ADK-Go 框架学习

目录

目录

1. 项目概述

1.1. 项目定义

Agent Development Kit (ADK) for Go 是 Google 开源的一个灵活、模块化的 AI 智能体开发框架，专门为 Go 语言设计。它将软件工程开发原则应用于 AI 智能体创建，旨在简化构建、部署和编排智能体工作流，从简单任务到复杂系统。

虽然它对 Google Gemini 和 Google Cloud 生态系统进行了原生优化，但其设计是**模型无关（Model-agnostic）和部署无关（Deployment-agnostic）**的。

ADK 的核心理念是将 AI 智能体的开发回归到传统的软件工程原则——注重代码的可读性、可测试性、模块化和版本控制。

开源地址: https://github.com/google/adk-go

1.2. 核心概念

- 智能体（agent）：为特定任务设计的基本工作单元。智能体可以使用语言模型（`LlmAgent`）进行复杂推理，或作为执行的确定性控制器，这些被称为"workflow agents"（`SequentialAgent`、`ParallelAgent`、`LoopAgent`）。
- 工具（tool）：赋予智能体超越对话的能力，让它们能够与外部 API 交互、搜索信息、运行代码或调用其他服务。
- 回调（callback）：你提供的在智能体处理过程中特定点运行的自定义代码片段，用于检查、日志记录或行为修改。
- 会话管理（`Session` 和 `State`）：处理单个对话（`Session`）的上下文，包括其历史记录（`Events`）和智能体用于该对话的工作内存（`State`）。
- 记忆（Memory）：使智能体能够在多个会话中回忆用户信息，提供长期上下文（区别于短期会话 `State`）。
- 资源管理（`Artifact`）：允许智能体保存、加载和管理与会话或用户相关的文件或二进制数据（如图片、PDF）。
- 代码执行（Code Execution）：智能体（通常通过工具）生成和执行代码以执行复杂计算或操作的能力。
- 规划（Planning）：一种高级能力，智能体可以将复杂目标分解为更小的步骤，并规划如何实现它们，如 ReAct 规划器。
- 模型：为 `LlmAgent` 提供动力的底层 LLM，支持其推理和语言理解能力。
- 事件（Event）：表示会话期间发生的事情（用户消息、智能体回复、工具使用）的基本通信单元，形成对话历史。
- 运行器（Runner）：管理执行流程的引擎，基于事件协调智能体交互，并与后端服务协调。

