

T-Agent——基于 Spring AI Alibaba 的城市交通治理智能体

1.项目概览

面对超大城市路网复杂时空挑战，传统“经验驱动”的交通治理模式已难以匹配**城市治理现代化、应急管理精细化的政务需求**。本项目立足**智慧城市建设、公共安全保障的政策导向**，利用 Spring AI Alibaba 框架，构建具备“全域感知”与“自主决策”能力的交通治理智能体（T-Agent），以 AI 重构交通政务业务流程，助力城市交通从“被动管理”向“主动智理”的代际跨越，**为政府交通治理提供数字化、智能化的核心工具**。

1.1 核心痛点与解决方案

针对传统交通系统存在的“外部盲区大、查询门槛高、应急响应慢、隐患挖掘浅”四大痛点，我们设计了四个核心 Agent 场景解决痛点：

1.1.1 事前·全域研判

痛点： 传统预测仅靠历史流量线性外推，无法感知外部“黑天鹅”事件。

解决： 联网搜索 Agent。自动抓取互联网活动日程与气象数据，融合内部路网拓扑，生成包含“高风险路段”的综合预警报告。

1.1.2 事中·即时感知

痛点： 指挥人员不懂 SQL，查数据需排队等技术人员，决策滞后。

解决： 数据查询 Agent。业务人员用自然语言（如“查2月1日拥堵情况”）直接驱动数据库，毫秒级生成报表，实现数据平权。

1.1.3 事中·动态指挥

痛点： 应急预案（SOP）是静态 PDF，事故现场查找困难，极易导致决策失误。

解决： 应急响应 Agent。基于 RAG（检索增强生成）将静态文档转化为动态、可执行的“行动清单”（如：封锁路口、调度救护车），辅助快速决策。

1.1.4 事后·隐患挖掘

痛点： 传统报表只有统计数字，无法定位空间上的“事故黑点”。

解决： 数据分析 Agent。自动编写代码进行 DBSCAN 空间聚类，精准识别事故高发区域并生成归因分析。

1.2 技术创新点

1.2.1 DeepResearch 驱动的全域感知 Agent

突破传统模型“无法感知当下”的限制，融合 ReAct 推理链与联网深度搜索能力。

自主规划：Agent 构建“思考-行动-观察”闭环，自主拆解复杂问题。

外部感知：遇到未知变量（如未来交通风险）时，自动触DeepResearch，实时抓取互联网公开情报（气象、舆情、活动），实现“内部数据+外部情报”的逻辑互证与全域研判。

1.2.2 SQL + 向量双引擎混合检索 (Hybrid RAG)

构建了“结构化数据 (NL2SQL) + 非结构化知识 (Vector)”的双通道检索架构。既能通过 SQL 精准提取毫秒级交通指标，又能通过向量检索调用静态 SOP 预案。系统自动评估问题类型，动态路由并融合两类数据源，彻底解决传统 RAG 只能查文档、无法查业务数据的痛点。

1.2.3 全链路流式交互体验 (Streaming UX)

针对 DeepResearch 和复杂分析场景带来的长耗时挑战，采用 SSE (Server-Sent Events) + React 增量渲染技术。将 Agent 的“思考路径”（如：正在搜索天气...、正在聚合数据...）实时推送到前端。变“黑盒等待”为“白盒展示”，让用户直观看到 AI 如何抽丝剥茧解决问题，极大提升了系统的可解释性与用户信任度。

1.3 项目价值

本项目不是简单的技术验证，而是交通治理数字化转型的实战演练。它打破了数据与决策之间的壁垒，不仅提升了指挥中心的响应效率，更通过主动风险防御，切实保障城市交通安全。

2. 业务场景与痛点分析

为验证系统处理复杂时空任务的能力，我们将业务场景具象化为四个核心指令，分别对应交通治理周期的预测（Risk Assessment）、感知（Traffic Query）、处置（Emergency Response）与复盘（Safety Review）四个环节。以下是针对验证环境选型及四大指令的深度剖析：

2.1 验证场景：为何选择纽约曼哈顿数据？

为了平衡“验证高难度场景”与“确保赛期内项目落地”，我们制定了以下选型与执行策略：

2.1.1 验证靶场：为何选择纽约曼哈顿（2024.02）？

我们特意选取纽约曼哈顿在2024年2月的真实数据作为测试集，核心考量如下：

- a. **极限压力测试**: 曼哈顿拥有全球最密集的“棋盘式”路网，且2月涵盖暴雪极端天气与春节/情人节等高频社会活动，是验证 AI 处理复杂时空关系与外部干扰（DeepResearch）的绝佳样本。
- b. **全周期闭环验证**: 选取完整的历史月份，使我们拥有了从“月初预测”到“月末实际事故”的完整真值，从而能客观量化系统在预测与复盘环节的准确率
- c. **数据完备性**: 依托 NYC Open Data，拥有出租车、Citibike、交通事故及气象等多模态真实数据，确保项目并非“空对空”演示，而是基于真实物理世界的实战复盘。

2.1.2 MVP 可行性策略：为何本项目能在大赛中落地？

考虑到比赛周期限制，我们严格遵循 MVP（最小可行性产品）原则，确保系统“小而美且可用”：

- a. **数据零门槛**: 依托 NYC Open Data 亿级公开数据（出租车/事故/气象），无版权争议与获取壁垒，确保项目不卡在数据供给侧。
- b. **业务做减法**: 我们不追求大而全的通用系统，而是聚焦“预测-感知-处置-复盘”这四个最核心的交通治理指令，集中资源打通最小业务闭环。
- c. **技术高复用**: 基于 Spring AI Alibaba 成熟框架，利用 Java 强大的生态兼容性，避免了从零造轮子，确保在短时间内构建出企业级稳定的演示系统。

2.2 场景一：基于多模态的趋势研判（事前）

User Instruction: “Risk Assessment: 请对曼哈顿地区2024年2月的交通状况进行风险评估”

传统痛点: 被动式规划与信息盲区面对“评估未来一个月交通风险”这类宏观指令，传统系统的痛点在于维度的单一性。传统的交通预测模型通常仅依赖历史同期的路网流量数据（如去年3月的车流），属于“线性外推”。然而，交通是一个开放的复杂巨系统。传统模型无法感知“2024年3月”这个时间窗口内外部变量的干扰，例如：是否有联合国大会召开？气象局是否预报了连续暴雪？曼哈顿是否有大型马拉松赛事？这些非结构化的外部信息往往是导致交通瘫痪的“黑天鹅”，但它们存在于互联网新闻、气象报告或政府公文中，是传统数据库的盲区。

Agent 突破: DeepResearch 驱动的全域感知 Agent 接收指令后，不再局限于内部数据库，而是启动 DeepResearch（深度搜索）能力：

- a. **外部情报搜集**: 联网检索曼哈顿 2024年3月的大型活动日程表、施工通告及长期气象预报。
- b. **多源数据融合**: 将外部事件的时间地点映射到路网拓扑中，结合历史流量基线。
- c. **生成评估报告**: 最终输出一份包含“高风险拥堵路段预测”、“重点关注日期”及“诱导建议”的综合风险评估报告。

2.3 场景二：基于自然语言驱动的即时全景感知（事中）

User Instruction: “Traffic Query: 帮我查询2024年2月1号的交通情况”

传统痛点：高门槛的数据查询壁垒 对于指挥中心的非技术人员来说，要查询“2024年2月1号”这样特定时间段的全量交通情况，通常面临操作复杂的痛点。传统方式需要操作员在复杂的 BI 仪表盘中反复筛选时间控件、区域控件，或者向技术部门提需求编写 SQL 代码来提取平均车速、拥堵指数等指标。这导致决策者与数据之间隔着一道“技术墙”，获取信息的时延往往以小时计，无法满足即时指挥需求。

Agent 突破：NL2SQL 实现的数据平权 Spring AI Alibaba 架构下的 Agent 具备强大的 Text-to-SQL (NL2SQL) 能力：

- a. 语义解析：精准识别“2024年2月1号”为时间过滤条件，“交通情况”对应数据库中的 traffic_speed、congestion_level 等字段。
- b. 自动执行：自动生成并在数仓中执行 SQL 语句，毫秒级返回当日的流量热力图与核心指标摘要。让不懂代码的业务人员也能通过对话，瞬间获取上帝视角的数据支撑。

2.4 场景三：基于动态知识库驱动的应急指挥（事中）

User Instruction: “Emergency Response: 请查找2024年2月1日的交通事故，并提供相对应的应急救援方案”

传统痛点：预案脱节与决策滞后 当系统检索到交通事故后，核心痛点在于从“发现问题”到“解决问题”的断层。传统的应急响应中，调度员查看到事故后，需要人工去翻阅厚重的《交通应急管理手册》(PDF/纸质)，凭记忆寻找对应的处置流程（是封锁车道？还是全线分流？）。在分秒必争的事故现场，这种依赖人工检索 SOP (标准作业程序) 的方式极易因紧张或经验不足导致决策失误，甚至引发二次事故。

Agent 突破：RAG 驱动的动态决策辅助该指令触发了 Agent 的 RAG 机制：

- a. 精准定位：首先通过数据库锁定 2 月 1 日的具体事故类型（如：追尾、危险品泄漏）。
- b. 知识挂载：Agent 立即在向量数据库中检索对应的应急预案文档片段。
- c. 方案生成：结合事故发生的具体地点（如：位于单行道还是十字路口），Agent 将静态的 SOP 文档转化为动态的、可执行的行动清单（例如：“检测到第五大道事故，建议：1. 调度最近的西区医院救护车；2. 封闭第42街入口；3. 调整上游红绿灯为全红 60 秒”）。

2.5 场景四：基于大模型的智能归因与隐患挖掘（事后）

User Instruction: “Safety Review: 帮我复盘一下2024年2月的交通安全整体情况，识别出事故黑点区域”

传统痛点：数据沉睡与表层统计 在月度安全复盘时，痛点在于数据资产无法转化为治理智慧。传统报表只能提供简单的统计图（如：事故总数 50 起），但难以回答“哪里是事故高发黑点？”、“为什么这个月事故率上升？”等深层问题。要识别“事故黑点”，通常需要 GIS 专家进行专业的空间聚类分析，普通业务系统难以自动完成，导致安全隐患整改缺乏靶向性。

Agent 突破：Code Interpreter 驱动的深度分析 面对复盘指令，Agent 不仅是查询者，更是数据分析师：

- a. 全量扫描：拉取 2024年2月的所有事故记录（坐标、时间、类型）。
- b. 空间聚类：利用内置的代码解释器（Code Interpreter）或算法模型，对事故坐标进行 DBSCAN 聚类分析，自动识别出空间上的“事故黑点”。
- c. 归因洞察：结合该区域的路网结构，给出分析结论（如：“时代广场周边在雨天事故率飙升20%，识别为事故核心黑点，建议增加防滑铺装”），直接为管理者的月度汇报提供核心素材。

