Token 数量	所有请求的提示词的 Token 总数。	1000
	总生成 Token 数量	所有请求生成的 Token 总数。	1000
	处理成功的请求数	已被成功处理的总请求数。	1000

8.5.2 推理引擎需可观测的实践

首 Token 时间（TTFT）对客户体验比较重要。TTFT 的影响因素较多，比如提示词长度、并发请求排队、KV Cache 使用率等。如果我们观测到比较大的 TTFT，要怎么去优化呢？

- 若 KV Cache 使用率较高，且 GPU 显存使用接近上限的时候，可能是（--gpu-memory-utilization）这个参数设置得比较低，建议向上调整，将更多的 GPU 显存分配给推理引擎去使用。
- 若推理引擎中的等待请求数较高，说明并发请求数较大，超过了推理引擎的处理能力，这时可以增加显卡来提升处理能力。
- 若提示词较长，也会导致较大的 TTFT，这种情况可以将提示词改写得更加简洁。
- 升级 vLLM 可能也是一个选择，vLLM v1 版本相比 v0 做了大量的设计改进，比如优化了引擎结构、更强的调度器等，对整体性能会有改善。

在 LLM 推理引擎可观测技术的加持下，结合分布式调用链系统，开发人员将会对 AI 系统获得切实可行的洞察，深入了解延迟问题，准确定位性能瓶颈，帮助我们在改善 AI 应用性能方面迈出重要一步，并确保我们的基础架构能够面向未来。

AI 评估

AI Evaluation

CHAPTER

09

基于评估降低 AI 应用的不确定性

AI 评估体系

基于 LLM 的自动化评估

自动化评估落地实践

P259-P278

10

AI Security

P281-P302

11

Conclusion and Outlook

P305-P308

ALIBABA CLOUD

9.1 基于评估降低 AI 应用的不确定性

阿里云 CIO 蒋林泉在2025 AICon 全球人工智能开发与应用大会曾分享过：在落地大模型技术过程中总结过一套方法论，叫 RIDE，即 Reorganize（重组组织与生产关系）、Identify（识别业务痛点与 AI 机会）、Define（定义指标与运营体系）、和 Execute（推进数据建设与工程落地）。其中，Execute 提到了评估系统重要性的核心原因，即这一轮大模型最关键的区别在于：度量数据、评测没有标准答案。而既然这是没有标准答案的，就意味着成本最高，也成为落地的瓶颈。在 AI 领域里经常提到一个词叫“品味”，这里讲的“品味”，其实就是如何做评估的问题。

9.1.1 从确定性到不确定性

在传统的软件研发中，测试是开发过程中必不可少的一环，测试保障了输入和输出的确定性，以及向前兼容性，是软件质量保障的基础。测试覆盖率和准确率是评价质量的指标，而准确率必须保持在100%的水平。

- 传统软件在很大程度上是确定性的。给定相同的输入，系统将始终产生相同的输出。其故障模式，即“bug”，通常是离散的、可复现的，并且可以通过修改特定的代码行来修复。
- AI 应用，其行为本质上是非确定性和概率性的。表现出统计性的、上下文相关的故障模式和不可预测的涌现行为，这意味着对于相同的输入，它们可能会产生不同的输出。

此外，模型发布的惊人速度（仅2024年，就发布了80多个 LLM）和巨额投资（预计到2028年，将超过6320亿美元），给建立新的评估和治理范式带来了前所未有的紧迫性。传统的 QA 流程，专为可预测的、基于规则的系统而设计，已无法充分应对这些由数据驱动的、自适应系统的挑战。即使在发布阶段进行了充分的测试，也无法保障上线后，面对各种输出稳定性问题。因此，评估不再仅仅是部署前的一个阶段，而是贯穿 AI 应用整个生命周期的持续性、战略性要务。

9.1.2 幻觉和不确定性的根因

AI 应用的非确定性源于其核心技术架构和训练方法。与传统软件不同，AI 应用本质上是概率性系统。它们的核心功能是基于从海量数据中学到的统计模式，来预测序列中的下一个词，而非真正意义上的理解 。这种固有的随机性既是其创造力的源泉，也是其不可靠性的根源，并导致了“幻觉”现象，即模型生成听起来合理但实际上不正确或无意义的输出 。

产生幻觉和不确定性的根本原因错综复杂：

- **数据导致的缺陷：**模型的知识完全受限于其训练数据。如果数据不完整、包含事实错误或反映了社会偏见，模型将会继承并放大这些缺陷 。它无法提供训练数据之外的信息，例如未来的事件或未曾接触过的私有数据 。
- **架构与建模的产物：**Transformer 架构及其训练过程本身就会引入不确定性。预测下一个词元的核心任务鼓励模型在信息不足时进行“猜测”。对训练数据的过拟合可能导致模型死记硬背而非泛化，同时，注意力机制的错误可能使其忽略提示中的关键部分 。
- **错位与不确定性：**模型可能拥有正确的知识，但由于与用户的具体指令不完全对齐，导致未能正确应用这些知识 。虽然幻觉通常与模型的不确定性有关，但研究表明，模型也可能在具有高置信度的情况下产生幻觉，这使得此类错误尤其隐蔽且难以检测 。

9.1.3 AI 评估的重要性

对组织而言，系统性评估是 AI 治理的基石，支撑风险识别、流程验证与模型持续优化，防止偏见、幻觉或失效决策造成运营与声誉损失。尤其在医疗、金融等高风险领域，缺乏本地化评估可能导致模型失效或加剧不平等，带来严重后果。

从服务客户上看，评估是构建信任的核心机制。负责任 AI 的实践，如公平性、透明度、可解释性与问责制，必须通过可度量的评估来实现。当 AI 参与关键决策时，利益相关者需理解其逻辑。评估提供实证支持，回应公众对偏见、错误与“黑箱”操作的担忧，增强客户与监管机构的信任，推动技术采纳。

从商业视角看，严格评估直接转化为竞争优势。经过验证的 AI 系统在准确性、效率与用户体验上表现更优，助力企业加速创新、控制成本、提升市场竞争力。评估成为筛选高价值项目、优化资源投入、实现 AI 投资回报的关键杠杆。

更深远地，AI 的非确定性催生了评估经济，这是一个由可观测性平台、持续监控、自动化评估与治理即服务构成的新生态。评估不再是阶段性任务，而是与开发并行的持续工程实践，标志着 AI 原生时代质量保障的新范式。

9.2 AI 评估体系

为了系统地理解和应用 AI 评估，必须首先建立一个清晰的分类框架。AI 评估并非一个单一的概念，而是由多种不同目标、方法和应用场景构成的复杂领域。本节将介绍评估方法的基础二分法，探讨从静态到动态评估的演进，并综合近期学术研究，勾勒出现代 AI 评估的多范式全景。

9.2.1 评估方法的基础二分法

理解 AI 评估可以从两个基本的维度开始：评估的对象（内在与外在），评估的执行者（自动化与人工）。

1、内在评估 vs. 外在评估

- **内在评估：**此方法侧重于孤立地评估模型输出的固有质量，而不考虑其在特定任务中的应用效果。评估的维度通常包括流畅性（Fluency）、连贯性（Coherence）、语法正确性以及事实准确性等。例如，评估一个语言模型生成的文本是否通顺、逻辑是否一致。内在评估的优势在于它能够提供对模型核心语言能力的细致洞察，并且可以在受控环境中进行。然而，其主要缺点是评估结果可能无法直接反映模型在真实世界应用中的实际效用。
- **外在评估：**与内在评估相反，外在评估通过衡量模型在特定下游任务或应用中的表现来评估其质量。例如，通过评估一个由AI驱动的邮件助手是否能有效提升用户的办公效率，来判断该 AI 模型的价值。这种方法的优势在于它能更准确地衡量模型在真实场景中的性能和商业价值。但其缺点是评估结果可能高度依赖于特定任务，泛化能力有限，并且搭建评估环境通常需要大量的资源和基础设施。

2、自动化评估 vs. 人工评估

- **自动化评估：**此方法利用计算指标（如用于机器翻译的 BLEU 分数或用于文本摘要的 ROUGE 分数）或利用其他 AI 模型（如 LLM-as-a-Judge）来对模型输出进行打分。自动化评估的主要优势是其高可扩展性、低成本和结果的一致性，使其成为模型迭代开发过程中快速反馈的理想选择。然而，它往往难以捕捉人类语言中的细微差别、主观感受（如幽默、创意）和复杂的上下文依赖。
- **人工评估：**此方法依赖人类评估员的判断来评估AI系统的输出质量，特别是在衡量帮助性、创造力、用户满意度等主观维度时。人工评估是评估质量的“黄金标准”，因为它能深刻理解语言的细微之处和文化背景。但其显而易见的缺点是成本高昂、耗时漫长、难以规模化，并且评估结果可能受到评估员主观偏见的影响。在实践中，最佳的评估策略通常是自动化和人工评估的混合模式，利用自动化的效率进行大规模初步筛选，并结合人工的深度洞察力对关键或模糊的案例进行精细评估。

9.2.2 从静态到动态评估的演进

随着 AI 模型能力的飞速发展，传统的静态评估方法逐渐暴露出其局限性，推动了评估范式向更具挑战性和真实性的动态评估演进。

- **静态基准：**静态基准由固定的、预先定义的数据集和任务组成，例如 GLUE、SuperGLUE 和 MMLU 等。这些基准在推动 AI 发展、标准化模型比较方面发挥了至关重要的作用。然而，它们的主要问题在于，随着模型能力的增强，模型可能会记住测试集中的答案，或者在基准上达到饱和状态，导致分数上的差异不再能真实反映模型能力的差距。
- **动态评估：**动态评估旨在模拟真实世界场景的复杂性和不可预测性，通过变化的变量、意外的输入和演进的上下文来测试系统。这种方法不再依赖固定的问答对，而是评估模型的泛化能力、适应性和鲁棒性。主要的动态评估方法包括：
  - 基于模拟的测试：创建虚拟环境，模拟真实世界的各种条件，以在不产生实际风险的情况下评估 AI 代理的性能。
  - 实时对抗性基准：建立一个平台，让不同的AI模型相互竞争或与人类进行对抗。例如，Kaggle 游戏竞技场通过战略游戏来评估模型的规划和适应能力，而聊天机器人竞技场则通过众包用户的投票来对聊天机器人的性能进行排名。这类评估的核心是测试模型的战略推理能力，而非记忆力。
  - 程序化生成：在测试时使用算法动态生成新的、唯一的测试用例，从而有效防止模型对特定测试集过拟合。

9.2.3 现代 AI 评估的多范式全景

当前的 AI 评估领域并非铁板一块，而是由多个并存的、有时甚至是相互隔离的研究范式所构我们可以识别出以下6个主要的评估范式：

- **基准测试范式：**这是最传统的范式，专注于通过标准化的数据集和指标（如 GLUE）对 AI 系统进行性能排序和比较，目标是推动技术进步和衡量模型能力。
- **“Evals”范式：**此范式主要关注 AI 系统的安全性和潜在危害。它通常采用对抗性测试和红队

演练等方法，主动探查模型的弱点和危险能力，旨在为模型的安全部署提供保证。

- **心理测量学范式：**借鉴心理学的测量科学，该范式旨在评估 AI 系统内在的、不可直接观测的潜变量，如推理能力或空间感知能力 。其目标是解释模型在不同任务上表现差异的根本原因，而不仅仅是报告性能分数。
- **人机交互范式：**此范式将 AI 系统置于与人类协作的真实情境中进行评估，重点关注可用性、用户满意度、人机协同效率以及用户信任度等。其目标是确保 AI 技术能够真正地增强人类的能力，并带来积极的用户体验。
- **形式化方法范式：**该范式运用数学和逻辑学的工具，对 AI 系统的某些属性（如鲁棒性、公平性）提供形式化的、可证明的保证。其目标是追求最高级别的可靠性和确定性，尤其适用于安全攸关系统。
- **社会技术范式：**此范式将 AI 系统视为嵌入在更广泛社会结构中的一部分进行分析，考察其对社区、组织和整个社会的系统性影响，包括对就业、公平和权力动态的改变。其目标是全面理解和引导 AI 技术的社会后果。