注意：多模态流式处理、评估、部署、调试和追踪等功能也是更广泛的 ADK 生态系统的一部分，支持实时交互和开发生命周期。

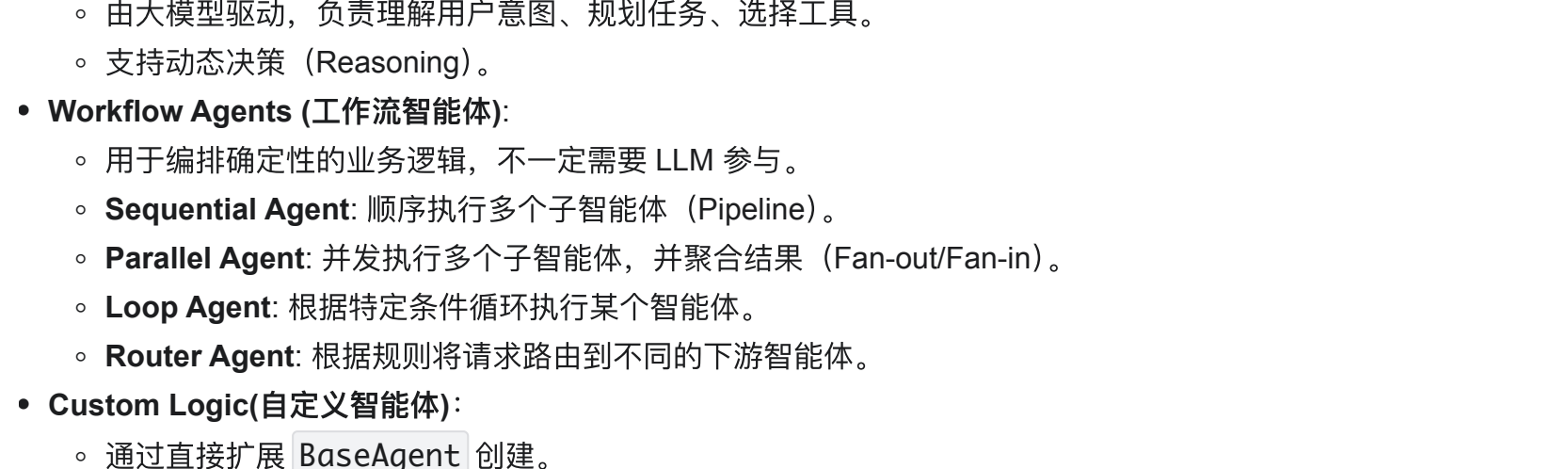
1.3. 核心特点

1. **Idiomatic Go (地道的 Go 风格)**:
   - 充分利用 Go 的接口（Interfaces）、并发（Goroutines/Channels）和强类型系统。
   - 设计上避免了 Python 风格的动态魔法，让代码更易于维护和重构。
2. **Code-First Development (代码优先)**:
   - 所有的逻辑、工具定义、编排流程都通过代码显式定义。
   - **优势**: 方便进行单元测试、代码审查（Code Review）和版本管理。
3. **Multi-Agent Architecture (原生多智能体架构)**:
   - 不仅支持单个 LLM 智能体，还内置了多种**工作流智能体（Workflow Agents）**，如顺序执行、并行执行、循环执行等。
   - 支持层级化设计，一个智能体可以作为另一个智能体的"工具"。
4. **A2A Protocol (Agent-to-Agent 协议)**:
   - **亮点功能**: ADK 定义了一套标准协议，允许智能体之间进行互操作。
   - 支持本地调用，也支持跨网络调用远程智能体。
   - https://a2a-protocol.org/latest/
5. **Rich Tool Ecosystem (丰富的工具生态)**:
   - 内置 Google Search、Code Execution 等工具。
   - 支持 **MCP (Model Context Protocol)**，可轻松集成数据库和其他第三方服务。

2. 架构与核心模块详解 (Architecture & Modules)

ADK 的架构设计非常清晰，主要由以下几个核心模块组成：

2.1. Agents



ADK 中的智能体不仅是调用 LLM 的封装，而是具备特定行为模式的单元。主要分为三类：

- **LLM Agents (推理智能体)**:
  - 由大模型驱动，负责理解用户意图、规划任务、选择工具。
  - 支持动态决策（Reasoning）。
- **Workflow Agents (工作流智能体)**:
  - 用于编排确定性的业务逻辑，不一定需要 LLM 参与。
  - **Sequential Agent**: 顺序执行多个子智能体（Pipeline）。
  - **Parallel Agent**: 并发执行多个子智能体，并聚合结果（Fan-out/Fan-in）。
  - **Loop Agent**: 根据特定条件循环执行某个智能体。
  - **Router Agent**: 根据规则将请求路由到不同的下游智能体。
- **Custom Logic(自定义智能体)**:
  - 通过直接扩展 `BaseAgent` 创建。
  - 允许你实现独特的操作逻辑、特定的控制流程或标准类型无法涵盖的专业集成，满足高度定制的应用程序需求。

特性	LLM 智能体（`LlmAgent`）	工作流智能体	自定义智能体（`BaseAgent` 子类）
主要功能	推理、生成、工具使用	控制智能体执行流程	实现独特逻辑/集成
核心引擎	大型语言模型 (LLM)	预定义逻辑（顺序、并行、循环）	自定义代码
确定性	非确定性（灵活）	确定性（可预测）	可以是任一种，取决于实现
主要用途	语言任务、动态决策	结构化流程、编排	定制需求、特定工作流程

虽然每种智能体类型都有其独特的用途，但真正的力量往往来自于它们的组合。复杂的应用程序经常采用Multi Agents，其中：

- **LLM 智能体**处理智能的、基于语言的任务执行。
- **工作流智能体**使用标准模式管理整体流程。
- **自定义智能体**提供独特集成所需的专业功能或规则。

2.2. Tools

工具是智能体与外部世界交互的桥梁。

- **定义方式**: 通过 Go 的结构体和函数标签定义，ADK 会自动生成 JSON Schema 供 LLM 理解。
- **内置工具**:
  - `GoogleSearch`: 联网搜索能力。
  - `CodeExecutor`: 安全的沙箱代码执行环境。
- **自定义工具**: 任何 Go 函数都可以被包装成工具。