2.6 总结：从“治理”到“智理”的跨越

通过上述对四个 User Instruction 的拆解，本项目清晰地界定出 Spring AI Alibaba 赋能下的业务价值：

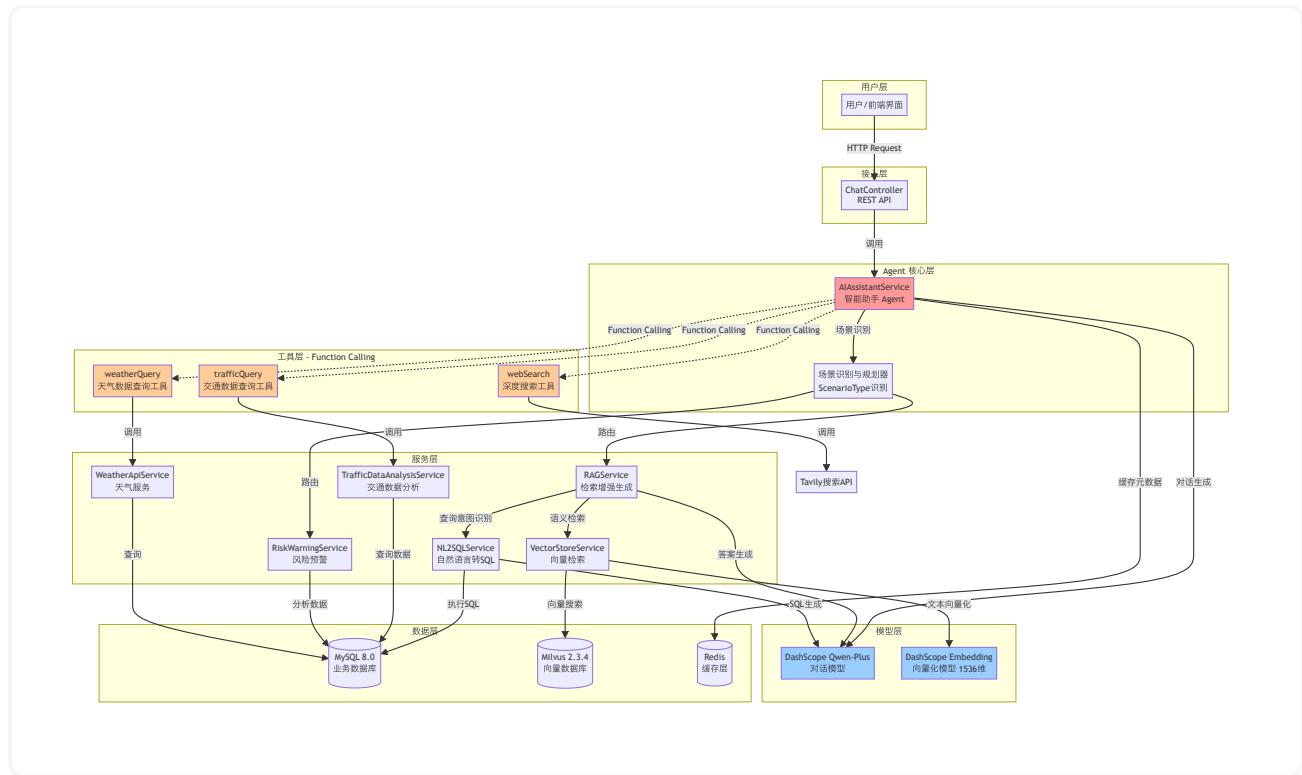
- a.Traffic Query 解决了常规问答中数据获取的效率问题（NL2SQL）。
- b.Risk Assessment 解决了事前对于未知外部环境的盲区问题（DeepResearch）。
- c.Emergency Response 解决了事中对于交通事故预案的执行时效问题（RAG）。
- d.Safety Review 解决了事后鉴于往期交通情状整体提取并进行隐患挖掘的深度问题（Data Analysis Agent）。

综合来看，该项目不仅是一个技术验证，更是一次将交通治理从“经验驱动+人工查询”向“数据驱动+AI决策”转型的实战演练。

3.系统架构设计

3.1 系统架构总览

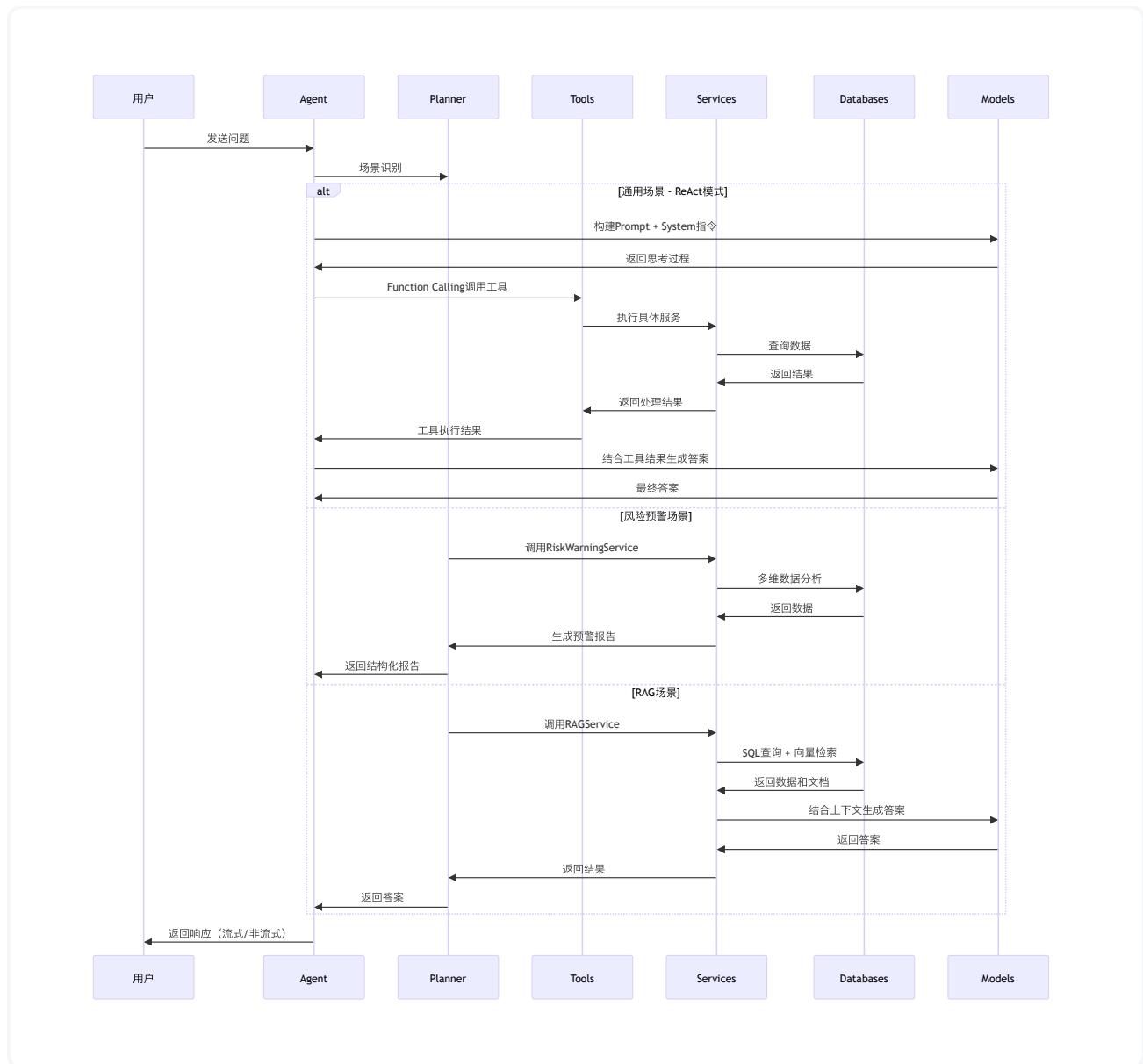
3.1.1 整体架构图



该架构图展示了 T-Agent 系统从用户请求到数据响应的完整流程。核心特点包括：

- ReAct 模式实现**: Agent 通过 Function Calling 机制自主选择和调用工具
- 多工具协同**: trafficQuery、weatherQuery、webSearch 三大工具协同工作
- 混合检索策略**: 结合 SQL 查询（结构化数据）和向量检索（非结构化知识）
- 双模型支持**: 对话模型（Qwen-Plus）+ 向量化模型（DashScope Embedding）

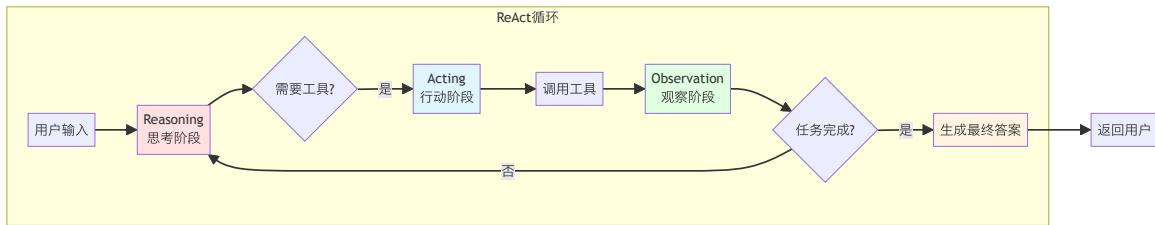
3.1.2 数据流转架构图



数据流说明：

- 场景路由**: 系统首先识别用户问题属于哪个场景（通用/风险预警/应急响应/数据治理）
- ReAct 循环**: 在通用场景下，Agent 进入"思考-行动-观察"循环，自主决策调用哪些工具
- 混合检索**: RAG 服务同时使用 SQL 查询和向量检索，确保答案的准确性和完整性
- 流式响应**: 支持 SSE 流式输出，提升用户体验

3.1.3 ReAct 模式执行流程图



a. ReAct循环机制：

- i. 从用户输入开始，先进入思考阶段（Reasoning），分析当前任务是否需要外部工具支持
- ii. 若需要，则进入行动阶段（Acting），调用相应工具（如数据库、API等）

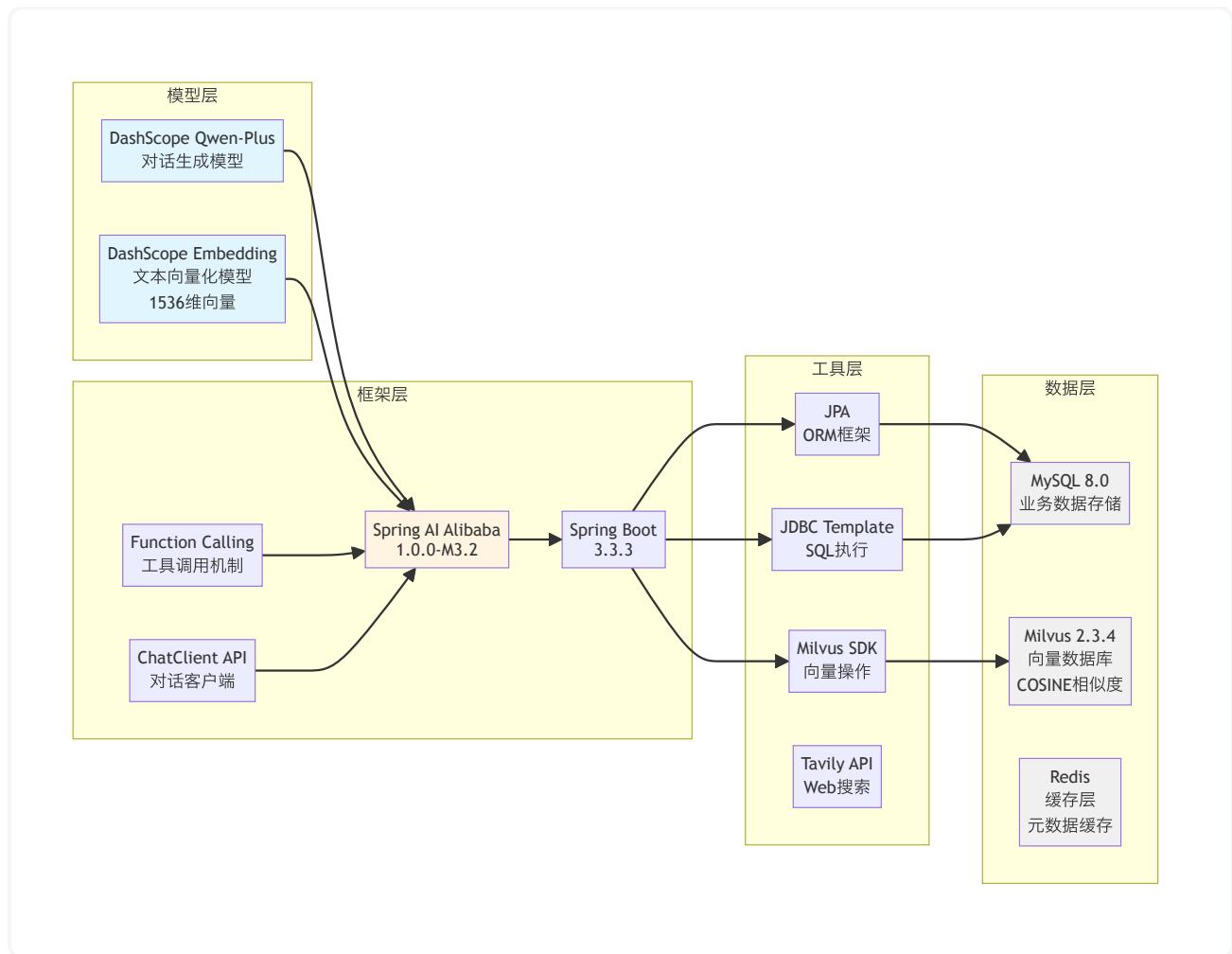
b. 观察与迭代：

- i. 工具调用后进入观察阶段（Observation），获取执行结果
- ii. 系统判断任务是否完成：若未完成，返回思考阶段继续推理；若已完成，则生成最终答案