对组织而言，选择何种评估范式并非纯粹的技术决策，而是其战略优先级和风险偏好的直接体现。一家追求快速产品市场匹配的初创公司可能会优先采用人机交互范式，以优化用户体验和任务完成率。相反，一家开发前沿基础模型的顶尖实验室则会投入巨资于“Evals”范式，以探测和缓解可能存在的灾难性风险。这种视角将评估从一个简单的验证步骤，提升为一个与商业目标和企业价值观紧密结合的战略工具。一个组织的评估栈——即其选择、组合和优先排序不同评估范式的策略——深刻地揭示了该组织在性能、用户价值和风险规避之间的权衡。

更进一步看，这些评估范式的并存与隔离，也暴露了当前 AI 评估领域的一个核心挑战：没有任何一个单一范式能够全面覆盖性能、安全、可用性和社会影响等所有关键维度。这种鸿沟正是推动如 HELM（Holistic Evaluation of Language Models）和 NIST AI 风险管理框架等综合性、多维度框架出现的主要驱动力。

这些框架本身并非新的评估范式，而是旨在弥合现有范式之间差距的“元框架”。它们通过强制要求进行多指标、跨领域的评估，推动从业者采纳一种更全面、更负责任的视角，从而避免部署那些虽然在某一维度（如性能）表现出色，但在其他维度（如安全性或公平性）存在严重缺陷的系统。这标志着 AI 评估学科正在走向成熟，从孤立的单点测试迈向系统性的综合治理。

9.2.4 构建一个整体性的评估体系

一个强大而全面的 AI 评估体系是负责任创新的基石。它不仅仅是一系列测试，更是一个贯穿 AI 生命周期的结构化框架，旨在系统性地验证模型的性能、鲁棒性、公平性和合规性。本节将详细阐述构建这样一个体系所需的核心组件。

一个整体性的评估体系是一个由多个相互关联的组件构成的有机整体，每个组件都在AI生命周期的不同阶段发挥着关键作用。综合行业最佳实践，一个完整的评估体系应包含以8个核心组件：

- **性能指标：**这是评估的基础，用于量化模型完成其预定任务的效果。根据任务类型，指标可以包括准确率、精确率、召回率、F1 分数等。
- **鲁棒性与泛化测试：**此组件旨在评估模型在面对未曾见过的新数据或在受到对抗性干扰时的表现。测试方法包括压力测试、分布外数据测试和对抗性攻击模拟，以确保模型不仅是记住了训练数据，而是真正学会了解决问题。
- **偏见与公平性评估：**这是确保 AI 系统符合伦理要求的关键。该组件通过使用公平性指标（如人口统计均等、机会均等等），评估模型是否对不同的人口群体（如按种族、性别、年龄划分）产生系统性的歧视性结果。
- **可解释性与可解释性：**为了建立信任和实现问责，模型必须能够以人类可理解的方式解释其决策过程。此组件利用 LIME、SHAP 等技术，使模型的黑箱变得透明。
- **合规性与伦理考量：**AI 系统必须遵守日益增多的法律法规，如欧盟的《通用数据保护条例》（GDPR）和《人工智能法案》（AI Act）。此组件确保系统的开发和部署符合法律要求，并与组织的伦理准则保持一致。
- **持续监控与模型漂移检测：**模型部署后并非一劳永逸。由于现实世界的数据分布会随时间变化，模型的性能可能会下降，即“模型漂移”。此组件通过实时监控模型的输入、输出和性能指标，及时发现并应对性能衰退问题。
- **决策框架：**评估的最终目的是为了指导行动。该组件建立了一套清晰的流程，用于将评估结果转化为具体的决策，如批准部署、进行再训练、调整模型参数，或者在发现严重问题时进行回滚或下线。
- **自动化与人在回路：**为了在效率和质量之间取得平衡，评估体系应结合自动化工具和必要的人工监督。自动化评估可以处理大规模、重复性的测试，而人在回路机制则利用人类的专业知识和常识来处理自动化系统难以判断的模糊、复杂或高风险案例。

9.3 基于 LLM 的自动化评估

随着生成式 AI 能力的飞速发展，传统的评估方法已显得力不从心。为了应对这一挑战，一个名为 LLM-as-a-Judge 的前沿范式应运而生。

9.3.1 传统指标的局限性

传统的自动化评估指标，如用于机器翻译的 BLEU 和用于文本摘要的 ROUGE，其核心是基于词汇或短语的重叠度来计算分数。这种方法对于评估需要捕捉语义、风格、语气和创造力等细微差别的现代生成式AI模型来说，存在根本性的不足。一个模型生成的文本可能在措辞上与参考答案完全不同，但在语义上却更准确、更具洞察力。传统指标无法识别这种情况，甚至可能给予低分。

另一方面，虽然人工评估被视为评估质量的“黄金标准”，但其高昂的成本、漫长的周期和固有的主观性，使其难以适应 AI 技术快速迭代的开发节奏。这种评估能力的滞后，形成了一个严重的瓶颈，常常导致有潜力的 AI 项目陷入试点困境，无法有效验证和改进。因此，业界迫切需要一种既能理解语言的细微差别，又具备可扩展性和成本效益的评估新方法。

9.3.2 LLM-as-a-Judge 范式介绍

LLM-as-a-Judge 范式正是为了填补这一空白而出现的。它利用一个功能强大的大型语言模型（通常是前沿模型）来扮演裁判的角色，对另一个 AI 模型（或应用）的输出进行评分、排序或选择。这种方法巧妙地结合了自动化评估的可扩展性和人工评估的细致性。

- **工作机制：**其基本工作流程是，向裁判 LLM 提供一个精心设计的提示词。这个提示通常包含：
  · 被评估模型的输出。
  · 产生该输出的原始输入或问题。
  · 一套明确的评估标准或“评分指南”，用自然语言描述，例如“请评估以下回答的帮助性、事实准确性和礼貌程度”。裁判 LLM 随后会根据这些信息，生成一个分数、一个判断（如回答 A 优于回答 B）或一段详细的评估反馈。

- **核心应用场景：**
  - 数据标注：大规模、低成本地为数据集进行标注，合成检测数据，用于监督式微调或创建新的评估基准。
  - 实时验证：在应用中充当护栏，在输出返回给最终用户之前，实时检查其是否存在幻觉、违反政策或包含有害内容。
  - 为模型优化提供反馈：生成详细、可解释的反馈，指导模型的迭代改进。例如，其中一个 LLM 根据一套伦理原则来评估和修正另一个 LLM 的输出，从而实现模型的自我完善。

从一个更深层次的视角来看，LLM-as-a-Judge 范式能够将高级的、主观的人类偏好（通过自然语言评分指南表达）编译成一个可扩展、自动化且可重复执行的评估函数。这个过程将原本属于定性评估的艺术，转变为一门可以系统化实施的工程学科。其逻辑在于，该范式接收了抽象的、定性的输入（如评估回答的创造力），并将其转化为结构化的、定量的输出（如一个1到5的分数）。这种转化过程使得那些以往只能依赖昂贵且缓慢的人工评估才能衡量的复杂、主观标准，现在可以通过工程化的方式进行系统性评估。

9.3.3 LLM 裁判的评估模式

LLM-as-a-Judge 范式可以采用不同的模式进行评估，其中，成对比较通常能产生更可靠且与人类偏好更一致的结果。

- **逐点评估：**在这种模式下，裁判独立地评估单个模型输出，并根据预定义的量表（如1-5分制）给出一个绝对分数。这种评估可以是有参考的（即与一个标准答案进行比较）或无参考的（仅评估输出本身的质量）。
- **成对比较：**在这种模式下，裁判会同时看到两个来自不同模型的、匿名的输出，并被要求判断哪一个更好，或者两者是否相当。研究表明，这种相对判断的方式更符合人类的认知习惯，因此评估结果与人类偏好的一致性更高，也更稳定。这也是像聊天机器人竞技场这样的众包评估平台所采用的核心方法。

9.3.4 控制 LLM 裁判的负面影响

尽管 LLM-as-a-Judge 范式带来了革命性的进步，但它远非完美。将 AI 模型作为裁判引入了其自身的系统性偏见和局限性，如果不加以识别和缓解，可能会导致评估结果严重失真。以下将深入剖析这些自动化裁判的常见陷阱，系统地阐述位置偏见、冗长偏见和自我偏好偏见等问

题，并探讨其根本性的能力局限，最后提出一系列行之有效的缓解策略，以增强自动化评估的可靠性。

1、系统性偏见

研究已经证实，LLM 裁判在进行判断时会表现出多种类似人类的认知偏见，这些偏见会系统性地影响评估的公正性。

- **位置偏见：**在成对比较任务中，LLM 裁判常常表现出对排列在第一个位置的回答的偏好，无论其内容质量如何。这种先入为主的偏见可能会导致对模型性能的错误排序。
- **冗长偏见：**与人类通常偏爱简洁明了的回答不同，LLM 裁判倾向于给更长、更详细的回答打出更高的分数，即使这些回答可能包含冗余信息或不够切题。这可能会激励模型生成冗长的而非高质量的内容。
- **自我偏好偏见：**LLM 裁判可能偏爱由其自身或同系列模型生成的输出。
- **情绪与语气偏见：**裁判模型容易受到回答中所表达的情绪和语气的影响。它们可能偏爱听起来更自信、更积极的回答，即使这些回答在事实层面存在错误或误导性。
- **评分粒度有限：**LLM 裁判在处理粗粒度的评分任务（如二元判断或1-5分制）时表现相对可靠，但当评分量表变得更精细时（如1-100分制），其评分的随机性和随意性会显著增加，导致评估结果的可靠性下降。

这些偏见的存在，揭示了一个深刻的问题：LLM 裁判产生的定量分数带来了一种客观性的觉。这些分数看似是精确的、数据驱动的，但实际上它们是通过一个充满主观偏见的过程生成的。因此，一个来自 LLM 裁判的排行榜分数，并非对模型质量的纯粹客观测量，而是对该裁判模型内在偏好和价值取向的反映。这意味着，在解读评估结果时，关键问题不应是“模型的得分是多少？”，而应是“产生这个分数的裁判模型存在哪些偏见？”。这要求我们必须将评估评估者本身，作为构建可信评估流程中一个不可或缺的环节，以避免即当一个指标成为目标时，它就不再是一个好的指标。

2、递归依赖

LLM 裁判最根本的局限性在于其自身的能力上限。一个简单的道理是：一个 LLM 无法可靠地评估它自己无法正确回答的问题的答案。

- **能力天花板：**研究明确指出，LLM 作为裁判的评分能力与其作为应试者的答题能力之间，存在强烈的正相关关系。这意味着，对于那些最具挑战性、最前沿的问题，恰恰是评估需求最迫切的领域，即 LLM 裁判的可靠性最低。这构成了一个严重的悖论：一个较弱的 LLM 裁判无法公正地评估一个比它更强的候选模型。

- **推理幻觉：**更糟糕的是，作为裁判的L LM 本身也可能产生幻觉。它可能会为其评分提供听起来头头是道、逻辑严谨的解释，但这些解释本身可能是虚构的或与事实不符的。这使得验证裁判的判断过程变得异常困难。

为了有效评估一个前沿模型，理论上需要一个能力更强或至少相当的裁判模型。然而，要验证这个更强裁判的公正性，又需要一个更更强的裁判，如此循环往复，形成了一个无解的递归依赖。在绝对的前沿，自动化评估似乎永远落后于能力的发展。要打破这个循环，只能依赖那些不依赖于更强裁判的评估方法：首先是无可替代的人类专家评估；其次是动态的、以发现漏洞为目标的对抗性测试（红队演练），其目的不是精确评分，而是使模型失效；最后是探索模型内部状态和认知过程的内在评估。这意味着，随着 AI 模型变得越来越强大，在评估其安全性和可靠性时，对人类专家和红队演练的依赖将会增加，而不是减少。