2.3. A2A (Agent-to-Agent)

https://a2a-protocol.org/latest/

2.3.1. 定义

简单来说，**A2A 协议就是 AI 时代的"微服务架构"标准。**

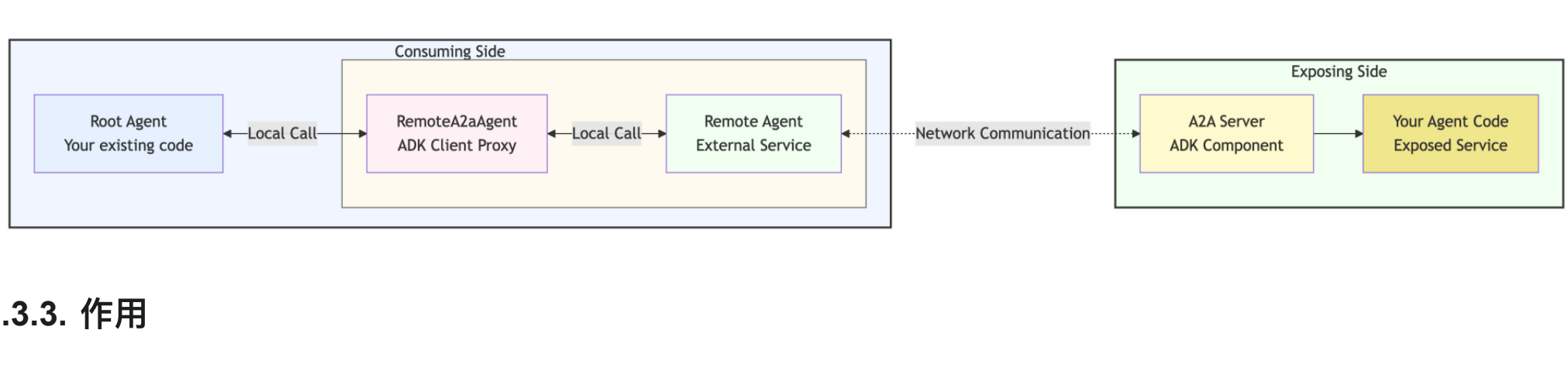
当你构建更复杂的智能体系统时，你会发现单个智能体通常是不够的。你会想要创建专门的智能体，它们可以协作解决问题。**A2A 协议**是允许这些智能体相互通信的标准。

- **Local Sub-Agents（本地子智能体）：**这些智能体与你的主智能体*在同一个应用程序进程中*运行。它们就像内部模块或库，用于将你的代码组织成逻辑的、可重用的组件。主智能体与其本地子智能体之间的通信非常快，因为它直接在内存中发生，没有网络开销。
- **Remote Agents (A2A)：**这些是作为独立服务运行的智能体，通过网络进行通信。A2A 定义了这种通信的标准协议。

2.3.2. 工作流

1. **暴露智能体服务(Exposing)：**你从一个现有的 ADK 智能体开始，你希望其他智能体能够与之交互。ADK 提供了一种简单的方法来"暴露"这个智能体，将其转换为 **A2AServer**。这个服务器充当公共接口，允许其他智能体通过网络向你的智能体发送请求。将其想象为为你的智能体设置一个网络服务器。
2. **请求智能体服务(Consuming)：**在单独的智能体（可能在同一台机器或不同的机器上运行）中，你将使用一个特殊的 ADK 组件，称为 `RemoteA2aAgent`。这个 `RemoteA2aAgent` 充当一个客户端，知道如何与你之前暴露的 **A2AServer** 通信。它在幕后处理网络通信、身份验证和数据格式化的所有复杂性。

ADK 提供了一种简单的方法来"暴露"这个智能体，将其转换为 **A2AServer**。这个服务器充当公共接口，允许其他智能体通过网络向你的智能体发送请求。



2.3.3. 作用

1. AI 领域的"微服务化"

- **痛点**：传统的做法是写一个"上帝智能体（God Agent）"，给它塞几万字的 Prompt 和几十个工具，导致推理变慢、容易产生幻觉、难以维护。
- **解法**：允许你将复杂的任务拆解为多个**小而专**的智能体。