c. 答案输出：最终答案在任务确认完成后生成，并返回给用户

3.2 技术栈详解

3.2.1 技术栈全景图



3.2.2 技术栈清单

3.2.2.1 模型层

a. DashScope Qwen-Plus

- i. 用途：主对话模型，负责理解用户意图、生成回答、执行 Function Calling
- ii. 配置：TopP=0.8, Temperature=0.7
- iii. 特点：支持中文优化，上下文理解能力强

b. DashScope Embedding Model

- i. 用途：文本向量化，支持语义检索
- ii. 维度：1536 维向量
- iii. 度量：COSINE 余弦相似度

3.2.2.2 框架层

组件	技术选型	版本	核心功能
AI 框架	Spring AI Alibaba	1.0.0-M3.2	统一 AI 调用接口

Function Calling	Spring AI Function	-	工具注册与自动调用
ChatClient	Spring AI ChatClient	-	流式对话、Prompt 管理、简化对话模型调用
后端框架	Spring Boot	3.3.3	应用基础框架，支持依赖注入、自动配置、Web 服务等基础能力
数据访问	Spring Data JPA	-	ORM 映射
响应式编程	Project Reactor	-	支持流式响应

3.3.2.3 数据层

a. 关系型数据库 (MySQL 8.0)

数据表	记录数	用途	关键字段
nyc_traffic_accidents	1000+	交通事故数据	CRASH DATE, CRASH TIME, borough, NUMBER OF PERSONS INJURED
subway_ridership	5000+	地铁客流数据	station_complex, ridership, transit_timestamp
nyc_permitted_events	500+	许可活动数据	Event Name, Start Date/Time, Event Borough
citibike_trips_202402	10000+	共享单车数据	start_station_name, started_at, ended_at
complaints	2000+	城市投诉数据	complaint_type, status, created_at
weather_data	29	天气数据	date, temperature, snow, precipitation
chat_history	动态	对话历史	session_id, user_message, assistant_message

b. 向量数据库 (Milvus 2.4.x)

集合名称	向量维度	索引类型	度量方式	用途
knowledge_base	1536	IVF_FLAT	COSINE	SOP文档、专家知识、治理经验

i. 字段结构：

- `id` : Int64 (主键, 自增)
- `embedding` : FloatVector (1536维)
- `content` : VarChar (文档内容)
- `category` : VarChar (分类: SOP/专家知识/治理经验)
- `title` : VarChar (标题/来源)

3.3.2.4 缓存层

a. Redis

- i. 用途：缓存 RAG 查询结果、存储元数据（图表、思考过程）

用途	Key 格式	过期时间	说明
RAG 缓存	<code>rag:answer:{hash}</code>	24小时	缓存 RAG 查询结果
会话元数据	<code>session:metadata:{sessionId}</code>	1小时	存储工具调用思考过程
查询结果缓存	<code>query:result:{hash}</code>	6小时	缓存 SQL 查询结果

3.3.2.5 工具层

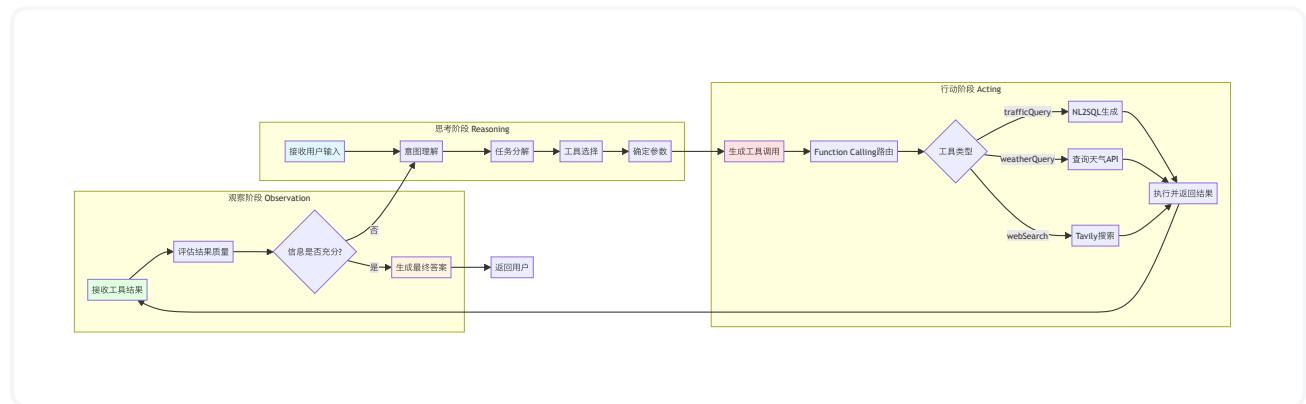
工具名称	实现类	功能描述	调用时机
trafficQuery	TrafficDataAnalysisService	NL2SQL + 数据分析	用户询问交通状况、事故统计
weatherQuery	Weather ApiService	天气数据查询	用户询问天气或天气影响
webSearch	TavilySearchService	互联网实时搜索	本地数据不足或需要最新信息

3.3 Agent 设计详解

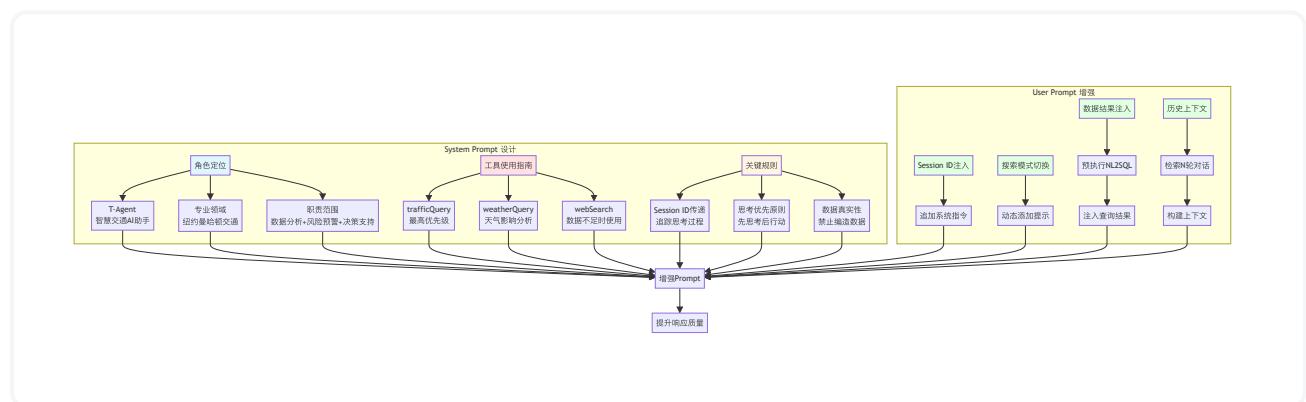
3.3.1 ReAct 模式实现

T-Agent 采用ReAct模式，通过设计 Prompt 让 AI 具备"思考-行动-观察"的循环能力，实现自主规划和工具调用。

3.3.1.1 执行流程详解



3.3.1.2 核心 Prompt 设计



a. 关键设计点：

- 明确工具优先级：**通过“必须优先使用”等强调词，引导模型正确选择工具
- 参数传递规范：**明确要求传递 Session ID，确保思考过程可追踪
- 思考链引导：**要求模型“先思考再行动”，形成 Reasoning → Acting 循环
- 数据验证机制：**要求模型验证工具返回结果，避免盲目信任

3.3.1.3 多轮ReAct示例

- 场景：** 用户问题：“分析2024年2月15日的交通风险，并给出建议”
- 执行流程：**



c. 详细步骤：

i. 天气数据收集

- **Thought:** 分析交通风险需要多维度数据，首先获取天气信息
- **Action:** 调用 weatherQuery 工具，参数为 2024-02-15
- **Observation:** 返回天气数据 - 暴雪，温度-5°C，道路结冰风险高

ii. 交通数据收集

- **Thought:** 暴雪天气会显著影响交通，需要查看当天实际交通情况
- **Action:** 调用 trafficQuery 工具，查询 2024年2月15日的事故数据
- **Observation:** 返回事故统计 - 当天发生45起事故，比平时增加80%

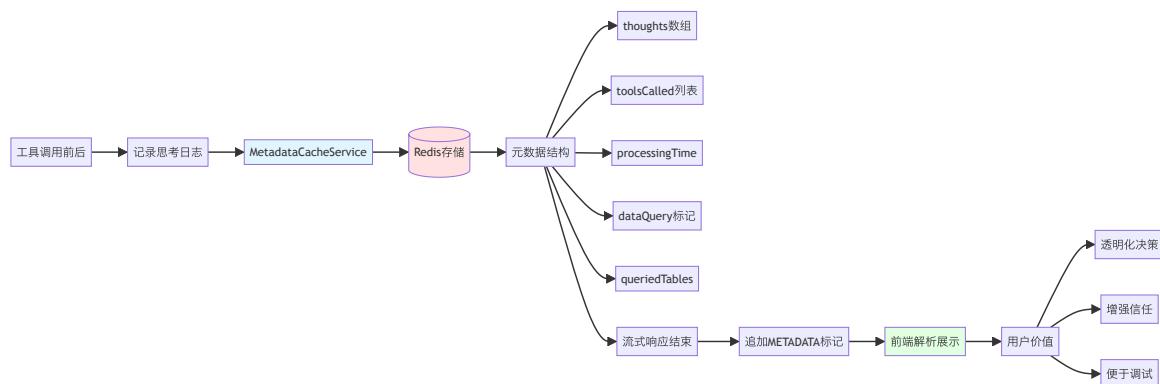
iii. 风险评估与报告生成

- **Thought:** 综合天气和交通数据，显示高风险状态，需要生成正式预警报告
- **Action:** 内部调用 RiskWarningService.generateRiskWarning
- **Observation:** 生成一级风险预警报告，包含高风险区域和应急建议

iv. 综合回答

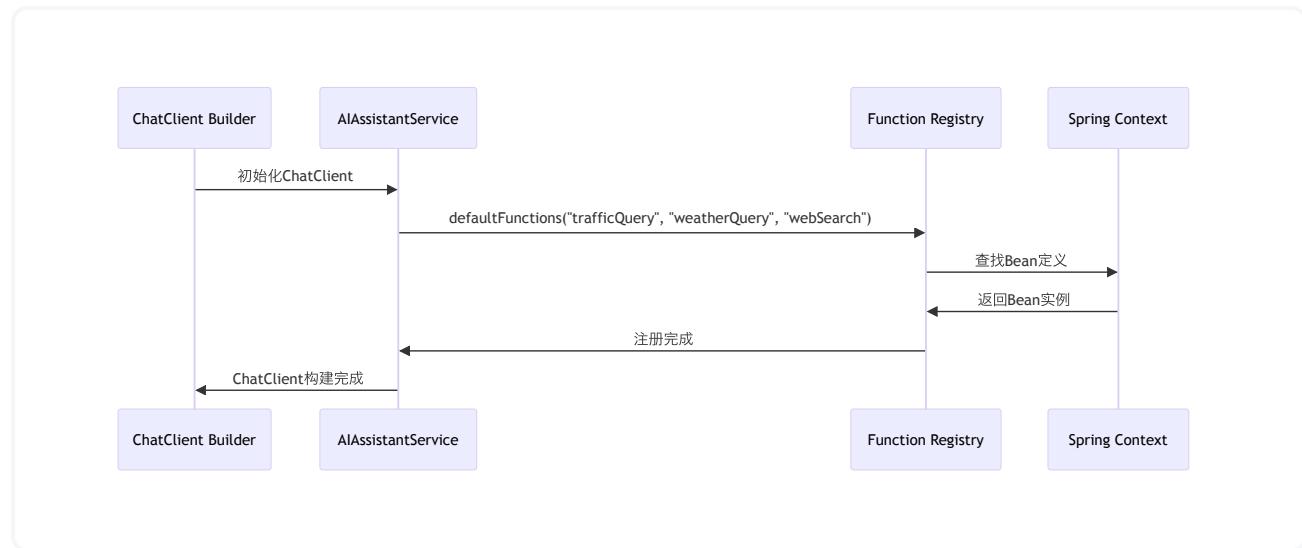
- 整合三轮工具调用的结果
- 生成结构化的风险分析报告
- 包含：风险等级、风险因素、影响区域、应急建议
- 引用数据来源，增强可信度