9.3.5 提升裁判可靠性的缓解策略

尽管 LLM 裁判的偏见和局限性根深蒂固，但通过一系列精心设计的技术和流程，可以显著减轻其负面影响，从而构建更可靠的自动化评估系统。

1、提示工程技术：

- **思维链（CoT）与分步推理：**强制裁判在给出最终分数前，先详细阐述其推理过程。这种方法能显著提高判断的逻辑性和一致性。
- **少样本提示（Few-Shot Prompting）：**在提示中提供几个高质量的评估示例（包括好的和坏的回答及其对应的正确评分和理由），可以有效地“校准”裁判，使其更好地理解评估标准。
- **清晰的评分指南：**用明确、无歧义的语言定义评分量表上的每一个分数点，并具体说明何为“有帮助的”、“有毒的”或“缺乏事实依据的”，以减少裁判的主观臆断空间。

2、结构化缓解措施：

- **交换位置：**为了对抗位置偏见，在进行成对比较时，可以进行两次评估，第二次将两个回答的位置互换。只有当裁判在两种排序下都给出一致的判断时，才采纳该结果。
- **基于参考的评分：**提供一个黄金标准的参考答案，可以极大地提升裁判在评估事实正确性方面的能力，尤其是在裁判自身对问题领域不甚了解时。
- **约束裁判输出：**通过技术手段限制裁判的输出格式（例如，强制要求评分必须是单个数字字符），并结合使用输出词元的概率，可以减少生成无关内容的可能性，提高评分的稳定性。

3、基于模型的解决方案：

微调：将一个通用 LLM 在一个高质量的、带有人类评估标签的数据集上进行微调，可以训练

出一个专门的“裁判模型”。这种模型能更好地与人类的评估模式对齐，但也是成本较高的一种方式。

- **使用多个裁判：**不依赖单个裁判的判断，而是同时使用多个不同的 LLM 组成一个陪审团，然后通过投票或平均分等方式综合它们的判断结果。这种方法可以有效地平滑掉单个模型的特有偏见，得出更稳健的评估结论。

4、高质量数据集：

- **数据集：**构建一份完整的数据集，作为标准答案，而不是让 LLM 自行判断准确性。
- **Bad Case 集合：**不断收集在线服务所有失败的输，在修正完应用的处理逻辑后，加入到集合中。

通过综合运用这些策略，组织可以在享受 LLM-as-a-Judge 带来的效率和规模优势的同时，最大限度地控制其内在风险，从而建立一个更值得信赖的自动化评估流程。

9.4 自动化评估落地实践

9.4.1 搭建评估系统的痛点

在 AI 应用进行评估的过程中，尽管已有多种评估框架和方法被提出，但在实际落地应用中仍面临诸多痛点。这些痛点不仅影响评估结果的可靠性，也制约了从评估中获得可行动的洞察，进而优化模型表现。以下是对这些痛点的详细展开：

- **数据的采集：**评估依赖于数据，目前各种 AI Agent 框架众多。在一个典型的 Agent 框架中，有 Planning 模块、访问 LLM 模块、工具模块，每个模块都有自己的数据。在评估时，需要把多种模块的数据关联起来。
- **LLM Judger 自身的准确性问题：**幻觉与语义理解偏差。评估系统本身依赖 LLM 进行打分时，评估模型也可能产生幻觉，形成双重不确定性。
- **Ground Truth 的获取困难：**评估需要可靠的“真实答案”（Ground Truth）作为基准，但在许多实际场景中，Ground Truth 难以获取或定义模糊。
- **特定领域黄金指标的定义与构建困难：**不同应用场景需要不同的评估指标（如准确性、相关性、安全性、工具调用正确性等），而通用指标（如 BLEU、ROUGE）在特定领域往往不适用。
- **评估成本高，难以规模化：**自动化评估依赖大模型，单次调用成本高，尤其在需要多维度、多轮评估时，成本呈指数增长。
- **数据预处理的困难：**在复杂系统中，模型行为涉及多步推理、工具调用、外部交互等，评估需深入到执行路径（Trace）或最小执行单元（Span）。
- **从评估到可行动的洞的断层：**评估的最终目标不是生成一个分数，而是为模型优化提供明确方向。然而，当前评估结果往往停留在“模型 A 比模型 B 得分高”这一层面，缺乏可操作的反馈。

9.4.2 评估系统的不同 Level

以终态为目标，我们可以看一下，对于一个完美的评估系统，应该有哪些能力。一个评估体系，按照不同的能力，分成5个等级：

	level 0.5	level1	level2	level3	level4
①固定评估模板	✓				
②人工指定 Ground Truth 到 Prompt		✓			
③支持预处理评估内容（过滤、删减）和后处理评估结果（统计：发现 Pattern、根因分析、对比）		✓			
④支持自定义评估		✓			
⑤支持回测		✓			
⑥根据用户的特定评估需求，帮助用户生成特定指标的评估模板			✓		
⑦基于文本聚类去重，理解用户的数据内容，生成全面的黄金指标和对应的评估模板				✓	
⑧支持合成 Ground Truth				✓	
⑨Agentic Eval：用户输入少量提示，自动理解数据，使用合成的Ground Truth，生成评估模板，多步操作，生成周期评估任务，生成报告。遇到无法归类的错误，自动加入模型					✓

从最基础的简单评估，到最复杂的 Agentic 评估，评估系统逐渐向着更加智能化，更加完善的方向上发展。

- **固定评估模版：**最简单的评估，采用一个简单的评估模板，完成一个单一任务，无法扩展，也无法处理复杂的场景。
- **Ground Truth：**Ground Truth 即是标准答案，只有有了标准答案，才能构成一个完整的 Benchmark，类似于程序的 UT、FT 等回归测试，知道如何判定答案是至关重要的。

- **预处理和后处理：**

- a. 在以下场景中，需要对数据进行预处理，针对处理后的数据进行评估：
  - 原始内容包含脏数据，例如包含一些 HTML 标记、符号、乱码等需要进行处理。
  - 只需要对原始数据的部分内容进行评估，因而需要抽取操作。
  - 在评估中需要涉及到多步操作，例如同时获取工具调用结果和 LLM 调用结果。因而需要对复杂的 Trace 数据进行关联后一起评估，例如对一个完整的 Trace 会话进行完整性评估，因而需要把 Trace 数据聚合在一起。这需要有一定的预处理手段，把多条数据关联起来。

- b. 在评估完成后，以下场景需要对数据进行后处理分析：
  - 查看分数的分布。
  - 查看分数的变化趋势。
  - 评估结果和其他内容进行关联分析，例如和用户、地域、爱好等进行关联。

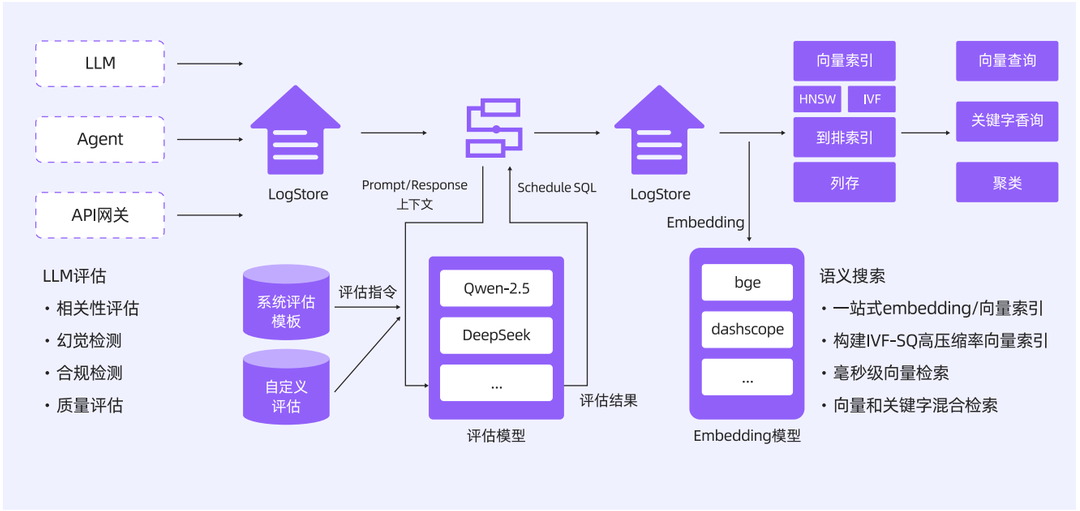
- **支持自定义评估：**内置模板无法满足用户的所有需求，只有支持自定义评估，才能在任意场景完成评估。
- **支持回测：**评估一般在两个场景使用。在研发阶段中，评估用来做实时反馈，根据反馈内容优化 Agent 的设计；在上线后，评估用来做线上的监控，监控泛化能力。日常的回测，能够帮助快速发现新的问题。同时，模型的迭代也是十分迅速的，以前无法满足的需求，可能随着模型的迭代就可以满足了，因而要实时的监控大模型的性能，通过回测可以快速发现模型的优化进度。
- **生成评估模版：**在自定义评估中，用户需要自行编写评估模板，如果能够帮助客户生成自定义评估任务的模板，减少用户的调试时间，那么可大大提升评估系统的效率。该能力作为一个可选的能力，因而评级为 level2。

在不同的场景下，评估方式千差万别，评估指标也存在很大的不同，每个业务场景有自己的独特的黄金指标，例如 SQL 生成领域，生成 SQL 的质量是黄金指标。如何定义领域内特殊的黄金指标、并且关注指标的重要性和全面性是一个挑战性的问题。

- **合成 Ground Truth：**Ground Truth 的重要性不言而喻，而获得 Ground Truth的过程成本却十分高昂，一般需要投入巨大的人力资源进行人工标注，产能匮乏。而借助于 LLM 的强大的生成能力，可以帮助开发者快速合成需要的数据集。当然合成的数据集需要进行人工抽样检验，保障生成质量。
- **Agentic 评估：**Agent 模式提升了处理 case 的复杂度，相比单步的 LLM Judger 可以处理更多更加复杂的场景，也提升了评估过程的自由度。

9.4.3 云原生评估系统

基于云原生的评估系统，可以帮助开发者轻松解决了以上落地中的痛点。



阿里云云原生可观测解决方案提供了一站式的数据采集、存储、评估、语义检索能力。

- 一站式的数据采集，无缝实时把大模型推理日志集中采集，集中存储，解决数据孤岛的问题：
  - 兼容 Opentelemetry 协议。
  - 自研无侵入探针：以 OpenTelemetry Python Agent 为底座，增强大模型领域语义规范与数据采集，提供多种性能诊断数据，全方位自监控保障稳定高可用。
  - 开源采集器 LoongCollector：装机量达到百万的高性能日志采集器，实时采集增量日志到服务端。

- **在线数据预处理的能力：**基于 SQL/SPL 强大的数据处理能力，完成提取/去重/关联登操作。提取操作可以只评估关键的信息；去重把重复的信息进行压缩，减少 LLM Judger 的负载、关联把相关数据合并在一起进行评估。

- **实时评估能力：**
  · 在 SQL/SPL 中提供评估算子，和预处理计算无缝衔接。
  · 评估算子无缝集成 Qwen 等最先进的大模型，提升 LLM Judger 的评估能力。

- **评估模板建设：**
  · 支持内置多种常见的评估模版，包括通用的准确性、安全性，以及工具调用、Rag 调用、Planning 等阶段的评估，覆盖大部分的场景。
  · 支持以 SPL、SQL 的形式自定义评估命令。在自定义评估中，用户可以自定义自己的数据预处理逻辑，组装 Prompt，以及处理响应结果。
  · 自定义评估模板，在用户自定义场景下，如果自定义的评估 Prompt 长度较长，可以把评估模板保存在 MetaStore 中，通过 MetaStore 通过名称加载 Prompt，避免评估脚本太长，影响编辑效率。