2. 分布式与跨语言调用

- **痛点**：Eino 或 LangChain 通常是在**同一个进程**内通过函数调用连接模块。如果你的 `搜索Agent` 跑在 Python 服务上，而 `业务Agent` 跑在 Go 服务上，怎么办?
- **解法**：A2A 定义了一套基于 HTTP/GRPC 的标准通信协议。
  - **作用**：Go 编写的主控 Agent 可以通过网络直接调用 Python 编写的数据分析 Agent，或者调用运行在另一台服务器上的 Agent。这使得**跨团队协作**开发 AI 应用成为可能。

3. 统一的输入输出标准

- **痛点**：Agent A 输出的是 Markdown，Agent B 需要的是 JSON，对接起来非常痛苦。
- **解法**：强制定义了 Agent 之间交互的 Schema（契约）。
  - **作用**：类似于 API 文档（OpenAPI/Swagger）。A2A 规定了握手、任务下发、结果返回、错误处理的标准格式。这保证了即使更换了底层的模型（从 Gemini 换到 DeepSeek），只要 A2A 接口不变，整个系统依然能稳定运行。

2.4. MCP

模型上下文协议 (MCP) 是一种开放标准，旨在标准化大型语言模型 (LLM)（如 Gemini 和 Claude）与外部应用程序、数据源和工具的通信方式。可以将其视为一种通用连接机制，简化了 LLM 获取上下文、执行操作和与各种系统交互的方式。

MCP 遵循客户端 - 服务器架构，定义了**数据**（资源）、**交互模板**（提示）和**可执行函数**（工具）如何由 **MCP 服务器**公开并由 **MCP 客户端**（可能是 LLM 主机应用程序或 AI 智能体）使用。

ADK涵盖了两种主要的集成模式：

1. **在 ADK 中使用现有 MCP 服务器：** ADK 智能体充当 MCP 客户端，利用外部 MCP 服务器提供的工具。
2. **通过 MCP 服务器暴露 ADK 工具：** 构建一个包装 ADK 工具的 MCP 服务器，使其可被任何 MCP 客户端访问。

2.5. Runtime & Deployment (运行时与部署)

- **Agent Runtime**: 提供了标准化的运行环境，处理输入/输出流、状态管理和错误处理。
- **Observability**: 集成了 OpenTelemetry，支持日志记录、链路追踪（Tracing），方便调试。
- **Deployment**:
  - 可以编译为二进制文件在本地运行。
  - 原生支持 **Google Cloud Run** 和 **GKE (Kubernetes)** 部署。
  - 提供了 **ADK Web UI**，一个可视化的调试界面，用于查看对话历史、工具调用链和性能指标。

2.6. Session & State (会话与状态)

- **Session**: 代表与用户的一次持续对话，包含唯一的 Session ID。
- **State**: 提供了内存管理机制，支持短期记忆（当前对话上下文）和长期记忆（通过向量数据库或外部存储）。
- **Context Caching**: 针对 Gemini 模型优化，支持上下文缓存以降低长 Context 的推理成本。

3. 智能体框架对比、选型

ADK-Go vs. ByteDance Eino

字节跳动的 **Eino (CloudWeGo)** 也是Go语言的智能体框架。以下是两者的深度对比：

维度	Google ADK for Go	ByteDance Eino (CloudWeGo)
核心设计哲学	**Agent-Centric (智能体中心)** 强调"智能体"作为独立个体，关注智能体之间的协作、协议 (A2A) 和行为模式。	**Graph/Pipeline-Centric (图编排中心)** 强调"组件"和"流"，将 LLM 应用看作一个有向无环图 (DAG) 的数据处理流。
编排方式	**Workflow Agents** 通过代码组合不同的 Agent 类型（Sequential, Parallel）来构建逻辑，更像写普通代码。	**Graph Engine** 提供了一套强大的图编排引擎，节点 (Node) 和边 (Edge) 概念明确，自动处理复杂的并发和数据流转。
流式处理 (Streaming)	**Go Iterators** 使用 Go 1.23+ 的标准迭代器处理流，简洁标准。	**StreamReader/Writer** 支持流的自动合并、复制、转换，对高并发流式处理优化极致。
互操作性	**A2A Protocol** 原生支持智能体之间的远程调用协议，适合构建分布式的多智能体系统。	**Component Interface** 主要通过标准化的 Go 接口（如 `ChatModel`，`Retriever`）在进程内组合，微服务化依赖 CloudWeGo 生态。
生态集成	**Google Ecosystem** Gemini, Vertex AI, Google Search, MCP 深度集成。	**Enterprise Production** 针对企业级生产环境优化，集成 Langfuse 观测，强调稳定性、高并发 (QPS) 和低内存泄漏。
上手难度	**中等 (Flexible)** 概念较少，更贴近原生 Go 开发体验。	**稍高 (Structured)** 有一套较为严格的组件定义和图编排 DSL，学习曲线略陡峭，但规范性更强。