3.3.1.4 思考过程可视化



3.3.2 Function Calling 机制

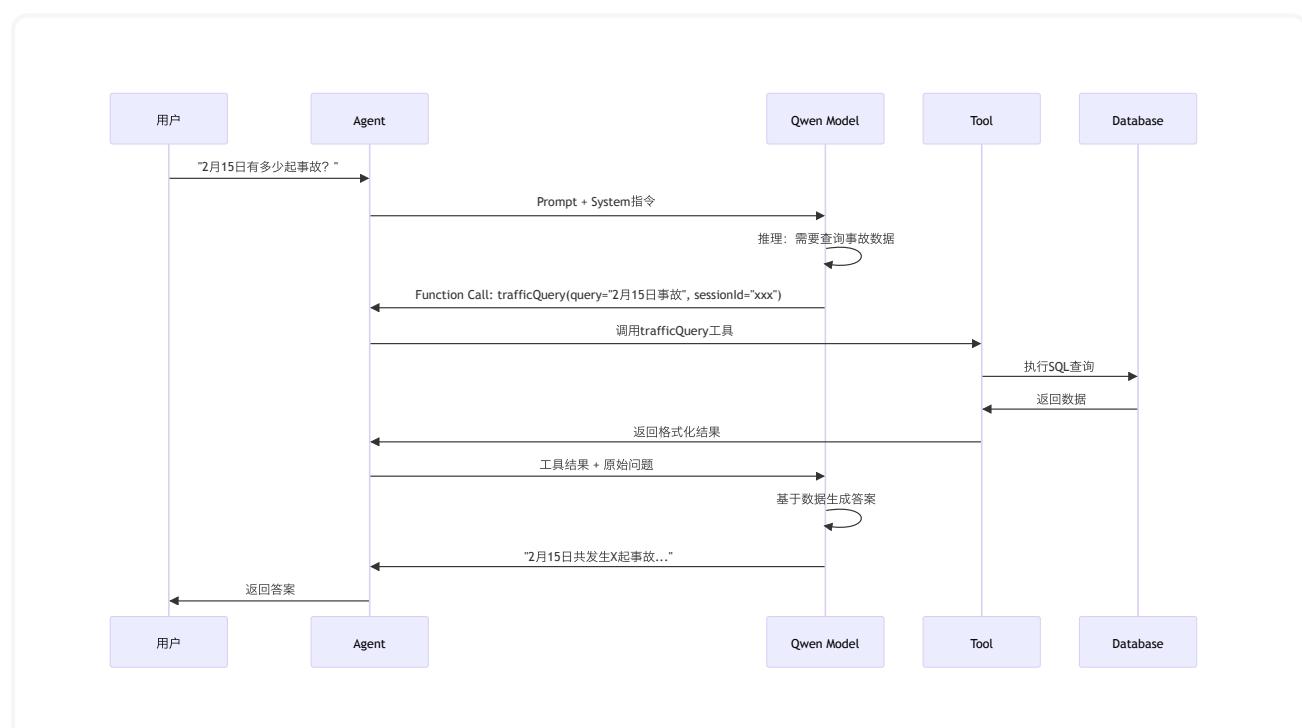
3.3.2.1 工具注册流程



工具定义示例（trafficQuery）：

- Bean 名称： trafficQuery
- 实现类： TrafficDataAnalysisService.analyzeUserQuery()
- 功能： 查询交通数据（事故、客流、事件等）
- 参数：
 - query：用户的自然语言查询
 - sessionId：会话ID（用于记录思考过程）
- 返回：格式化的数据分析结果

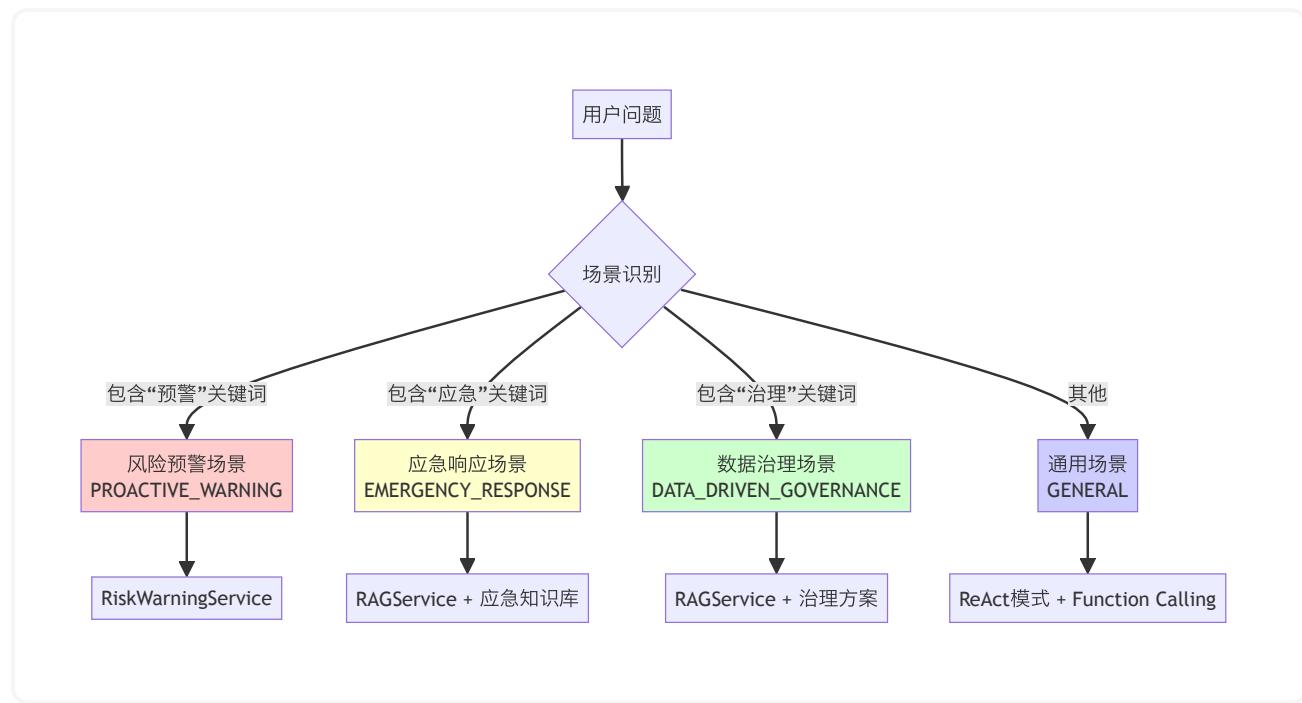
3.3.2.2 工具调用流程



3.3.3 场景路由与规划

系统实现了智能场景识别，根据用户问题自动路由到最合适的处理流程。

3.3.3.1 场景分类

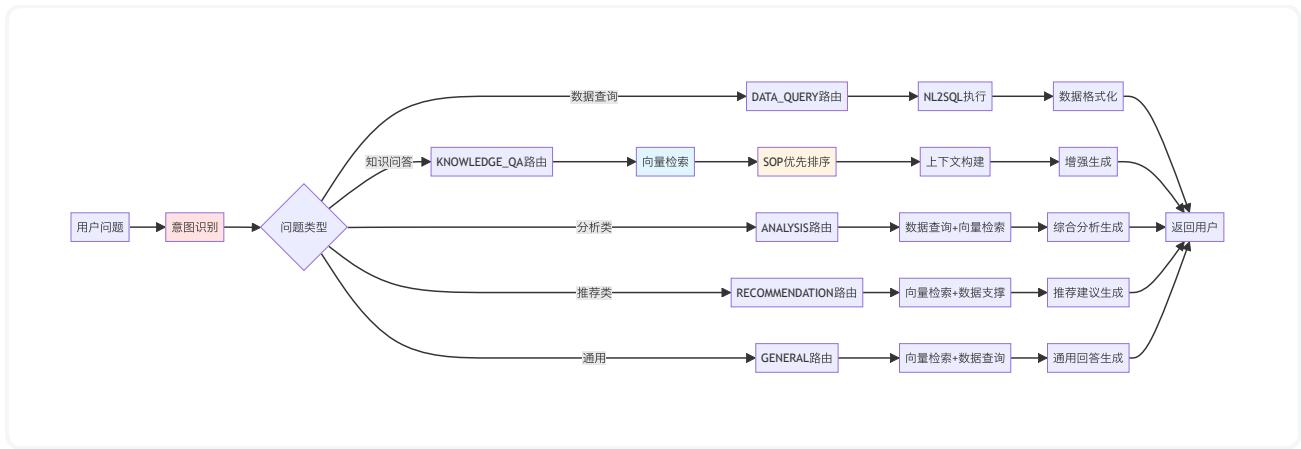


3.3.3.2 场景识别逻辑

- 关键词匹配：**通过预定义关键词集合快速识别场景
 - 风险预警：**预警、风险、隐患、危险
 - 应急响应：**应急、事故、紧急、救援
 - 数据治理：**治理、改善、优化、方案
- 场景特定处理：**
 - 风险预警：**调用 RiskWarningService，综合分析天气、交通、事件数据，生成结构化预警报告
 - 应急响应：**使用 RAG 检索应急 SOP 知识，结合实时数据生成处置建议
 - 数据治理：**分析历史数据，结合专家知识库生成治理方案
 - 通用场景：**进入 ReAct 模式，由 Agent 自主决策

3.4 核心服务架构

3.4.1 RAG 服务架构



3.4.1.1 意图识别

分析用户问题，判断是数据查询、知识问答、分析类、推荐类还是通用问题，使用关键词匹配 + LLM 辅助判断。

3.4.1.2 路径选择

- DATA_QUERY**: 纯数据查询，走 NL2SQL 执行路径
- KNOWLEDGE_QA**: 知识问答，走向量检索 → SOP优先排序 → 上下文构建 → 增强生成路径
- ANALYSIS**: 分析类问题，混合使用数据查询与向量检索，进入综合分析生成
- RECOMMENDATION**: 推荐类问题，结合向量检索与数据支撑，生成推荐建议
- GENERAL**: 通用问题，融合向量检索与数据查询，生成通用回答