- **后处理统计：**基于 SPL/SQL 对评估结果进行二次加工统计。例如进行 A/B 测试，对比不同 Prompt 模板的效果、对比不同模型的效果。

- **语义搜索：**
  - 基于语义搜索，对评估对象和评估结果进行精准筛选。
  - 语义聚类，对评估结果进行聚类分析，发现高频的 Pattern，以及离群点。

基于云原生的评估，可以一站式创建评估任务、查看评估结果，分析评估结果。

1、创建评估任务

在评估任务中，选择评估模板的选项，包括

- 评估任务语言。
- 评估类型，包括通用的 Rag 评估、Tool 调用评估、Planning 效果评估、通用评估。
- 选择评估任务名称。每个评估类型下包含了若干个评估任务。
- 输入过滤语句，过滤语句表示只评估 Trace 中的某些类型的节点，例如只评估 LLM 节点，或者只评估工具调用节点。

2、评估任务说明

- **评估任务按照结果形式分为两种：**
  - 结果以评分表示，附加一个评分解释。
  - 结果为语义评估，从原始内容富化出主题、总结等语义信息。
- **评估任务按照任务场景分为以下几种：**通用场景评估、语义评估、Rag 评估、Agent 评估、工具使用评估。

3、通用场景评估

0分表示需要关注，1分表示不需要关注，介于0-1分之间表示部分需要关注。

序号	评估任务	0分	1分
1	准确度	表示完全不准确	表示完全准确
2	计算器正确性	表示完全不正确	表示完全正确
3	简洁性	表示完全不简洁	表示完全简洁
4	包含代码	表示包含代码	表示不包含代码
5	包含个人身份信息	表示包含个人身份信息	表示不包含个人身份信息
6	上下文相关性	表示完全不相关	表示完全相关
7	禁忌词	表示包含禁忌词	表示不包含禁忌词
8	幻觉	表示存在幻觉	表示完全没有幻觉
9	仇恨言论	表示包含仇恨言论	表示不包含仇恨言论
10	有用性	表示完全无用	表示非常有用
11	语言检测器	表示无法检测语言	表示准确检测语言
12	开源	表示开源	表示非开源
13	问题与 Python 相关	表示与 Python 相关	表示与 Python 无关
14	毒性	表示有毒性	表示无毒性

4、语义评估

语义评估是对数据进行语义理解和处理，包括以下功能。

- **实体信息抽取（NER）**

从文本中抽取出原始的实体信息，包括人名、地名、组织名、公司名、时间表达、货币金额、百分比表达、法律文件、国家/地区/政治实体、自然现象、艺术作品、事件、语言、标题、图片和链接等。

- **格式信息提取**

提取 Markdown 或其他文本格式中的标题、列表、强调字体（粗体/斜体）、链接名称和 URL、图片地址、代码块、表格等内容。对表格进行特殊处理，将每个表格转换为 JSON 格式，其中每一列对应一个 key 和 value。

- **重点词汇抽取**

从长文本中抽取出代表语义的核心词汇，用于描述文本的主要含义。

- **数值信息抽取**

提取文本中出现的数值及其相关信息，如温度、价格等。

- **抽象信息抽取**
  - 用户意图识别：识别用户意图，如查询检索、文本润色、决策判定、操作指导等。
  - 文本摘要：用几句话描述文本内容，每句话描述一个话题。
  - 情绪分类：判断文本情绪为正面、负面或中性。
  - 主题分类：对文本涉及的主题进行分类，如体育、政治、科技等。
  - 角色分类：识别文本中涉及的角色，如系统、用户、医生等。
  - 语言分类：识别文本使用的语言，如中文、英文等。

- **生成相关问题**

针对给定文本，从不同角度提出若干个可以由文本内容回答的问题。

5、Rag 评估

序号	评估任务	0分	1分
1	Rag 召回语料和问题的相关性	完全不相关	完全相关
2	Rag 召回语料和答案的相关性	完全不相关	完全相关
3	Rag 语料是否存在重复	完全重复	完全不重复
4	Rag 语料的多样性	多样性最差	多样性最好

6、Agent评估

序号	评估任务	0分	1分
1	Agent 指令是否清晰	不清晰	清晰
2	Agent 规划是否有错误	存在错误	正确
3	Agent 任务是否复杂	复杂	不复杂
4	Agent 执行路径是否存在错误	有错误	无错误
5	Agent 是否最终达到了目标	未达到目标	达到了目标
6	Agent 执行路径是否简洁	不简洁	简洁

7、工具使用评估

序号	评估任务	0分	1分
1	规划是否调用了工具	否	是
2	遇到错误参数时，是否修正了错误的参数	未修正错误	修正了错误
3	工具调用的正确性	错误	正确
4	工具参数是否有错误	有错误	无错误
5	工具调用效率	效率较低	效率较高
6	工具是否合适	不合适	合适

8、查看评估明细

评估明细展示每一条评估的结果，在评估明细中，包括了：

- **TraceID、SpanID**：可以和原始的数据关联起来。
- **评分**：代表本次任务的评分。
- **解释**：代表评分的解读。

9、查看评分的分布

借助于可观测大盘，展示评分从0到1的分布情况，每个分数段上的 Trace 个数，基于大盘可以清晰的查看所有任务的质量分布。

9.4.4 评估系统总结

评估系统在 AI 应用中扮演了至关重要的角色，而 LLM-as-a-judge 是评估前沿方案。云原生集成了一站式的 Trace 数据采集，以及前处理、评估、后处理结合在同一个计算引擎中，开发者可以在一个系统内完成所有的评估任务。

AI 安全

AI Security

CHAPTER

09

AI Evaluation

P259-P278

10

AI 安全风险的来源和分类

保护应用安全

保护模型安全

保护数据安全

保护身份安全

保护系统和网络的安全

P281-P302

11

Conclusion and Outlook

P305-P308

ALIBABA CLOUD

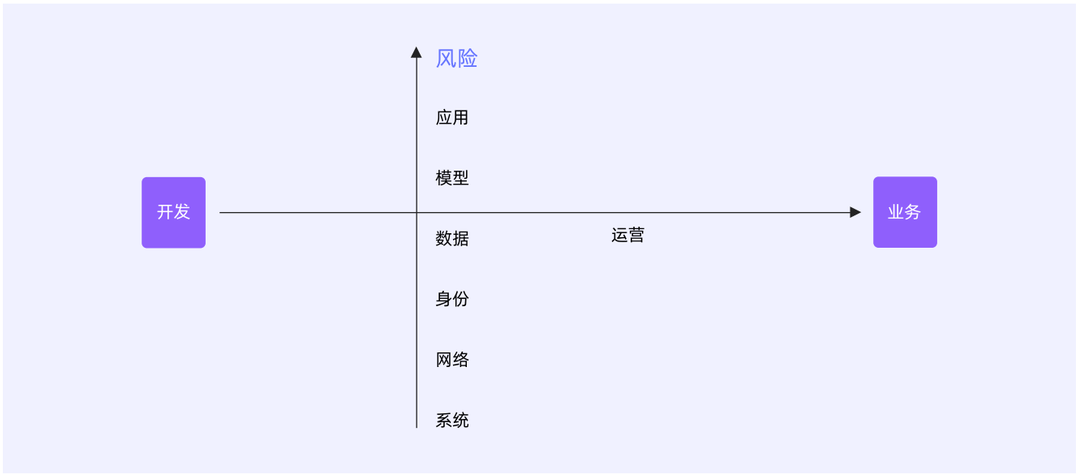
10.1 AI 安全风险的来源和分类

根据国内某安全咨询机构对包括金融、能源、制造、交通物流、电信运营商等十大行业客户的调研，82%受访者重点关注 Agentic AI 自身的风险分析及安全管控建议。AI 领域，尤其是Agentic AI 正处于结构性飞跃的风口，大多数企业已经在实施或者考量引入 Agentic AI ，其中，安全是企业最重要的考量之一。

AI Agent 最大的创新在于其自主决策和执行任务的能力。但这可能会成为攻击者恶意利用，例如通过提示词注入等手段操控 Agent 行为，而越权访问操作则可能导致数据泄露。同时，若 AI 基础设施存在漏洞，还有可能引发注入、逆向攻击和算力被滥用等威胁。最后，AI 与生俱来的非预期行为以及输出的不可预测性风险，加剧了内部治理和合规的挑战。

事实上，AI 的引入给企业在各个层面都带来了新的安全风险。企业在拥抱 AI 技术过程中，面临来自从系统（计算）、网络、身份、数据、模型以及应用等诸多方面的安全风险，主要体现在：

- **系统风险**：AI 模型软件的供应链风险、暴露面风险以及算力劫持风险。
- **网络风险**：面向公网的入侵攻击，以及内网的隔离风险。
- **身份风险**：对非人类身份（NHI）的管控，越权访问，身份冒充等。
- **数据风险**：Agent 模型训练时的数据投毒，以及输入/输出阶段的敏感信息泄漏等。
- **模型风险**：Agent 模型输入输出内容的恶意诱导、提示词攻击等风险。
- **应用风险**：当 AI 在线上提供服务时，会面面临 Web 入侵、DDoS 攻击导致服务不可用等风险。



总之，拥抱 AI 不能忽视一同到来的新的安全风险，这些新风险覆盖开发和运维的完整业务链路，这就要求企业需要构建起一个全栈的安全保护框架用于保护 AI。本章开始，我们将从应用安全、模型安全、数据安全、身份安全、系统和网络安全 5 节内容去阐述如何构建全栈的安全保护框架。

10.2 保护应用安全

10.2.1 背景和挑战

相比于传统的应用安全主要关注于静态代码和确定性逻辑不同，AI Agent 的应用安全发生了根本性改变，我们需要保护的对象为一个动态的、基于推理的、能够自主行动的系统，这不仅放大了传统漏洞的风险，也催生出了全新的攻击向量。

AI Agent 的核心能力便是可以与外部系统和工具交互，这也成为其面临的主要的安全风险。当 Agent 被授权访问网页、查询数据库或执行代码时，其角色就从一个信息的处理者转变为一个行动的执行者。这种转变，使得攻击者不再需要直接寻找暴露在公网服务中的漏洞，他们只需操纵相关的 AI Agent，就能让这些 AI Agent 代为攻击内外部的弱点服务。

服务器端请求伪造（SSRF）是引入 AI Agent 后最突出的威胁，具备联网能力的 Agent，其访问 URL 的核心功能使其天然成为 SSRF 的理想入口，攻击者仅需通过自然语言提示，即可诱导受信任的 Agent 向内部网络发起恶意请求，窃取云环境凭证或扫描内网资产。同时，Agent 也可能被攻击者诱导产生包含 SQL 注入或 XSS 等恶意载荷，攻击与之交互的后端服务。这个过程中，Agent 充当了攻击载荷的“投递员”，其合法的 API 调用行为使得这种攻击更难被传统的安全设备所察觉，将可信的助手转变为攻破服务器防线的“特洛伊木马”。

10.2.2 新的攻击面

AI Agent 中大型语言模型的引入，将攻击面从传统的代码和接口拓展到了模型的认知层面，催生出了新的攻击面。

如以消耗目标财务预算为核心的攻击（拒绝钱包攻击， Denial of Wallet, DoW），由于 AI Agent 被设计为自主执行任务并调用 API，这种自主性也成为了其被利用的弱点。攻击者不再需要攻破系统，只需通过构造恶意的语言诱导，就能将 AI Agent 变为一个无节制的资源消耗工具，使其无法正常为其他用户提供服务，达到拒绝服务类攻击的目的。攻击者通过构造复杂度炸弹式的指令，诱使 AI Agent 陷入生成海量文本的陷阱，迫使模型进行深度推理并生成数万乃至数十万的 Token，直接导致其自身的计算成本飙升。