对比总结与选型建议

- **选择 Google ADK (Go) 如果：**
  - 你主要构建 **"智能体" (Agents)** 应用，需要复杂的推理和自主决策能力。
  - 你需要 **Multi-Agent** 协作，特别是涉及分布式、跨服务的智能体交互 (A2A)。
  - 你深度依赖 **Google Cloud / Gemini** 生态。
  - 你喜欢轻量级、代码优先的开发模式，不希望引入过重的编排引擎。
- **选择 ByteDance Eino 如果：**
  - 你主要构建 **"RAG / 工作流"** 应用，业务逻辑是确定的流水线。
  - 你面临 **极高的并发要求** 和 **生产环境稳定性** 挑战（Eino 经过字节内部大规模验证）。
  - 你需要极致的 **流式处理 (Streaming)** 能力（如打字机效果的精准控制、多路流合并）。
  - 你更倾向于 **结构化、规范化** 的企业级开发框架，防止代码随业务增长而失控。

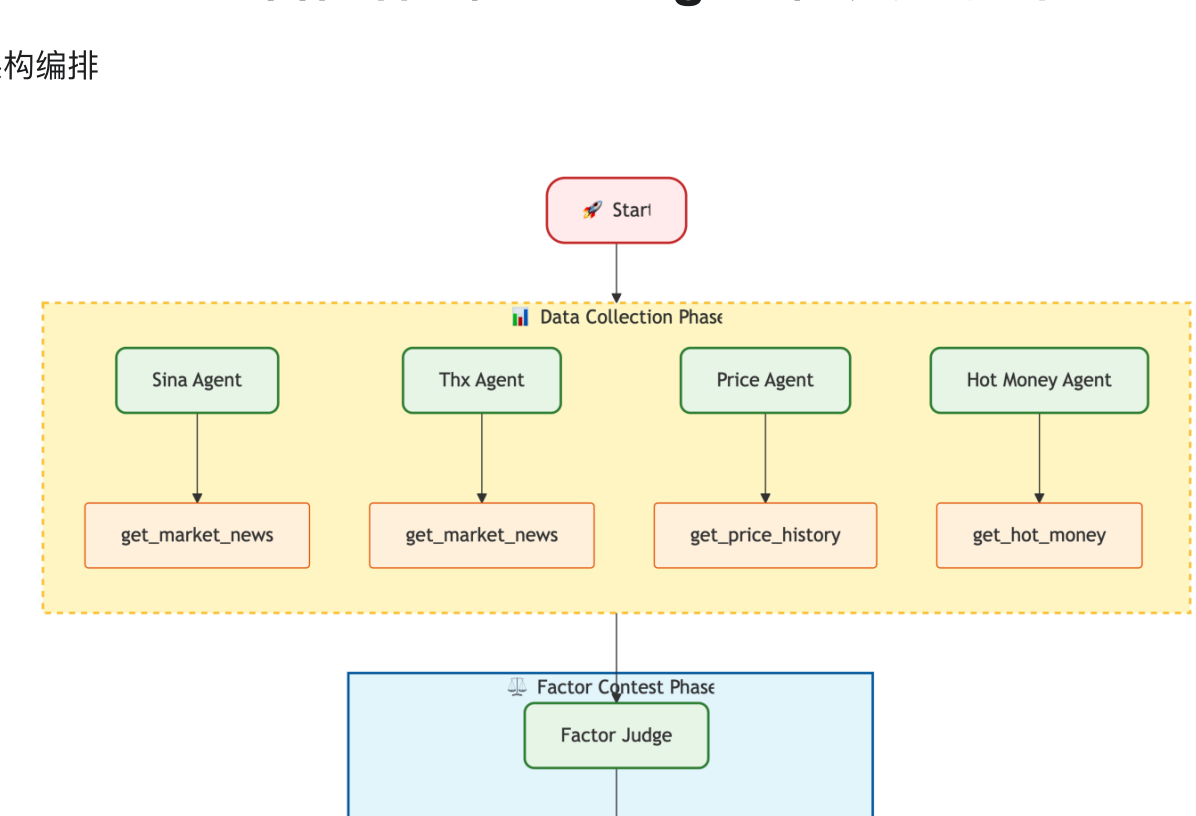
4. 总结

Google ADK for Go 结合 Go 的语言特性，提供了一套**简洁、高性能且具备强大互操作性**的解决方案。

对于希望在 Go 环境中引入 GenAI 能力的团队，ADK 提供了一个从原型到生产的坚实基础，特别是其 **A2A 协议** 为未来的分布式智能体网络通过了无限可能。

5. demo-多智能体（Multi-Agent）交易框架

架构编排



📊 Phase 1: 数据挖掘团队 (ParallelAgent)

- 并发执行：4 个 Agent 同时工作，互不干扰，极大提高了效率。
- 分工明确：
- sina/thx：负责基本面消息（API: 新浪/同花顺）。
- price：负责技术面价格趋势（API: 东方财富 K线）。
- hot_money（本次新增）：负责情绪面，抓取连板池、概念涨幅榜（API: 东方财富热榜）。
- 特点：每个 Agent 都是一个独立的 LLM，拥有调用特定 Tool 的能力。

⚖️ Phase 2: 因子裁判 (Factor Judge)

- 角色：首席数据官 (CDO)。
- 机制：这是第一个 "Contest (竞赛)" 节点。它接收 Phase 1 所有 Agent 的输出，去除噪音（如无效新闻），识别当天市场的主导逻辑（例如：是宏观利空主导，还是游资炒作主导？），最终生成一份权威的 Daily Market Brief。

🧠 Phase 3: 策略研究团队 (ParallelAgent)

- 并发执行：基于同一份 Daily Market Brief，不同的策略 Agent 并行思考。
- 多样性 (Beliefs)：
- 激进派：通过 Prompt 注入 "追逐高波动、事件驱动" 的信念。
- 保守派：通过 Prompt 注入 "低风险、高分红" 的信念。
- 目的：避免单一思维模型，通过多视角博弈来覆盖市场可能得机会与风险

🏆 Phase 4: 最终决策 (CIO)