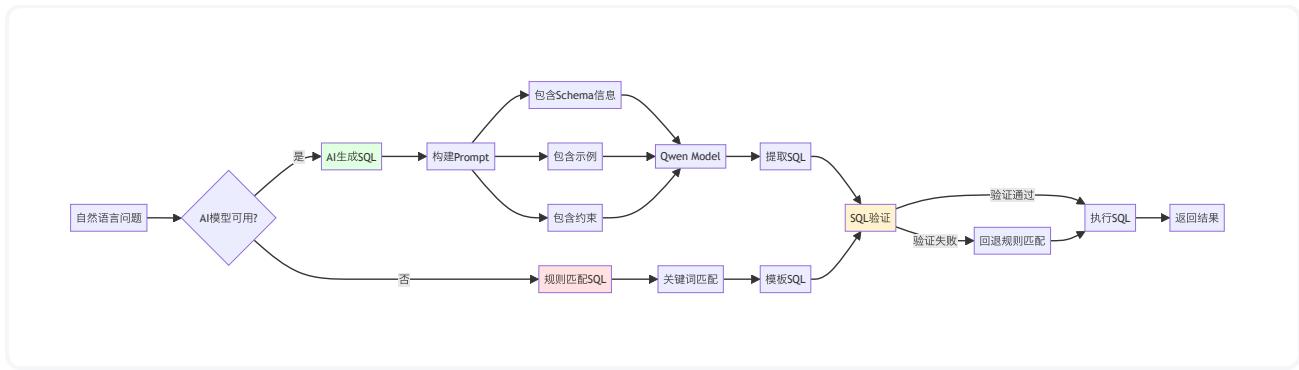
3.4.1.3 上下文构建

- 从 NL2SQL 获取结构化数据（用于 DATA_QUERY、ANALYSIS、GENERAL 路径）
- 从向量数据库（如 Milvus）检索相关知识文档（用于 KNOWLEDGE_QA、ANALYSIS、RECOMMENDATION、GENERAL 路径）
- 在 KNOWLEDGE_QA 路径中，额外经过 SOP 优先排序后再构建上下文

3.4.1.4 答案生成

各路由最终均将处理后的上下文与用户问题送入生成模块，LLM 基于增强后的上下文生成准确、连贯的答案并返回用户。

3.4.2 NL2SQL 服务架构



3.4.2.1 Schema 注入

将完整的数据库表结构信息注入 Prompt，包含表名、字段名、字段类型、数据范围等。

3.4.2.2 双路径保障

- a. **AI 路径：**使用 LLM 生成 SQL，灵活性高
- b. **规则路径：**基于关键词匹配生成 SQL，可靠性高
- c. AI 失败时自动回退到规则路径

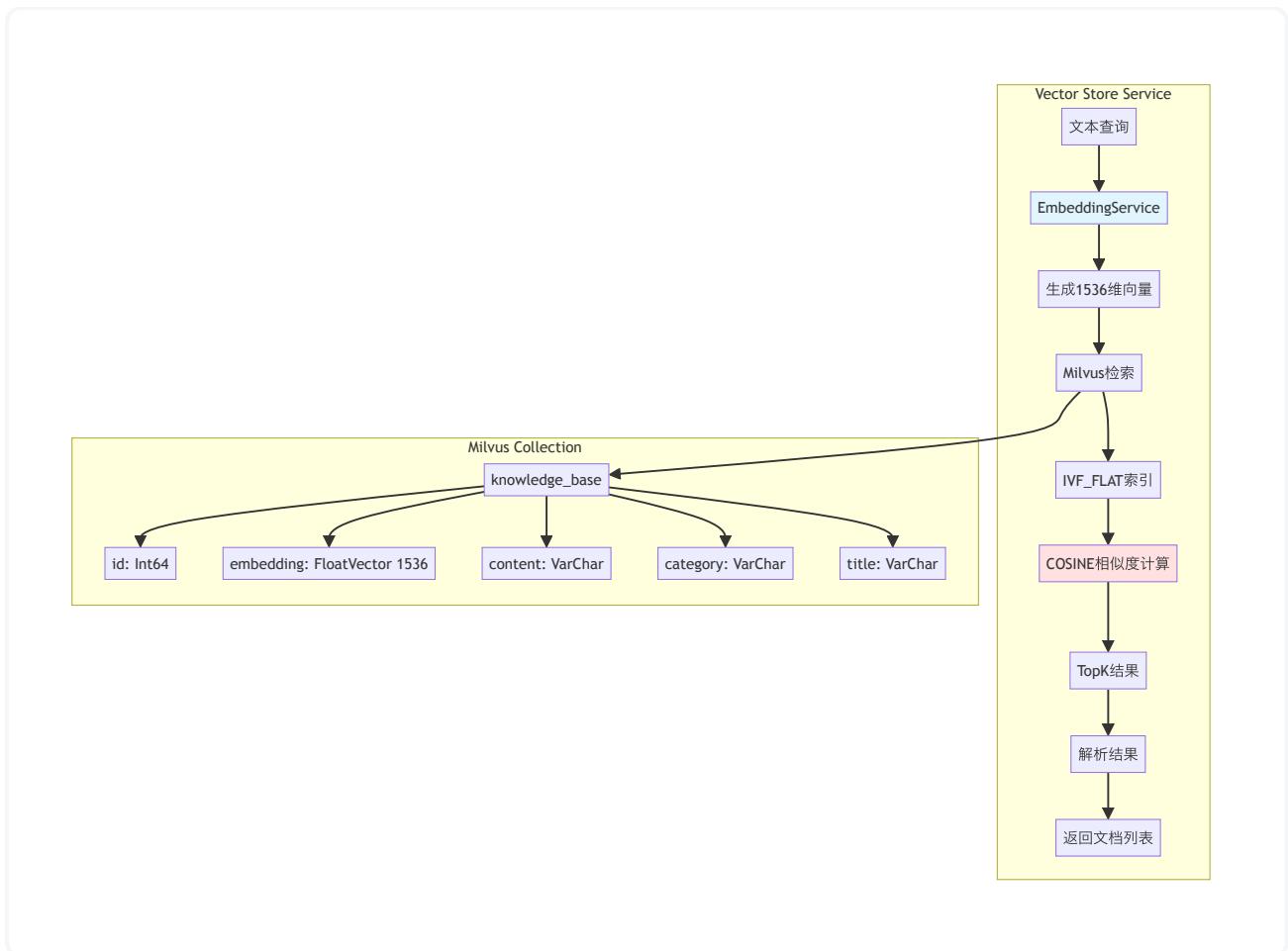
3.4.2.3 SQL 安全验证

- a. 只允许 SELECT 查询
- b. 禁止 DROP、DELETE、UPDATE 等危险操作
- c. 字段名验证，防止查询不存在的字段

3.4.2.4 字段名处理

- a. 支持带空格的原始字段名（如 `CRASH DATE`）
- b. 支持下划线标准化字段名（如 crash_date）
- c. 智能推荐使用原始字段名

3.4.3 向量检索服务架构



3.4.3.1 向量检索特点

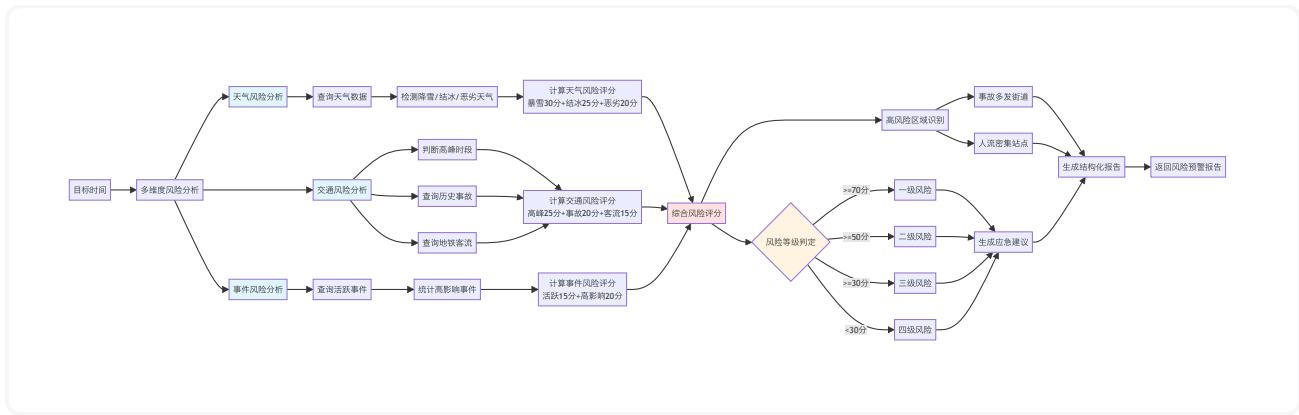
- a. 高维向量：使用 1536 维向量，语义表达能力强
- b. 余弦相似度：使用 COSINE 度量，适合文本语义匹配
- c. IVF_FLAT 索引：平衡检索速度和准确性
- d. 批量操作：支持批量向量化和批量插入

3.4.4 风险预警模块

3.4.4.1 功能定位

多维度风险分析，生成结构化预警报告。

3.4.4.2 技术架构



3.4.4.2 风险评分规则

维度	评分项	分值
天气	降雪	+30
	结冰风险	+25
	恶劣天气	+20
交通	高峰时段	+25
	历史事故>10起	+20
	高密度站点>5个	+15
事件	活跃事件>3个	+15
	高影响事件>0	+20

3.4.4.3 风险等级判定

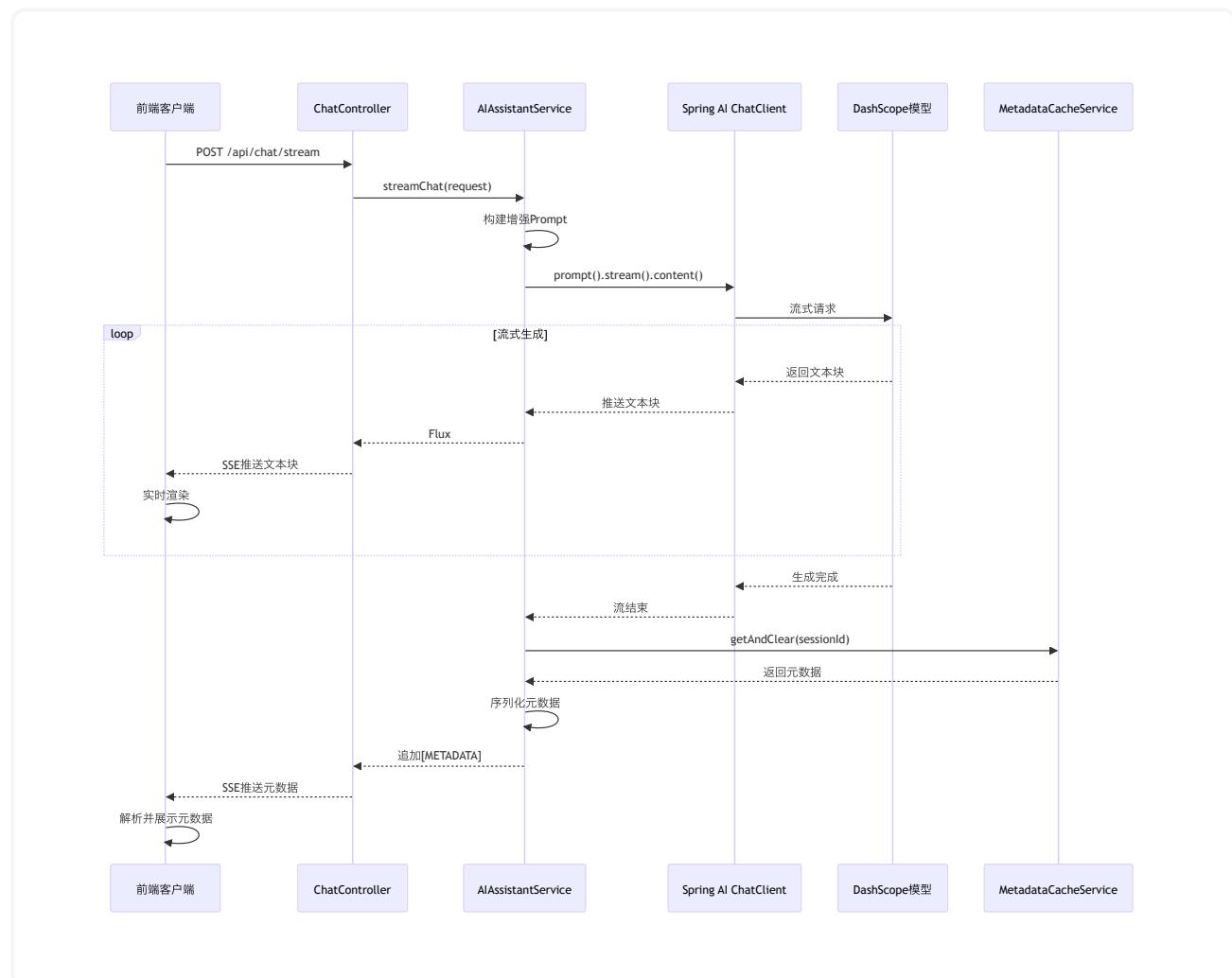
等级	分数	响应措施
一级	≥ 70	立即启动应急预案
二级	≥ 50	启动二级响应
三级	≥ 30	加强监控
四级	< 30	常规监控