在另一个维度上，攻击者也可以攻击 AI Agent 调用的外部 API，高频或海量触发复杂或昂贵的 API 调用，迅速耗尽企业的服务资源与预算。另外一部分风险在于对 AI Agent 整个工作流的劫持，由于 AI Agent 中大型语言模型无法精细地区分可信的系统指令和不可信的用户数据，攻击者将恶意指令隐藏在 AI Agent 将要处理的网页、表格、PDF 等外部数据中，当 AI Agent 处理这些被污染的数据时，隐藏的指令就会被执行，导致其行为被完全操控，泄露敏感信息或执行恶意操作。

10.2.3 防护思路

应对 AI Agent 所面临的应用安全风险，必须建立一个多层次的纵深防御体系。

首先，我们还是需要沿用并强化传统的应用安全最佳实践，包括对所有输入进行严格的验证，对 AI Agent 的所有输出进行严格的上下文编码，遵循最小权限原则，限制 AI Agent 及其工具的能力范围，在独立的沙箱环境中执行高危操作。在传统的应用安全最佳实践基础上，为抵御 DoW 类攻击，需要实施严格的资源治理策略，包括对 Token 消耗与高成本 API 调用设定精细的预算、配额与速率限制；针对工作流劫持类攻击，则需要设计更加完善的上下文隔离机制，以确保用户数据无法篡改系统指令。

10.2.4 解决建议

在安全防护产品和方案的选择上，一方面我们要重点关注对在 AI 场景下输入输出做安全检测的 AI 安全护栏方案，以及相关 AI 应用开发平台和网关（如百炼、Dify、API 网关、WAF 等）对 AI 安全护栏的集成；另一方面，由于 AI Agent 进一步打破了传统的攻击面和网络边界，要更加关注构纵深防御、动态检测、运行时安全等多种安全措施的组合。

例如当 AI Agent 未经过 WAF 或防火墙直接在内网发起恶意请求时，可以通过 RASP 等运行时检测方案或 VPC 防火墙来做防护，当 AI Agent 在用各种方式泄露敏感数据时，可以通过 API 安全、NDR 等流量安全方案，从敏感数据流转的角度去及时发现风险。

ALIBABA CLOUD

10.3 保护模型安全

10.3.1 背景与挑战

随着大模型技术加速向 AI 原生应用渗透，AI Agent 作为具备感知、决策与执行能力的核心载体，正广泛应用于智能客服、虚拟助手、知识问答等直接面向用户的交互场景。然而，其开放性、自主性与多模态输入输出特性也显著扩大了系统的风险敞口。AI Agent 不仅需要理解用户的自然语言指令，还需处理多模态输入、访问外部知识库、调用函数接口，甚至生成结构化内容或执行操作。这一系列行为链条使得其成为攻击渗透、内容失控与数据泄露的关键入口。因此，针对 AI Agent的运行特点构建细粒度、全流程的安全防护机制，是保障大模型应用可信落地的核心前提。

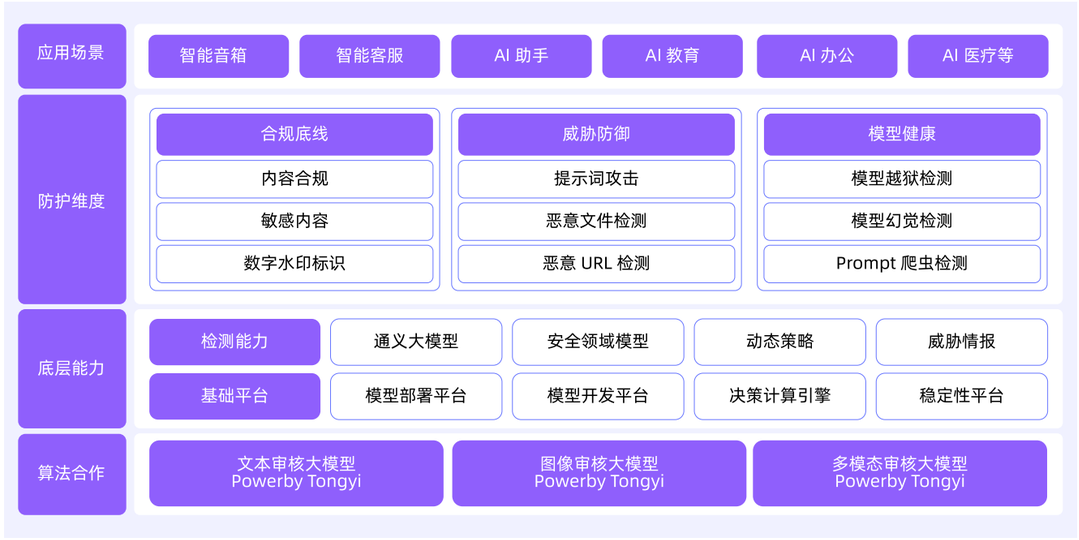
当前 Agent 面临的核心威胁已超越传统 Web 应用范畴，需系统性应对以下三类和模型安全相关的高危场景：

- **输入层威胁：**包括对抗样本攻击（如通过微调图像/音频诱导多模态Agent误判）、提示词注入（Prompt Injection）的变种攻击（如上下文分割攻击、语义混淆攻击），以及通过恶意文件（如PDF隐写术）触发供应链漏洞；
- **推理层威胁：**涵盖模型越狱（Jailbreaking）导致的伦理失控、RAG知识库的定向爬取（Prompt Crawling）引发的数据资产泄露，以及函数调用劫持（如篡改API参数执行未授权操作）；
- **输出层威胁：**涉及生成式钓鱼内容（如伪造银行通知）、模型幻觉（Hallucination）在医疗/金融场景的致命误导，以及通过隐写术（Steganography）在AIGC内容中植入隐蔽指令。

因此，针对 Agent 的运行特点构建细粒度、全流程的模型安全防护机制，是保障大模型应用可信落地的核心前提。

10.3.2 防护机制

为应对上述挑战，大模型原生防护机制（如 AI 安全护栏）应运而生，作为连接应用逻辑与大模型能力之间的“可信中间层”，提供覆盖输入、推理、输出全链路的一站式防护体系。其核心能力涵盖九大维度：



- **内容合规审核：**Agent 在与用户持续对话过程中可能因上下文引导或语义漂移生成涉政、低俗、歧视性等违规内容。护栏机制基于大模型审核引擎，在 Agent 响应生成前进行实时内容扫描，精准识别显性违规与隐喻表达（如变体、谐音、意识形态渗透），确保输出始终符合法律法规与社会主流价值观。
- **提示词攻击防御：**攻击者常通过精心设计的提示词（如 “Ignore previous instructions”）诱导 Agent 绕过系统约束，实现 “越狱” 或指令覆盖。通过构建融合同步检测与异步分析的多模型混合架构，可在 Agent 接收输入时即刻识别对抗性提示，阻断恶意指令注入路径，保障 Agent 行为始终处于预设策略边界内。
- **敏感信息防护：**在用户与 Agent 交互过程中，可能无意输入个人身份信息（PII）、企业密钥、银行卡号等敏感数据。需要对上述敏感数据进行精准识别与脱敏处理，防止这些信息被 Agent 记忆、记录或在后续响应中意外泄露，满足 GDPR、网络安全法等合规要求。
- **恶意文件检测：**当 Agent 支持文档上传功能（如简历解析、合同问答）时，攻击者可能通过 PDF、PPT、DOC 等文件嵌入宏病毒、可执行脚本或隐藏指令。通过对文件格式的深度解析，检测并清除嵌套攻击代码，从输入源头阻断 Agent 被恶意操控的风险。
- **恶意 URL 拦截：**Agent 在执行网络搜索、知识检索或调用外部 API 时，可能解析或生成钓鱼链接、恶意网站地址。通过对所有URL进行实时风险评估与黑名单匹配，防止 Agent 成为攻击跳板或诱导用户访问高风险站点，保障终端用户安全。
- **提示词反爬机制：**攻击者可能构造特定提示序列，试图通过 Agent 反复试探 RAG 知识库内容或模型训练数据，实施 Prompt 爬虫攻击。通过动态行为分析与模式识别，识别异常查询频率与语义意图，及时阻断数据资产被系统性窃取的风险。
- **模型越狱检测：**通过特定输入，攻击者可能突破大模型与预设的安全机制，使其生成不符合伦理、法律、正常价值观的内容。应对这类攻击模式，除了在前置输入环节部署提示词攻击防御

以外，通过对模型输出结果的越狱检测和判定，构建全链路、多环节的保障。

- **模型幻觉抑制：**由于缺乏真实世界验证机制，Agent 在推理过程中易产生“幻觉”，输出看似合理但事实错误的信息（如虚构法规、错误医疗建议）。通过上下文信息一致性比对与外部知识核对机制，在关键决策节点对 Agent 输出进行可信度校验，显著降低高风险领域中的误判概率。
- **数字水印标识：**当 Agent 生成图片等内容时，安全护栏依据《人工智能生成合成内容标识办法》，自动注入可见或不可见的数字水印，实现 AIGC 内容的可追溯、可审计，防范虚假信息传播与版权纠纷，真正做到“生成有痕、责任可溯”。

以阿里云 AI 安全护栏为例，基于自研大模型审核引擎，不仅实现了对已知威胁的全面覆盖，更可依托通义大模型技术底座，持续进化多模态审核能力，在毫秒级延迟下支持高并发处理，兼顾安全性与可用性。同时，通过可视化策略配置、自定义黑白名单与灵活阈值调节，满足不同行业客户的差异化合规需求。

10.3.3 模型安全的未来

通用安全模型难以识别企业自身的业务风险，各行业在金融、客服、搜索等 AI 应用中，面临定制化合规挑战。企业需要“能听懂业务语言”的审核模型，识别特定风险。

自定义检测 Agent 可满足不同行业客户的特定业务风险识别需求。这是一种支持用户自定义标签与提示词的智能检测模块，能够精准识别行业特定、场景特定的业务风险。它引入了专用算法模型，并支持多行业多场景灵活配置，从而实现从通用到专属的安全检测能力升级。

未来，随着 Agent 能力的持续演进，原生 AI 安全护栏需要同步升级，致力于保障其在复杂场景下的安全性与可控性，为 AI 原生应用的稳健发展构筑坚实防线。

10.4 保护数据安全

10.4.1 背景和挑战

大模型应用过程中经历了6个数据阶段，数据采集和接入、数据传输、数据存储、数据访问、数据使用、数据删除等，核心需要保障业务在使用过程中的训练数据、Promt、知识库、多模态数据以及日志等多重数据的安全与稳定。

大模型在云上应用与使用存在以下三方面的数据安全风险：

- **模型训练中的数据风险**

首先，用户在模型训练过程中，存在原始数据被投毒、数据清洗不完善、数据存放不安全等因素，影响模型与应用在训练与使用过程中会出现安全风险。其次，企业用户关注企业商业秘密在传输、存储过程中的加密和防攻击，应用处理过程中的权限限制。对个人用户而言，则要保障对其个人数据的控制权和安全性，保证对数据处理的知情同意。此外，需要健全、完善模型安全机制，防止通过模型输出逆向推断出原始数据，导致敏感数据泄露。

- **模型建设，用户对数据的可控性**

用户需要了解并控制模型对数据的使用情况，避免用户数据未经授权用于模型训练，造成数据的秘密状态被破坏，商业价值被稀释。传统人工智能对用户行为数据的依赖，使得“应用数据会被用于模型”的观念深植人心，甚至演化为用户对智能应用的戒备。用户担心自己上传的数据或与模型的交互数据，特别是企业商业秘密，在未经授权时被公开，或被用于二次训练，转化为大模型厂商提升模型能力的语料。

- **模型应用，操作可审计和责任可追溯**

用户数据被模型应用处理，需要多方权责事先需约定、事后可审计可追溯。在模型数据处理的复杂情况下，极易出现敏感数据泄露，因此对数据安全权责的认定及各方责任的判断提出了新挑战，实现“谁持有谁负责”、“谁使用谁负责”、“谁运营谁负责”将变得困难。