- 角色：首席投资官 (CIO)。
- 机制：这是第二个 "Contest" 节点。CIO 像人类基金经理一样，看着两份截然不同的投资提案，结合当前的市场环境（Brief），做出最终的资产配置决定（例如：50% 蓝筹防守 + 30% 热点进攻 + 20% 现金）。

```go
// 1. Model Initialization
model, err := gemini.NewModel(ctx, "gemini-2.0-flash-001", &genai.ClientConfig{
    APIKey: os.Getenv("GOOGLE_API_KEY"),
})
if err != nil {
    log.Fatalf("Failed to create model: %v", err)
}

// 2. Tools

getNewsTool, _ := functiontool.New(functiontool.Config{
    Name:        "get_market_news",
    Description: "Fetches the latest market news from Sina Finance.",
}, getMarketNews)

getPriceTool, _ := functiontool.New(functiontool.Config{
    Name:        "get_price_history",
    Description: "Fetches recent price history for Shanghai Index from EastMoney.",
}, getPriceHistory)

getHotMoneyTool, _ := functiontool.New(functiontool.Config{
    Name:        "get_hot_money",
    Description: "Fetches current hot concepts and sector performance from EastMoney.",
}, getHotMoney)

saveReportTool, _ := functiontool.New(functiontool.Config{
    Name:        "save_report",
    Description: "Saves the given content to a file. Useful for saving final reports.",
}, saveReport)

tools := []tool.Tool{getNewsTool, getPriceTool, getHotMoneyTool, saveReportTool}

// 3. Phase 1: Data Agents (Parallel)

sinaAgent, err := createDataAgent(model, "sina_summary_agent", "Focus on general market news and policy updates.", tools)
if err != nil {
    log.Fatal(err)
}

thxAgent, err := createDataAgent(model, "thx_summary_agent", "Focus on industry-specific news and corporate actions.", tools)
if err != nil {
    log.Fatal(err)
}

priceAgent, err := createDataAgent(model, "price_market_agent", "Focus on price trends, volatility, and technical indicators.", tools)
if err != nil {
    log.Fatal(err)
}

hotMoneyAgent, err := createDataAgent(model, "hot_money_agent", "Focus on hot concepts, limit-up stocks, and market sentiment.", tools)
if err != nil {
    log.Fatal(err)
}

dataTeam, err := parallelagent.New(parallelagent.Config{
    AgentConfig: agent.Config{
        Name:        "data_analysis_team",
        Description: "Generates data factors in parallel.",
        SubAgents:   []agent.Agent{sinaAgent, thxAgent, priceAgent, hotMoneyAgent},
    },
})
if err != nil {
    log.Fatal(err)
}

// 4. Phase 2: Factor Evaluation (Sequential - The "Contest" Judge)
factorJudge, err := llmagent.New(llmagent.Config{
    Name:        "factor_judge",
    Model:       model,
    Description: "Evaluates and synthesizes factors.",
    Instruction: fmt.Sprintf(`You are the Chief Data Officer.
Review the outputs from 'sina_summary_agent', 'thx_summary_agent', 'price_market_agent', and 'hot_money_agent'.
Identify the most compelling signal among them.
Synthesize a 'Daily Market Brief' that highlights the dominant market driver.