3.4.5 流式响应模块

3.4.5.1 功能定位

实现 Server-Sent Events (SSE) 流式对话，提升用户体验。

3.4.5.2 技术架构



4.核心功能演示

本章节通过四个真实业务指令，展示 **Spring AI Alibaba** 架构如何驱动 Agent 完成从数据查询到复杂决策的全流程，一一对应赛题对于“智能经营”、“合规响应”及“根因分析”的要求。

4.1 智能经营分析 (NL2SQL): 打破数据获取壁垒

4.1.1 目标

展示 Agent 将自然语言转化为可执行 SQL 的能力，验证对海量交通数据的即时查询能力。

场景指令： "Traffic Query: 帮我查询2024年2月1号的交通情况"

- 用户输入：“查询2024年2月1号的交通情况”

- 系统思考（中间态）：显示 Agent 正在识别自然语言，并展示生成的 SQL 语句



4.1.2 技术实现

基于 Spring AI Alibaba 的 **Chat Client** 接入通义大模型，利用 **Function Calling** 机制挂载数据库 Schema。Agent 能够理解“2月1号”的时间约束，自动生成零语法错误的 SQL 并执行，将数据提取时间从“小时级”压缩至“秒级”。

4.2 全域风险研判 (DeepResearch): 内外部数据融合

4.2.1 目标

展示 Agent 联网搜索外部信息并结合内部数据进行逻辑推理的能力，验证“主动防御”机制。

场景指令： "Risk Assessment: 请对曼哈顿地区2024年3月的交通状况进行风险评估"

用户输入： “评估曼哈顿2024年3月的交通风险。”

系统思考 (DeepResearch)： 界面展示 Agent 正在调取 Search API，显示关键词气泡：
Manhattan Events March 2024、NYC Weather Forecast March、UN Schedule。

推理过程： 系统识别到“3月17日 圣帕特里克节游行 (St. Patrick's Day Parade) ”与“第五大道”存在时空重叠。

最终输出： 生成一份《3月交通风险预警简报》

4.2.2 技术实现

利用 **DeepResearch**（联网增强）模式，Agent 突破了本地数据库的限制。通过实时抓取互联网公开事件数据，并将其映射到内部路网拓扑中，实现了超越传统模型的“外部干扰感知”。

4.3 合规与应急响应 (RAG): SOP 的动态化执行

4.3.1 目标

展示 Agent 基于私有知识库 (SOP 手册) 生成合规处置方案的能力，验证“标准行动生成”。

场景指令: "Emergency Response: 请查找2024年2月1日的交通事故，并提供相对应的应急救援方案"

- **用户输入:** “查找2月1日事故并给出救援方案。”
- **最终输出:** 将静态文档转化为动态行动清单



4.3.2 技术实现

基于 **Vector Store** (向量数据库) 存储企业私有的应急预案。Agent 通过语义检索精准匹配事故类型对应的条款，通过大模型的理解能力，将“死”的文档转化为现场指挥员可直接勾选执行的“活”清单。

4.4 隐患深度复盘 (Code Interpreter): 数据驱动的黑点挖掘

4.4.1 目标

展示 Agent 编写并执行代码进行复杂数据分析的能力，验证“事故黑点识别”。

场景指令： "Safety Review: 帮我复盘一下2024年2月的交通安全整体情况，识别出事故黑点区域"

用户输入：“复盘2月安全情况，识别事故黑点。”

最终输出：

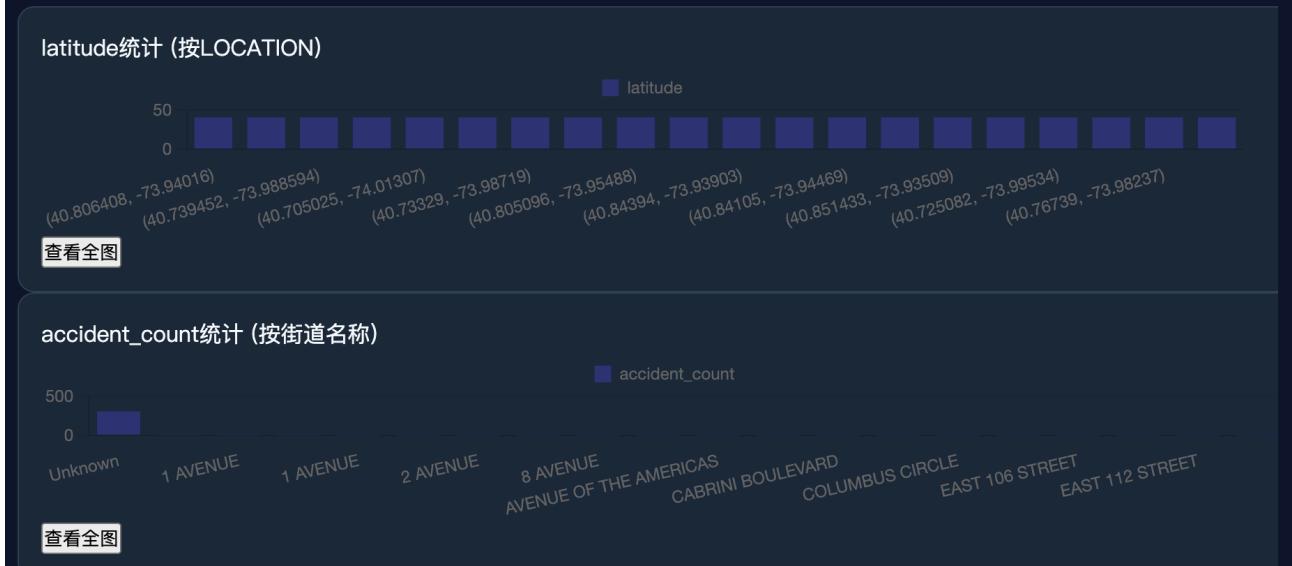
- 可视化图表：一张带有主装分类的区域事故数量图，直观清晰展示各区域响应事故。
- 洞察结论：“2024年2月曼哈顿共记录交通事故6,588起，整体事故频发。· 事故数据中307起未标注具体街道（Unknown），但可识别的高发路口为EAST 126 STREET与MADISON AVENUE交叉口。· 该路口位于上东区核心商业带，南北向主干道MADISON AVENUE车流密集，且周边商业活动频繁，行人过街需求高。”

【分析】· 2024年2月曼哈顿共记录交通事故6,588起，整体事故频发。· 事故数据中307起未标注具体街道（Unknown），但可识别126 STREET与MADISON AVENUE交叉口。· 该路口位于上东区核心商业带，南北向主干道MADISON AVENUE车流密集，且周边商业活动频繁，行人过街需求高。· 主要致因为“驾驶分心/注意力不集中”（占比206%）和“未按规定让行”（占比56%），表明驾驶员行为是关键风险好，无降雪或强降水，排除恶劣天气对事故率的直接影响。【结论】2024年2月曼哈顿交通安全最大黑点为EAST 126 STREET与MADISON AVENUE交叉口。该区域因道路功能复合（通勤+商业）、人车流量大及驾驶员违规行为集中，导致事故发生频率显著高于其他路段。事故主要，非环境所致。【建议】· 在EAST 126 STREET/MADISON AVENUE部署智能交通监控系统，自动识别“闯红灯”、“未让行”等违法行为。· 优化信号配时方案，增设行人专用相位（Pedestrian Scramble Phase），减少人车冲突。· 增设中央隔离带与行人安全岛，降低碰撞风险。· NYPD 开展专项执法行动，针对“驾驶分心”行为进行劝导与处罚。· 推动社区宣传计划，在周边商户张贴交通安全提示，提升公众安全意识。

● 处理时间: N/Ams  数据查询

■ 查询表: nyc_traffic_accidents, nyc_traffic_accidents, nyc_traffic_accidents, weather_api_manhattan_2024_02

✓ 已接入天气数据并生成图表



4.4.2 技术实现

集成 **Code Interpreter**（代码解释器）。针对 SQL 难以处理的空间聚类与复杂统计任务，Agent 能够现场编写 Python 脚本进行深度挖掘，从单纯的“看数据”进化为“诊断业务”，实现企业级的交通智Agent级别的综合性应用。

4.5 流式响应与 React 前端交互

为了解决 AI 生成长文本（如深度研判报告）或执行复杂任务（如 DeepResearch 联网搜索）时的等待焦虑问题，本项目采用了 Spring AI Stream 流式接口与 React 增量渲染技术，将“全量等待”转化为“即时反馈”。

4.5.1 后端机制：Spring AI Stream 流式输出

传统模式下，用户需要等待 10-20 秒才能一次性看到结果。我们利用 Spring AI Alibaba 的 ChatClient.stream 接口，基于 Server-Sent Events (SSE) 协议实现了打字机效果：

- a. **Token 级推送**：后端不等待整个回答生成完毕，而是每生成一个 Token（字/词）就立即推送到前端。
- b. **首字延迟 (TTFT) 优化**：将首字响应时间压缩至 500ms 以内，用户发出指令后立即看到系统开始“思考”或“撰写”。
- c. **多阶段状态流**：在处理复杂任务（如场景四的 DeepResearch）时，我们不仅推送最终文本，还将中间步骤（Thought Chain）作为特殊事件流推送。
- d. **数据流示例**：[Searching: 'Manhattan Traffic Events'] -> [Analyzing Data] -> [Generating Report]。