一方面，需要在数据泄露或滥用方面，对各方应承担的责任事先进行原则性约束。另一方面，在调用模型进行应用编排时，需要对过程和多方权利信息进行记录管理，以备事后找到对应的问题源头和安全薄弱环节，和相关方进行权益主张。此外，上述过程仅靠模型服务商自己难以自证，需要更好的透明度管理和验证机制，做到操作可审计。

10.4.2 防护框架

为了应对上述数据安全挑战，降低用户对于大模型服务平台的数据安全担忧，需要基于云平台基础安全保障的安全防护机制和能力，以大模型服务的数据作为保护重点对象，围绕数据收集、传输、存储、访问、处理、删除等全生命周期数据安全保障需求，实现大模型服务数据安全保护各环节全覆盖，构建公共云平台+ 大模型服务平台的安全防护能力，并通过第三方权威机构的严格审计来验证自身的安全合规性，从而打造“平台可靠、链路可信、数据可控、自主可选、操作可审、责任可追”的数据安全保障体系。

10.4.3 构建全数据全生命周期的安全保障

构建面向大模型与 Agent 全数据全生命周期安全保障，通过增强用户在模型推理、微调、RAG 的数据收集、传输、存储、访问、处理以及删除的安全技术手段，集成数据安全、密钥管理服务（KMS）、内容安全以及访问控制（RAM）等诸多云原生安全产品的能力，实现高级数据安全目标客户的灵活、可配可拓展的 AI 数据保护的需求。

1、数据收集

（1）数据来源

用户使用大模型服务平台常见场景模型推理、RAG、模型微调中，主要涉及以下几类数据来源：

a. 用户上传类数据

- 模型训练和测试数据：用户上传到云平台、用于模型训练（包括持续预训练和微调训练数据、评测数据集），这类数据由用户准备好并上传到百炼平台，同时百炼平台也提供了数据清洗、数据增强等数据进一步加工能力。
- 知识库数据：以阿里云百炼平台为例，提供了 RAG 和 Agent 应用能力，这里将涉及到用户上传的各种非结构文档（如pdf、doc、pptx、word、md 等）、以及经过百炼平台加工后生成适合 RAG 使用的中间、向量化数据。简言之：上传的原始文档、解析后的中间结果、以及便于 RAG 检索的向量化数据。
- 多模态数据：以阿里云百炼平台为例，提供了 VL 视觉理解大模型、万相生图模型、ASR 模型、TTS 大模型等多模态模型，涉及到图片、音视频等多模态数据的上传和处理。

b. 模型文件和推理数据

- 微调训练后的模型参数文件：以阿里云百炼平台为例，提供模型微调与训练能力，使用用户授权的训练及测试数据，对百炼可选的模型进行训练后产生的模型参数文件。
- 提示词 Prompt 数据：包括原始 Prompt、大模型生成的答案、推理日志等数据。

c. 运行日志类数据

- 日志和 Tracing 的可观测数据：以阿里云百炼平台为例，应用和模型服务过程中，涉及的推理日志、应用日志和操作审计日志等，百炼平台也提供大模型应用 Tracing 可观测能力，涉及到可观测日志和 Tracing 日志存储。

（2）数据分类

数据分类分级作为数据安全和治理的基础，用户需要识别自有云账号下存储资源中存储的模型训

练和测试、知识库数据资产类别和等级，以及为敏感数据保护采取相应的安全策略、风险管理，减少数据泄漏风险。

以阿里云数据安全中心为例，提供的分类分级能力，将基于用户所在行业的角度，遵循国家标准及个性化业务诉求，从数据价值、敏感性、数据合规和业务需求等多角度将数据分为4个安全级别：S1、S2、S3、S4。支持多模态数据的解析及分类分级自动识别能力，识别范围包括云上结构化数据库数据、半结构化流量数据、非结构化文件数据、图片等多类数据，而后依据分类分级结果进行提取、脱敏、加密、权限控制。提供内置模版、识别模型、特征识别能力，内置模板涵盖通用互联网、金融行业、电力行业、车联网以及重保场景等多个行业场景。分类分级模版主要依据包括但不限于《GB/T 43697—2024 数据安全技术数据分类分级规则》、《GB/T 35273—2020 信息安全技术个人信息安全规范》、《JR/T 0197—2020 金融数据安全 数据安全分级指南》、《YD/T 3751-2020 车联网信息服务数据安全技术要求》等国家或行业标准，以及阿里巴巴数据治理的最佳实践提炼而成。同时也支持用户基于自身数据场景，调用自定义特征或产品内置通识类别分类特征，AI 生成导入或自定义编辑自身的分类分级识别规则模版。

（3）数据脱敏

对于大模型用户的模型训练和测试、知识库等数据，需要确保满足不包含影响大模型回答质量的敏感信息，以及《个人信息保护法》等敏感信息数据的安全合规要求。因此，采集时需进行数据脱敏。

以阿里云数据安全产品为例，提供数据脱敏能力，可覆盖多模态（图片、文件、结构化数据等）数据源，支持解析900 多种文件类型，包括：常见的文本类文档（txt、log 等）、办公类文档（word、excel、Power-Point 等）、压缩嵌套文件（zip、rar 等）、代码文件、设计工程文件等的提取解析。支持对敏感图片（身份证、驾照）类型识别、针对图片中的文字进行 OCR 识别。支持客户自定义敏感数据识别规则，提供基于关键词、Meta 信息、正则表达式、数据表列名的敏感数据识别能力。内置哈希、洗牌、加密、遮盖、替换等多种通用脱敏算法。

（4）数据去毒

为进一步提升定制用户的模型训练和测试、知识库数据及模型训练微调质量，防止针对模型数据集投毒攻击，需要对自有的存储资源中多种违规内容（例如涉黄、涉政、暴力、违法等）进行实时监控、检测和拦截。面向广泛人员提供大模型应用的用户，若有对人员输入进行安全防护需求，可接入提问护栏机制，进行恶意意图识别，数据安全产品提供覆盖图片、视频、语音、文字等多媒体的内容风险检测的能力，帮助用户发现暴恐、色情、涉黄、暴力、惊悚、敏感、禁限、广告、辱骂等风险内容或元素，并对内容进行拦截或清洗。

2、数据传输

（1）面向数据传输提供私有网络加密

数据访问建议使用诸如阿里云 Private Link 以及 VPC 网络加密能力，实现用户通过专有/私有网络进行访问用户存储的训练、微调、推理等数据。其次，面向 VPC 网络流量提供全面的监控与审计，可以针对网络异常调用进行实时监控与恶意行为拦截。

（2）提示词推理加密

针对模型推理服务（纯模型调用、RAG 应用、Agent 应用）提供全链路的加密方案。需要确保输入提示词 Prompt 和模型生成的答案全程不可见。解密只会发生在两个地方：根据输入 Prompt 进行 RAG 片段召回，以及大模型 Prompt 生成回答时。可以遵循最小必要原则对 Prompt 进行解密和使用，并且该过程只在内存中瞬间存在，不做任何的持久化存储。

（3）应用协议加密

在安全网络协议方面，为防止中间人、嗅探等网络攻击手段获取到用户与大模型服务，可在应用层使用 HTTPS 协议进行安全数据加密传输以及采用传输层安全性（TLS 1.3 1.2 1.1）协议，为云服务和用户之间的数据传输提供保障。TLS 可提供严格的身份验证、消息隐私性和完整性保障，能够有效检测消息篡改、拦截和伪造行为。

3、数据存储

（1）存储隔离

为防止数据泄漏、知识产权窃取、敏感数据被恶意利用，用户大模型相关数据需在用户自有的云账号下独立存储使用。

以阿里云为例，云平台保障用户数据安全完全归属，模型推理、训练、RAG 应用的用户数据均支持外接存储部署方式，支持外接 OSS、ES、ADB、SLS。数据采集接入外接存储归属客户的数据库实例，用户对数据 100% 完全自主可控，用户数据自主管理。阿里云的 ADB（向量数据）、ES（测试数据、长期记忆）、SLS（历史会话、审计日志、观测系统数据）、NAS（模型文件）、OSS（训练、测试集、知识库）数据存储服务中均做到租户化安全隔离。

同时，大模型平台服务运行在云的虚拟化执行环境上，虚拟化层面实现了不同磁盘空间之间的安全隔离。例如，推理环境的本地存储通过安全容器实现虚拟化级别的隔离，从而保证在推理环节中，执行用户请求的推理服务只能访问分配给它的磁盘空间。

（2）存储加密

以采用阿里云大模型服务平台百炼为例：

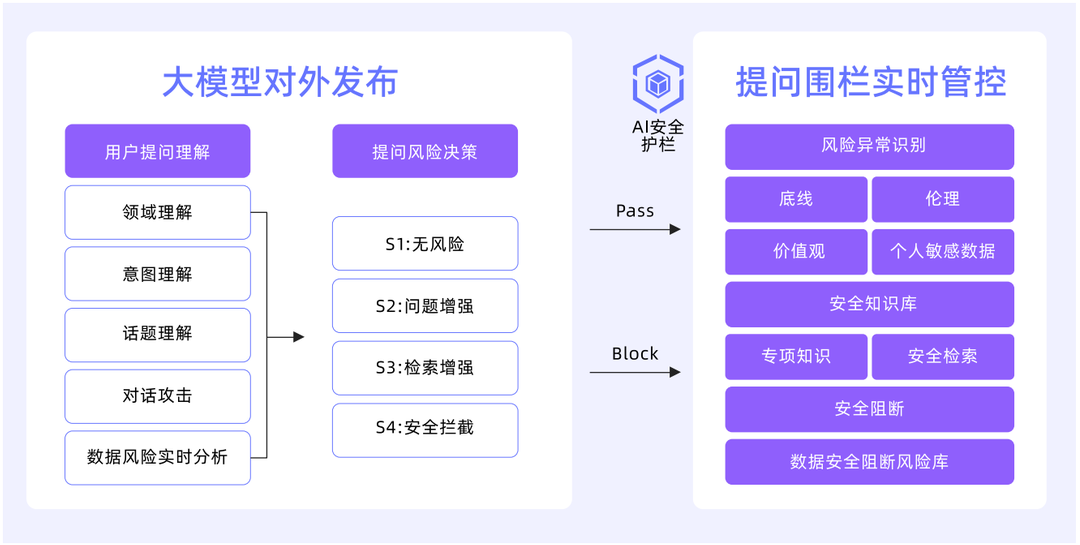
- 微调训练数据集、知识库、多模态数据加密阿里云大模型服务百炼平台的测试、训练数据集、知识库文件支持存储于用户外接的对象存储 OSS 中，OSS 支持服务端和客户端的存储加密能力。在服务端的加密中，支持使用服务密钥和客户自选密钥作为主密钥进行数据加密。在客户端的加密中，支持使用客户自管理密钥进行加密，也支持使用客户 KMS 内的主密钥进行客户端的加密。
- 模型文件加密：NAS 支持使用服务托管密钥和用户自选密钥作为主密钥进行数据加密。
- 向量数据加密：ADB 支持使用服务托管密钥和用户自选密钥作为主密钥进行数据加密。
- 历史日志加密：SLS 支持使用服务托管密钥和用户自选密钥作为主密钥进行数据加密。

4、数据访问

可采用云平台原生的访问控制（如阿里云 RAM，Resource AccessManagement,）服务产品提供，用于身份管理与资源访问控制，是账号安全管理和安全运维的基础。对于定制用户场景，大模型平台服务通过访问控制服务关联角色能力访问归属于用户的存储、ElasticSearch、SLS 等云上资源。

5、数据处理

数据处理安全至关重要，可使用云原生的大模型护栏机制。以阿里云AI安全护栏为例，可以满足客户在 Agent 与模型使用过程中的数据、内容信息的安全进行实时过滤与拦截，可以有效针对用户在 Agent 使用过程中的数据安全管控能力，从而减少数据泄露风险。 AI 安全护栏的安全能力支持风险异常识别，支持意图识别，实时过滤伦理、价值观、个人敏感数据，进行安全问答阻断。可根据用户需求，构建安全知识库与专项知识库，实现数据安全规则灵活自定义与风险决策。