Discard weak or conflicting signals.
Output result in language: %s`, "中文"),
    OutputKey: "daily_market_brief",
})
if err != nil {
    log.Fatal(err)
}

// 5. Phase 3: Research Agents (Parallel)

belief1 := "专注于短期事件驱动机会：优先关注公司公告、并购重组、订单暴增、技术突破等催化事件；偏好中小市值、高波动的题材股，适合激进套利策略。"
researcher1, err := createResearchAgent(model, "research_agent_aggressive", belief1, "daily_market_brief")
if err != nil {
    log.Fatal(err)
}

belief2 := "专注于稳健的确定性事件：关注分红、回购、业绩预告确认、重大合同落地和政策利好等；偏好大盘蓝筹、低波动、确定性高的标的，适合稳健配置。"
researcher2, err := createResearchAgent(model, "research_agent_conservative", belief2, "daily_market_brief")
if err != nil {
    log.Fatal(err)
}

researchTeam, err := parallelagent.New(parallelagent.Config{
    AgentConfig: agent.Config{
        Name:        "research_team",
        Description: "Proposes strategies based on different beliefs.",
        SubAgents:   []agent.Agent{researcher1, researcher2},
    },
})
if err != nil {
    log.Fatal(err)
}

// 6. Phase 4: Final Decision (Sequential - The "Contest" Final Judge)
cio, err := llmagent.New(llmagent.Config{
    Name:        "chief_investment_officer",
    Model:       model,
    Description: "Makes the final portfolio decision.",
    Instruction: fmt.Sprintf(`You are the CIO.
Review the proposals from 'research_agent_aggressive' and 'research_agent_conservative'.
Based on the current market environment (from the Brief), decide which strategy to adopt, or how to blend them.
Output the Final Portfolio Allocation and the rationale.
Output result in language: %s

CRITICAL REQUIREMENT:
You MUST call the 'save_report' tool to save your final decision to a file named 'final_portfolio.md'.
The content of the report MUST be in CHINESE (Simplifed Chinese).
Do NOT output the full report in the conversation; only output a confirmation message like "Report saved to final_portfolio.md".`, "中文"),
    Tools: tools, // Give CIO access to tools so it can use save_report
})
if err != nil {
    log.Fatal(err)
}

// 7. Orchestrator
contestTradeSystem, err := sequentialagent.New(sequentialagent.Config{
    AgentConfig: agent.Config{
        Name:        "contest_trade_system",
        Description: "Full ContestTrade pipeline.",
        SubAgents:   []agent.Agent{dataTeam, factorJudge, researchTeam, cio},
    },
})
if err != nil {
    log.Fatal(err)
}

// 8. Launcher
config := &launcher.Config{
    AgentLoader: agent.NewSingleLoader(contestTradeSystem),
}

l := full.NewLauncher()
if err = l.Execute(ctx, config, os.Args[1:]); err != nil {
    log.Fatalf("Run failed: %v\n\n%s", err, l.CommandLineSyntax())
}
```

项目地址：https://github.com/google/adk-go

接口文档： https://pkg.go.dev/google.golang.org/adk

项目文档：https://google.github.io/adk-docs/get-started/go/