4.5.2 前端架构：React 增量渲染与组件化

前端基于 React 框架，设计了一套能够解析混合数据流的渲染引擎，不再是简单的文本展示，而是实现了“根据内容自动挂载组件”：

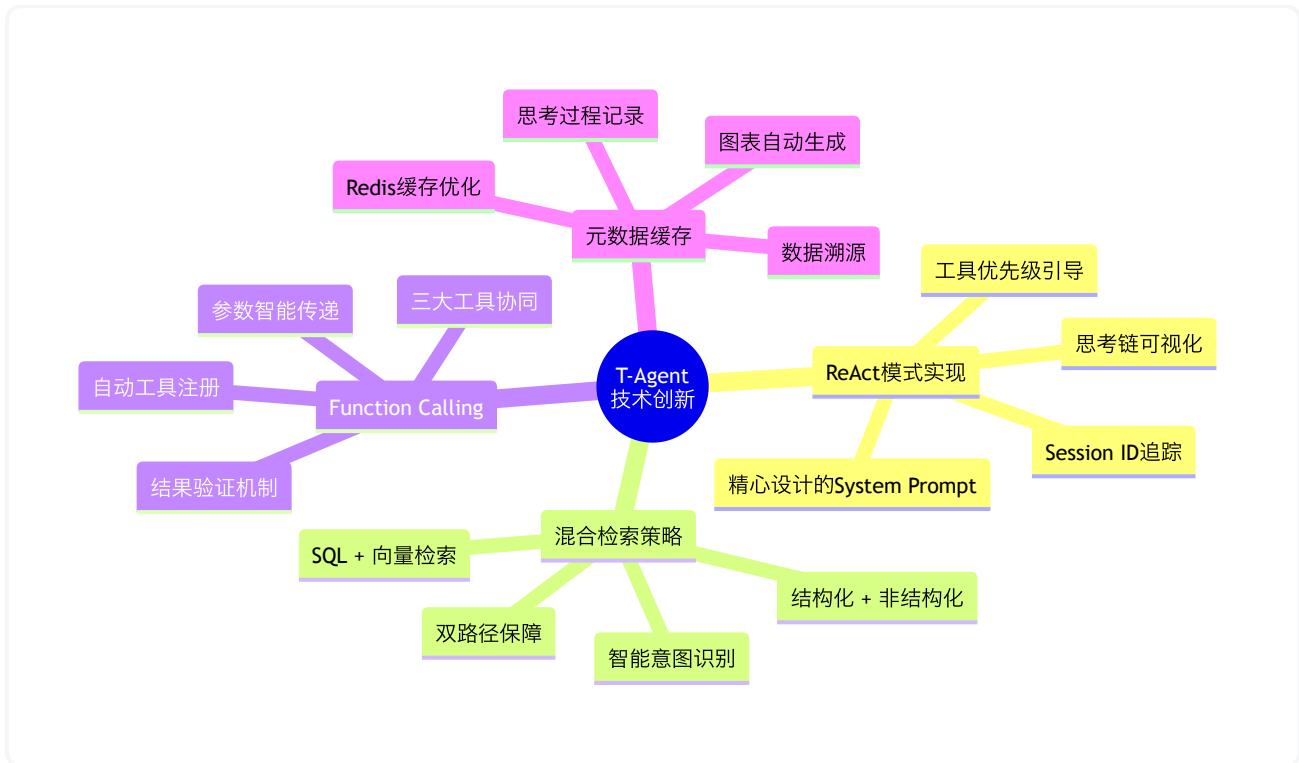
- a. **自定义 useChat Hook**：封装了 SSE 连接逻辑，维护一个动态更新的 message 状态。当后端推送新的 chunk 时，自动追加到当前对话框中，触发 React 的局部重渲染。
- b. **结构化数据与组件映射 (Component Mapping)**：这是本项目前端最大的亮点。Agent 返回的不仅仅是 Markdown 文本，还包含特定的结构化数据标记。React 前端通过正则匹配或 JSON 解析，动态加载对应的业务组件。
- c. **文本流**：使用 react-markdown 组件，配合自定义的渲染器，实时展示加粗、列表等格式，实现平滑的“打字机”视觉效果。

通过这种 Spring AI Stream + React Dynamic Components 的组合，我们将原本枯燥的文本对话，升级为了一场可视化的、动态的智能交互体验

5. 创新点与难点

5.1 创新点

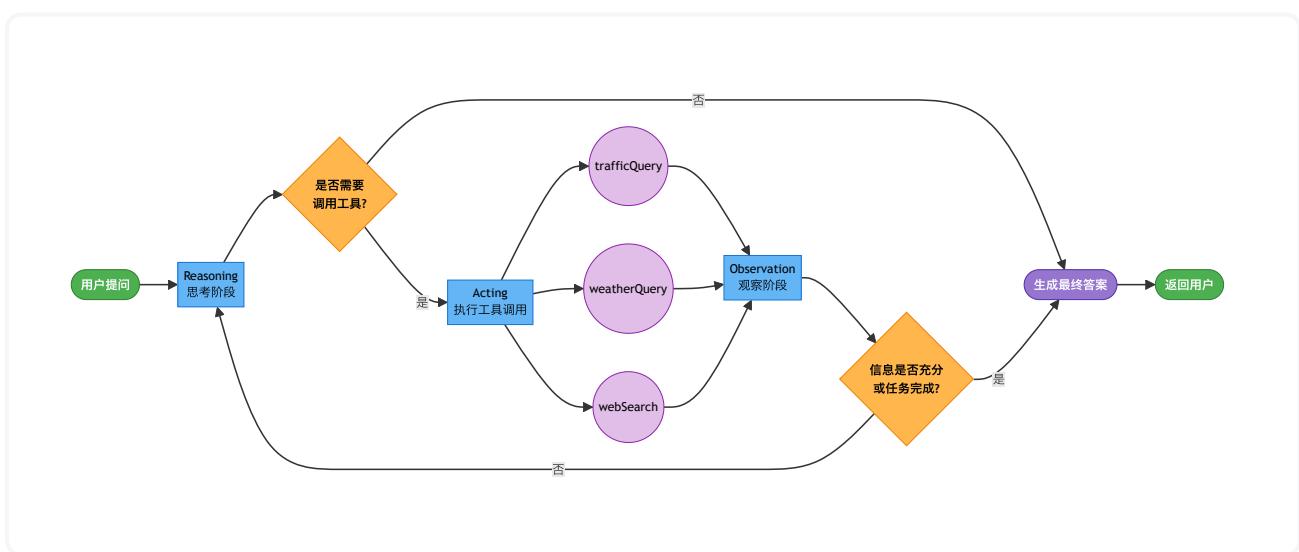
本系统围绕智慧交通场景的核心需求，进行了多项关键性技术创新，使智能体能够面向真实业务环境完成复杂任务。以下从三个最具代表性的创新点展开说明。



5.1.1 ReAct 推理链的工程化落地

在智慧交通场景中，用户的问题往往并不只是一次性检索就能回答。例如“分析某天的交通风险并给出建议”这样的任务，需要先判断是否需要天气数据，再查询事故情况，再结合历史趋势或相关事件进行综合性推理。而传统问答系统通常只能直接生成一句答案，无法进行多步思考，也无法在合理的顺序中自动调用工具获取信息，更无法向用户展示推理的逻辑过程。这种不透明、不可靠的推理方式，不适用于交通治理等高可信度需求的场景。

智慧交通中的很多业务查询本质上是“多步骤推理任务”，而非单轮问答式任务；它需要系统具备跨数据源整合能力、步骤规划能力与可靠的工具调用能力。



为解决这一痛点，本系统引入了 ReAct (Reasoning + Acting) 推理模式，并通过精心设计的 System Prompt 与 Function Calling 机制，将其工程化地融入整个智能体执行流程。当 Agent 接收到用户请求后，会首先进入 Reasoning (思考) 阶段，分析问题并判断是否需要调用外部工具；若需要，则进入 Acting (行动) 阶段 自主选择合适的工具（如 trafficQu

`ery`、`weatherQuery`、`webSearch`），并以规范化参数形式执行调用。工具返回结果后，Agent 会进入 Observation（观察）阶段，评估所获得的信息是否足以继续推理或产出最终结论。整个过程以“思考—行动—观察”的闭环形式不断迭代，直到获得可以支撑答案的完整信息链。

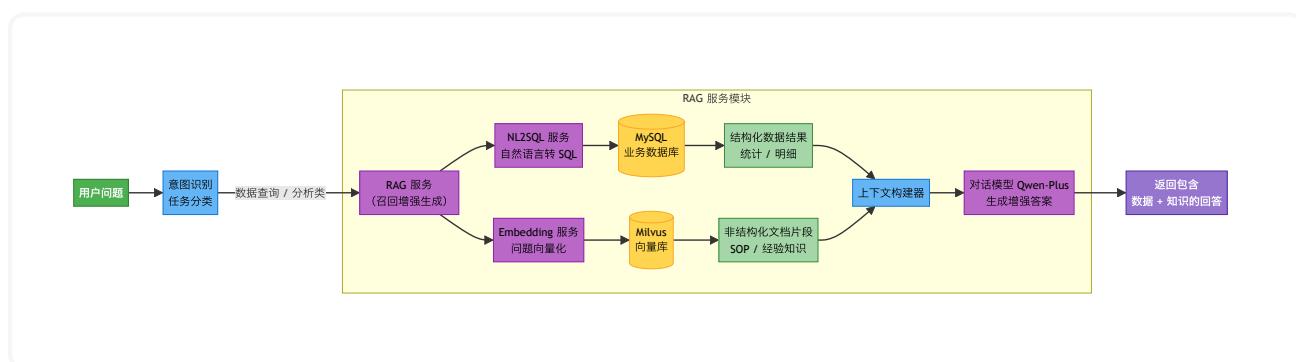
在工程实践中，系统不仅提供了统一的工具调用接口，还结合场景识别策略在合适场景中自动触发 ReAct 模式。同时通过 Prompt 规则约束工具优先级与参数格式，保证模型行为可控；通过会话标识记录每轮推理过程，使推理链路可追踪、可审计、可调试。

ReAct 推理链的引入，使 T-Agent 从传统的“被动式问答”升级为“主动式智能规划者”。在交通风险分析等复杂应用场景中，Agent 能够主动分解任务、多轮调用工具、整合多源信息，并最终生成包含数据依据与原因分析的高质量回答。相比过去的一次性生成结果，这一机制大幅提升了系统的智能性、稳定性与可信赖性，使用户获得的答案更准确、更透明，也更适用于支持分析决策类业务需求。

5.1.2 SQL + 向量混合检索 RAG 架构

在智慧交通场景中，许多业务问题都具有多维度、跨数据源的特征。例如“分析某日事故原因”“预测节假日交通风险”之类任务，既需要从业务数据库中获取精确的结构化数据，又需要结合应急预案、治理经验、历史案例等非结构化知识。因此，仅依赖传统 SQL 查询或单纯向量检索的方式，都无法提供完整且具备解释性的答案。

这一行业共性难题的核心在于：结构化信息与非结构化知识长期孤立，无法在单一回答中被统一利用。



为解决这一行业共性难题，本系统设计了一套 **SQL 与向量检索并行的混合式 RAG 架构**。系统通过 NL2SQL 将用户自然语言转换为 SQL，使模型能够直接访问业务数据库并生成结构化结果；同时，问题也会被向量化并用于检索知识库，以获取适用于解释、研判或建议的文本知识。两类信息随后由上下文构建模块进行融合，使回答同时具备数据精确性与知识背景完整性。例如，在分析事故原因时，系统既能提供“当日共有 45 起事故”这样的硬事实，也能提取相应 SOP 或专家经验作为分析依据。