6、数据删除

为了提供用户对自身数据的控制权，一旦用户申请注销大模型服务平台账号后，可按照云平台提供的账号注销流程，按照云用户隐私权政策要求进行删除个人信息，或对其进行匿名化处理。

以阿里云百炼大模型服务平台为例，同时支持对应用内的业务数据进行删除与迁移，删除指定的数据及其索引关系及过程文件等功能，范围包括用户上传数据、模型文件和推理数据、运行日志类数据。

若用户使用自主接入云产品 VPC 独立部署方案，用户对云账号下的各数据资源具有完全的处理与删除权限，并可根据自身业务场景需要对数据进行迁移。

ALIBABA CLOUD

10.5 保护身份安全

10.5.1 背景和挑战

AI Agent 背景下，非人类身份（NHI）的数量和复杂性正在迅速增长。常见的 NH I凭证包括 AK/SK、API 密钥、OAuth 令牌和证书，一个人类身份后面可达到45倍的机器身份。预计到 2030年，NHI 与人类身份的比例将达到50:1，在 AI Agent 占比越来越多的趋势下，企业需要投入更多资源来管理这些身份，确保其安全性。

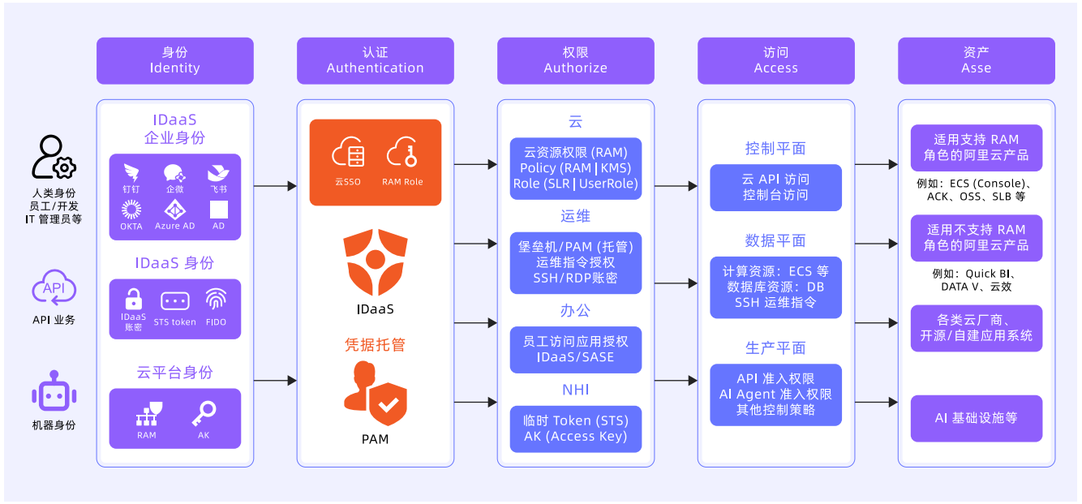
AI 身份的管理不同于当前人类身份管理，AI Agent 行为链复杂不可测，动态权限需求使得传统的静态权限模型难以适用，黑客可通过窃取 API 密钥、访问令牌等敏感凭据，可轻易绕过传统安全防护，实施数据窃取、资源劫持等恶意行为。

未来的 AI 智能体认证需要适应其动态多变的特性，比如当前的 MCP 虽然有了 OAuth2.1 的身份认证框架，但是需要 MCP 开发者自行创建身份认证，这可能导致安全隐患，尤其是对于远程部署的 MCP Server，如果缺乏身份验证，任何人都可以直接访问 MCP Server，并调用其提供的工具接口，从而导致未经授权的数据访问、信息泄露等问题。

10.5.2 身份安全闭环

1、身份权限治理整体架构

针对 AI 场景下面临的复杂凭据管理挑战，需要构建一套全方位的纵深防御框架，覆盖检测、管控以及审计等关键环节，并形成完整的安全闭环。



2、事前：NHI检测与风险识别

在事前阶段，可通过针对敏感 API 接口和文件的异常访问的管理与监控，使得企业能够迅速察觉到潜在的凭据盗用和数据窃取企图，从而在威胁造成实际损害之前采取有效的防范措施。

3、事中：动态权限管理机制

- **JIT 权限授予与业务场景结合**

在权限授予方面，建议采取“按需分配、用完即收回”的策略。系统会根据预设的业务规则和安全策略，自动生成一个具有严格时间限制（如5分钟时效）的临时密钥。一旦任务完成，临时密钥将自动失效，从而显著降低因长期有效的“僵尸凭据”而引发的安全风险。

- **细粒度权限管控实践**

在权限管控方面，强调细粒度的访问控制。

4、事后：自动化审计与清理

- **僵尸凭据自动化清理**

针对僵尸凭据问题，可以利用堡垒机的运维审计接口，从而消除废弃凭据可能带来的安全隐患。

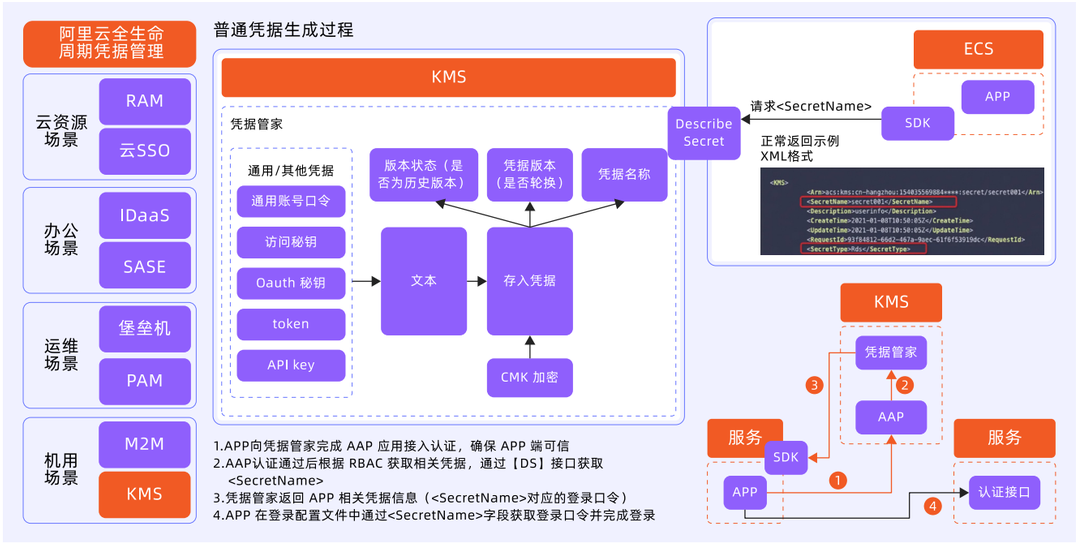
- **风险预测与对抗演练**

此外，可以基于 AI-BOM 以及 AI-BAS 能力，借助 AI 技术，梳理 AI 资产和敏感凭据，并模拟各种可能的攻击路径和场景，主动发现潜在的安全漏洞。

ALIBABA CLOUD

10.5.3 通过密钥管理实现凭据秘钥托管与轮转

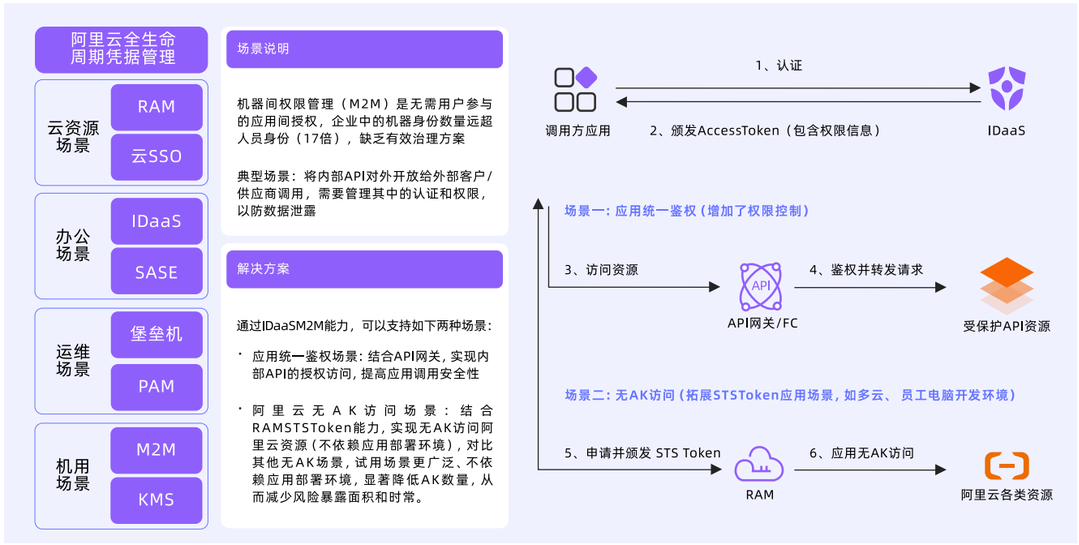
密钥管理可为企业提供安全可靠的凭证秘钥托管和轮转，保护敏感数据的加密密钥全生命周期安全。



- **统一密钥管理与加密服务：**云上密钥管理服务（KMS）支持托管多种类型的密钥，包括通用账号口令、Oauth 密钥、API Key 等。通过与云资源的紧密结合，为 ECS 实例、RDS 数据库等提供便捷的加密密钥管理服务。用户可以通过 KMS 的 API 或管理控制台轻松创建、启用、禁用和删除密钥，实现密钥的集中管控。
- **自动化密钥轮转机制：**KMS 具备自动密钥轮转功能，根据预设的轮转策略（如定期轮转、按需轮转），自动更新密钥版本。旧密钥版本将被标记为历史版本，但仍可支持解密操作，确保数据的向前兼容性。密钥轮转过程透明化，无需用户手动干预，有效降低因密钥长期未更新导致的安全风险。
- **细粒度密钥访问控制：**结合 RAM 策略和 KMS 权限管理，企业可以精确控制哪些用户或角色有权访问特定的密钥。基于资源标签和条件表达式，实现多维度的访问控制策略，例如限制特定 IP 范围内的密钥访问、指定时间段内允许密钥使用等，满足企业复杂的密钥管理需求。

10.5.4 通过 M2M 能力解决 NHI 凭据的托管

NHI 包括服务账号、应用程序内嵌账号等，它们在企业 IT 系统中扮演着关键角色，但传统的身份管理方法往往难以有效应对这些身份的复杂性和规模性。可通过 M2M（Machine to Machine）能力，为 NHI 凭据托管提供了一套创新且高效的解决方案。



- **集中化的 NHI 凭据管理：**阿里云推出的 M2M 解决方案允许企业将所有 NHI 的凭据集中存储和管理。这不仅包括传统的服务账号密码，还涵盖了各类 API 密钥、证书和其他形式的机器身份凭证。通过一个统一的控制台，管理员可以清晰地查看和管理所有 NHI 凭据的状态、使用频率和授权范围，从而实现对这些凭据的全局掌控。

- **动态授权与最小权限原则：**基于细粒度授权机制，M2M 能力支持对 NHI 的动态授权。这意味着每个非人类身份仅被授予完成其任务所必需的最小权限。例如，一个仅用于读取特定数据库表的应用程序，其权限将被严格限制在该操作范围内。通过与企业内部的访问控制策略相结合，动态授权可以随着业务需求的变化而灵活调整，确保NHI凭据的使用始终符合安全最佳实践。

- **自动化凭据轮转与生命周期管理：**M2M 支持自动化的凭据轮转策略，根据预设的时间间隔或触发条件（如检测到潜在的安全威胁），自动更新 NHI 的凭据。整个轮转过程无需人工干预，大大降低了因手动操作失误导致的安全风险。

10.6 保护系统和网络安全

10.6.1 背景和挑战

在 AI Agent 的全生命周期中，其基础设施的安全性直接决定了应用的可靠性、可信度和合规性。从模型训练到推理服务，AI 系统的复杂性、分布式架构以及对海量数据的依赖，使得任何基础设施层面的疏漏都可能引发模型盗用、数据泄露或服务滥用等严重风险。所以构建安全、可控运行环境，需要从全局安全态势到计算、网络等细节层面，建立多层次防护体系。

10.6.2 基础设施的统一安全态势管理

AI 应用的分布式特性与开源框架的广泛使用，导致资产构成日益复杂，安全团队常面临“看不清、管不到”的挑战。为此，需通过全局视角对 AI 资产进行统一管理，实现风险的提前感知与有效控制。

1、AI 资产的自动发现与盘点

安全管理始于清晰的资产清单。以阿里云云安全中心产品为例，提供智能资产发现能力，可穿透云环境的复杂架构，精准识别与AI相关的计算资源、容器、模型服务实例等。例如：

- **自动化识别：**系统可自动标记云服务器 ECS、人工智能平台 PAI、容器服务 ACK 中的 AI 资产，覆盖从底层计算节点到具体应用组件（如 Ollama、LM Studio）的全链路资源。
- **集中化视图：**通过“AI 应用”、“PAI”等标签分类资产，用户可快速定位云产品、容器镜像或主机资源，形成动态更新的资产清单，为后续风险分析提供基础支撑。

2、多维度风险评估

在资产可视化的前提下，通过持续性风险检测，将威胁暴露于攻击发生前：

- **开源组件漏洞检测：**针对 Ollama、LM Studio等 AI 框架，定期扫描组件漏洞，及时预警潜在暴露面。
- **公网暴露面分析：**监控 AI 服务的公网暴露情况，标记高风险端口（如未授权开放的 SSH 或 Jupyter 端口），帮助收敛攻击面。

- **配置风险检查：**基于阿里云及主流云厂商的AI安全最佳实践，检测 PAI、EAS 等平台的配置合规性，避免因配置错误引发风险。
- **敏感信息扫描：**通过镜像与文件系统无代理扫描技术，识别明文存储的API密钥（如阿里云在 PAI-EAS Token、OpenAI Key），防止凭证泄露导致的滥用。核心价值在于其“上下文关联”能力。例如，若发现某 Ollama 组件存在漏洞，系统将直接关联该漏洞对具体模型服务实例的影响，帮助安全团队按业务优先级制定修复策略，实现“以 AI 应用为中心”的精准治理。

10.6.3 计算层安全加固

ECS GPU 实例作为 AI 模型训练与推理的核心算力载体，其安全防护是基础设施的基石。需从基础防护与可信计算两个层面构建防线。

1、主机安全基础防护

所有承载AI应用的ECS实例需部署基础安全措施：

- **安全客户端部署：**安装云安全中心客户端，提供实时漏洞扫描、基线检查、异常登录检测及AK泄露告警。
- **安全组最佳实践：**遵循最小权限原则，关闭非必要端口，限制 SSH/RDP 等管理端口仅对运维堡垒机IP开放，避免公网暴露。

2、容器化环境安全防护

容器技术提升了 AI 应用的敏捷性，但共享内核的特性增加了容器逃逸风险。需通过安全沙箱与镜像全生命周期管理，平衡敏捷开发与安全需求。在阿里云上，容器服务 ACK 提供的安全沙箱为每个容器提供轻量级虚拟机隔离，实现内核级防护：

- **攻击遏制：**即使容器被攻陷，攻击者无法突破沙箱边界影响宿主机或同集群其他容器。
- **适用场景：**适合多租户AI服务或需运行不可信代码的场景，且性能损耗极低，接近原生容器体验。

3、镜像供应链安全

通过镜像扫描与签名机制，确保容器交付安全：

- **扫描与合规性：**在阿里云上，容器镜像服务 ACR 在 CI/CD 流程中自动扫描镜像，检测系统漏洞、应用漏洞、恶意样本及敏感信息（如硬编码密钥）。
- **签名与验签：**开发人员对合规镜像签名，ACK 集群仅允许携带有效签名的镜像部署，防止供应链投毒。技术优势在于将安全左移至开发阶段（镜像扫描）与运行阶段（沙箱隔离），形成从代码构建到生产部署的完整闭环。

10.6.4 网络隔离与访问控制

网络隔离与访问控制在 VPC 和安全组的基础防护之上，需构建更高级的网络边界防御体系，形成多层次纵深防护架构。

1、互联网边界防护
针对需要对外提供服务的 AI Agent，通过云防火墙以下核心能力强化公网流量管控：

- **智能访问控制：**支持基于七层协议、域名、URL 及地理位置的精细化策略配置。
- **主动威胁防御：**内置威胁情报库和攻击特征库，实时阻断已知攻击类型，提供漏洞临时修补的"虚拟补丁"。
- **全链路可视化：**提供全局流量日志分析与可视化展示，满足安全审计与攻击溯源需求。

2、内网微隔离防护
云防火墙的 VPC 边界防护功能可实现：

- **阻断攻击横向渗透路径：**通过深度监控跨 VPC 及混合云流量，防止攻击者从非核心业务区（如开发测试环境）向核心数据区（如训练数据存储 VPC）渗透。
- **零信任网络实践：**要求所有内部流量均通过身份认证与策略验证，消除传统内网信任模型的安全隐患

防护体系架构设计，安全组与云防火墙形成互补防护，云防火墙作为中枢管控节点，提供全局流量策略管理、威胁检测及可视化分析 二者联合构建"点-线-面"立体防御体系：通过安全组实现节点级防护（点），VPC 边界防火墙拦截跨区流量威胁（线），互联网防火墙把控公网入口（面），形成覆盖全网络层级的防护网络。

安全的 AI 基础设施并非一劳永逸的终点，而是一个需要持续评估、优化和演进的动态过程。随着 AI 技术和攻击手段的不断发展，安全防护体系也必须保持同步的迭代与升级，将安全真正内化为 AI 原生应用架构的固有属性。

CHAPTER

09

AI Evaluation

P259-P278

10

AI Security

P281-P302

11

通向 ASI 之路

P305-P308

总结与展望

Conclusion and Outlook

如果您已经读完前面 10 章，相信您已经对 AI 原生应用的概念和实践有了基本的了解，也对构建 AI 原生应用所需要的技术深度和知识广度有了较完整的认知。

通向 ASI 之路注定漫长，亦不乏荆棘。但若我们能看清那些贯穿始终的底层趋势，就能做到心中有数。接下来，我们将从技术架构、应用场景、治理体系、社会形态四个维度，对未来作些展望。理解它们，便能让我们在未来的不确定性中，始终保持战略定力与行动自觉。

技术架构：从模型到生态的跃迁

模型能力进化：从大语言模型到世界模型

- **自学习与自进化：**AI 模型将突破静态训练的局限，通过强化学习和动态反馈机制实现持续进化。
- **世界模型的崛起：**AI 将逐步构建对物理世界的完整感知和理解能力，能够模拟复杂环境，并通过自学习加速科学发现。

数据飞轮升级：从静态积累到动态进化

- **上下文工程的突破：**数据处理从提示词工程升级为上下文工程，通过动态强化学习和多模态数据融合，实现模型在真实场景中的自我优化。
- **合成数据的广泛应用：**合成数据将成为模型训练的核心资源，尤其在隐私敏感领域（如医疗、金融等），通过 AI 生成高质量、合规的数据集，降低数据获取成本。

AI 原生架构：从通用 Agent 到多 Agent 协同

- **元机器与任务分工：**复杂任务由大模型主导，简单重复任务由小模型执行，形成元机器架构。
- **多 Agent 协同网络：**AI 中台将沉淀基础模型能力和 Agent 服务，通过任务自动规划、工具调用和跨 Agent 协同，实现跨场景、跨系统的全局优化。

应用场景：从数字到物理的全面渗透

应用范式革命：从代码工程到 Agent 工程

- **开发门槛降低：**人类无需编写复杂代码，只需通过自然语言启动 AI Coding，再自动生成 Agent，实现以目标驱动开发，让非技术人员也能构建 AI 应用。
- **应用迭代加速：**Agent 支持动态调优，即当场景需求变化时，无需重新开发，只需更新规则或补充数据，Agent 即可自主适配，大幅缩短应用迭代周期。

AI 中台：从资源沉淀到智能协同

- **能力沉淀升级：**从中台沉淀基础计算和数据资源，拓展为沉淀可复用的数字员工组件，也就是把 Agent 封装为标准化模块，企业可像搭积木一样组合组件，快速配置业务流程。
- **协同目标升级：**从单场景效率优化到跨系统全局最优，中台打破业务系统壁垒，基于企业整体目标调度 Agent 资源，实现全链路协同。

服务模式变革：从被动响应到主动服务

- **服务场景延伸：**从接管数字世界服务到改变物理世界，Agentic AI 深度接管数字业务，Physical AI 与人类协同完成物理任务。
- **人机交互升级：**从 GUI（图形交互）到 GenUI（生成式交互）的模态进化，通过皮电传感器、心率监测、语调分析、微表情识别等，实时捕捉用户情绪状态，动态调整服务策略。

治理体系：从技术可信到社会契约

可信 AI 架构：技术、过程与结果的三重保障

- **技术可信：**建立算法审计机制，通过自动化工具检测模型中的偏见、漏洞，确保算法逻辑透明、公平。
- **过程可信：**实现决策链路可追溯，记录 Agent 的任务接收、数据使用、决策依据等关键节点，当出现问题时可回溯至具体环节。
- **结果可信：**确保价值对齐，通过伦理规则的嵌入，让 AI 输出符合人类价值观与法律法规，避免生成有害信息或做出违背伦理的决策。

监管框架：界定权责边界

- **分级分类监管：**根据 AI 应用的风险等级制定差异化规则，低风险场景简化审批流程，高风险场景强化事前评估与事中监控。
- **责任归属明确：**探索 AI 实体的法律责任框架，明确用户、AI 开发者、基础设施供应者等各方的责任划分。

社会形态：从协作到共生

协作方式：从分工到共生

- **角色分工重构：**人类聚焦价值定义与伦理边界，AI 承担规律发现与高效执行，人类不再是生产知识的唯一主体，而是定义要实现什么愿景、并制定不可逾越的伦理规则；AI 则通过数据分析发现隐藏规律、验证实现路径、高效执行落地，反哺人类认知。
- **信任机制建立：**AI 的自进化在人类框架内进行，即人类设定核心原则，AI 在原则内优化流程，同时人类不断调整原则细节，逐步建立人机信任。

职业结构：从替代到创新

- **职业需求变革：**职业形态向高价值、强协同转型，重复性劳动将被 AI 替代，创意设计师、伦理审查员等新兴职业涌现，人类职业将向创造性、战略性和伦理监督方向迁移。
- **超级个体涌现：**也就是媒体常说的一人公司。优先掌握 AI 应用能力的个体成为超级个体，他们无需精通技术，只需学会定义目标、管理 Agent、解读 AI 反馈，即可借助 AI 放大个人价值。

组织模式：从集中化到分布式

- **组织形态轻量化：**一人公司将成为常态，个体创业者无需组建庞大团队，只需通过 AI 平台配置数字员工，即可完成从产品设计到销售的全流程；大型企业则通过“核心团队+数字员工”降低管理成本，快速响应市场变化。
- **先发优势显著化：**率先落地 AI 原生应用的组织将占据先机，这类组织能通过数字员工生态快速迭代业务、降低成本、提升用户体验，拉开与竞争对手的差距。

ASI 的到来不是一蹴而就的，而是技术、场景、治理、社会持续协同进化的结果。走在这条路上，我们既需保持对技术的想象力，也需坚守人类的主导地位，即通过完善的架构设计、场景落地与治理体系，让 AI 安全地成为推动社会进步的核心力量。

AI 原生应用架构-白皮书

AI-NATIVE APPLICATION ARCHITECTURE WHITE PAPER