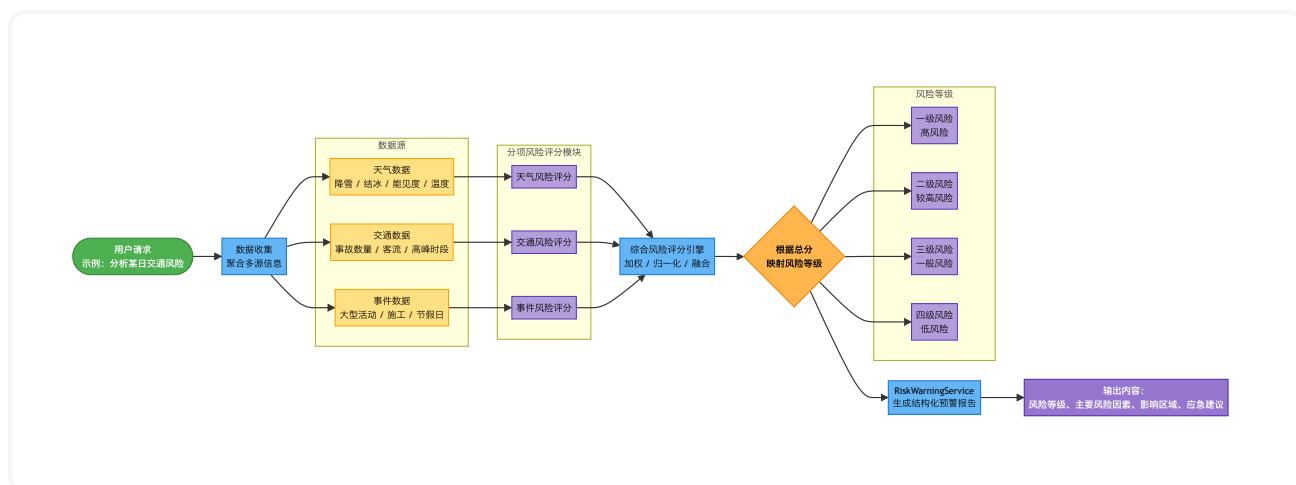
混合检索RAG架构极大提高了答案的完整性和准确性。模型既不会漏掉数据库中的关键数据，也能获取语料库中的相关知识，实现“双保险”。例如在交通事故分析场景，系统既能通过 SQL 获取某日期事故发生数，又能从知识库中检索相关安全指南，从而给出兼具准确数据和背景解释的答案。这种架构创新使智能体能够处理更广泛的问题类型，在智慧交通领域

提供可靠的决策支持。对业务而言，混合检索确保了信息覆盖全面且细节可靠，提升了系统回答复杂问询（如趋势分析、原因解析）的能力，有助于建立用户对系统的信赖。

5.1.3 多维交通风险评分模型及结构化报告输出

城市交通风险评估涉及多因素影响，例如天气状况、历史事故数据、特殊事件等。传统上需要人工或单一指标衡量风险，难以及时全面地给出预警。本系统需要在智慧交通场景下，针对特定日期或场景自动评估风险等级并生成结构化报告，以便相关部门快速获取决策依据。挑战在于如何融合多源异构数据形成统一的风险评估，以及如何让生成的报告清晰展示关键风险因素。

在实际城市治理中，“缺乏可量化、可复用、可自动化的风险评估模型”长期是行业痛点，导致风险研判依赖人工，灵敏度不足、响应滞后、且难以形成统一标准。



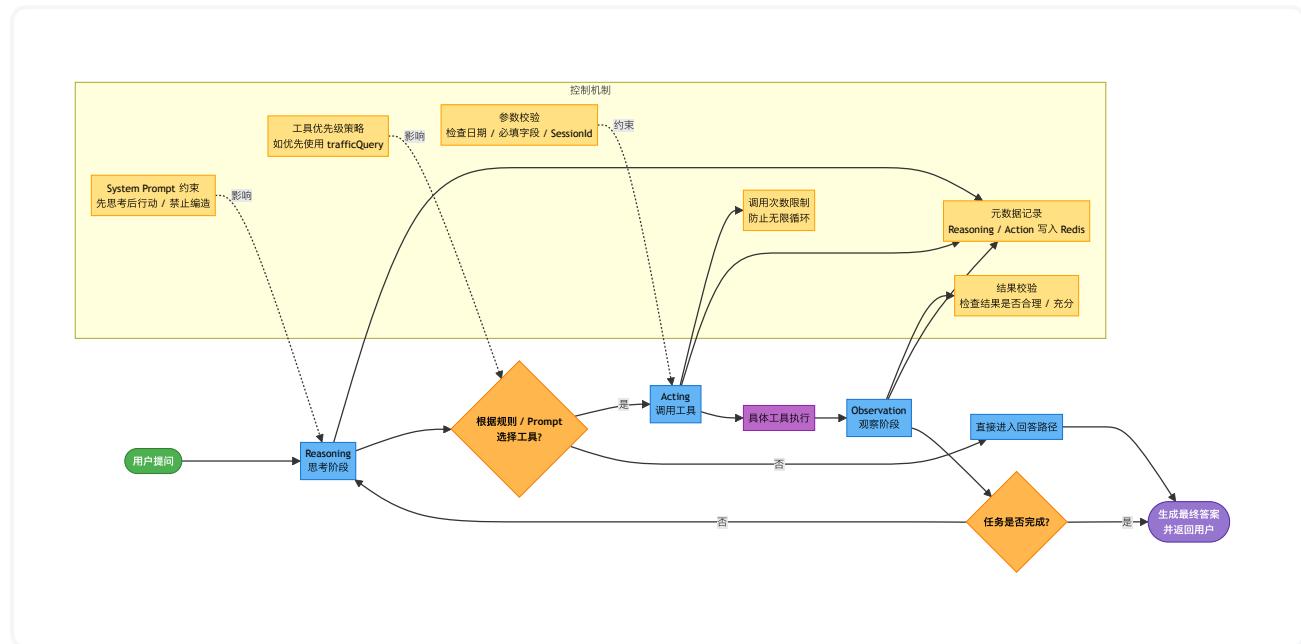
为此，本系统研发了多维交通风险评分模型，并通过 Agent 调用专门的 RiskWarningService 来生成结构化的风险分析报告。风险评分模型综合考虑了天气因素（如降雪、温度对交通的影响）、交通历史数据（如事故频率、同比变化）以及事件信息（如节假日活动等）。当用户提出诸如“给出某日期交通风险分析”的请求时，Agent 将在 ReAct 推理链中先后获取各维度数据：调用 weatherQuery 获取天气状况，调用 trafficQuery 获取当日事故统计，最终调用 RiskWarningService 对收集的数据进行分析评估。RiskWarningService 内部依据预设的风险评估模型计算风险等级（如一级风险、二级风险），并按照模板生成包含风险等级、主要风险因素、影响范围、以及建议措施等字段的结构化报告。报告生成后，Agent 将该报告嵌入答案或直接返回给用户。

通过多维风险评分模型的引入，系统能够自动化地完成交通风险评估，实现对复杂场景的及时预警。生成的结构化报告使输出信息一目了然：明确的风险等级和因素说明便于用户快速理解重点，并可作为决策依据。例如，在“大雪天气下某日交通是否高风险”的提问中，系统最终返回的报告会指出该日因暴雪导致事故激增，风险等级为一级，并附上建议的应对措施。这种创新为智慧交通管理提供了实用价值：相关部门可更迅速地获取专业的风险研判结果，提前采取措施；同时普通用户也能获得权威且易读的出行风险提示，体现了本系统在业务场景中的价值提升。

5.2 技术难点

在创新体系构建的同时，系统也需要解决多项工程与算法挑战，确保整个智能体具有足够的稳定性、安全性和一致性。以下挑选三个最具代表性的难点进行说明。

5.2.1 ReAct 推理链稳定性与工具调用控制



在引入 ReAct 多步推理后，模型需要自主决定何时、如何调用工具，这带来了新的不确定性。如果缺乏约束，Agent 可能出现不当行为。

例如：选择错误的工具、忽略必须调用的数据源，甚至陷入无休止的思考-行动循环。此外，多轮工具调用也增加了系统复杂度，如何确保每轮输出可靠且不偏离正轨是一个挑战。稳定控制不佳可能导致错误累积，影响最终答案的正确性和一致性。

针对上述问题，本系统在Prompt设计和系统机制上采取了多重措施以稳定 ReAct 推理过程。

a. 明确的工具优先级与调用规则：

在 System Prompt 中预设各类工具的优先级，通过强调语句要求模型在特定场景中“必须优先使用指定工具”，例如在涉及交通数据分析时优先调用 trafficQuery，有效降低工具选择误差。

b. 严格的参数传递规范：

系统要求所有工具调用必须携带会话 ID 与必要参数，保证推理链条的连续性，使每一步 Reasoning 与 Action 都能够被追踪、审计与调试。

c. 思考链引导提示：

为避免模型跳步或省略关键推理步骤，Prompt 明确要求模型遵循 **Reasoning → Acting** 的顺序执行，通过“先思考、再行动”的提示确保推理链条结构化、可解释。

d. 工具返回结果的验证机制：

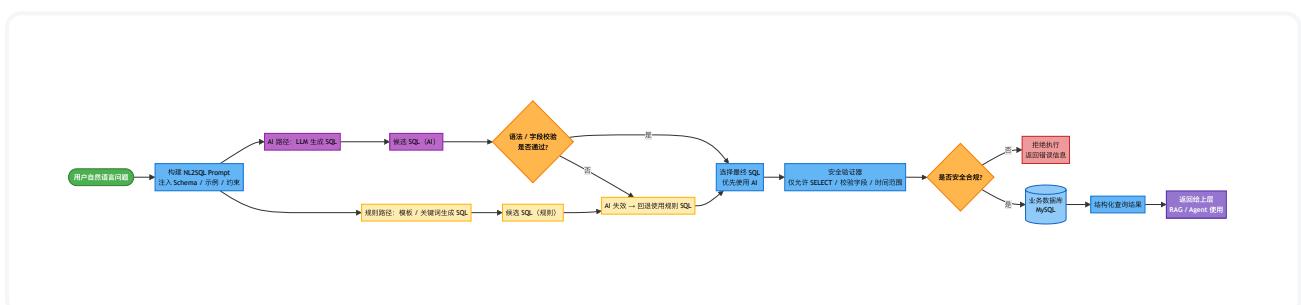
在 Observation 阶段，模型必须检查工具返回的数据是否合理，例如与预期是否相符、信息是否完整。若不足以支持下一步推理，则需主动发起补充工具调用，以减少错误累积。

e. 循环调用的监控与限制：

系统层面设置了最大工具调用次数等安全阈值，防止模型陷入无限循环。此外，每轮 Reasoning / Action 都会被记录在会话元数据中，为调试推理链和可视化展示提供支持。

通过上述策略，系统有效地提升了 ReAct 推理链的稳定性和可控性。实际运行中，Agent 基本能按预期顺序调用工具，未出现明显偏离脚本的行为。工具调用成功率和有效性提高，模型不会因为错误理解而调用无关工具。尤其是在复杂任务下，多轮推理过程始终可追踪、可监控，这使开发者能及时发现并修正异常情况，进一步增强系统可靠性。整体来看，这一难点的解决保证了创新的 ReAct 机制切实发挥作用，为用户提供稳定可靠的多步推理服务。

5.2.2 NL2SQL 的 SQL 生成准确性与安全性



将自然语言转换为SQL查询（NL2SQL）是系统获取结构化数据的关键环节。然而这项技术存在两方面难题：准确性和安全性。准确性方面，模型可能生成语义或语法不正确的SQL（如字段名错误、联表关系不当），导致查询结果偏差甚至报错。安全性方面，如果不加限制，恶意或不当输入可能诱使生成危险SQL语句（如试图删除数据），或执行耗时的全表扫描，威胁数据安全与系统性能。因此，需要在允许灵活查询的同时，严格防范错误和风险。

本系统围绕 NL2SQL 模块设计了多层保障机制，以提高生成SQL的准确性并确保安全合规。

a. 完整 Schema 注入以增强语义理解

在 Prompt 中嵌入数据库的表结构信息，包括表名、字段名、字段类型及字段含义范围，使模型在生成 SQL 时具备充分的结构化知识，有助于减少字段拼写错误、表名误用等常见问题。

b. “双路径” SQL 生成机制提升稳定性

- i. **AI 路径（LLM 生成 SQL）**：具备较强灵活性，适用于开放式、复杂查询。

ii. 规则路径（模板或关键词匹配生成 SQL）：稳定可靠，适用于标准化查询场景。

在实际执行时，系统会优先尝试 AI 路径；若检测到生成结果语法错误、字段不匹配或置信度不足，则自动回退到规则路径，确保无论模型表现如何，都能产出有效的 SQL。

c. 严格的 SQL 安全验证机制

在执行 SQL 之前，系统会对生成的 SQL 进行强制审核，以杜绝潜在风险：

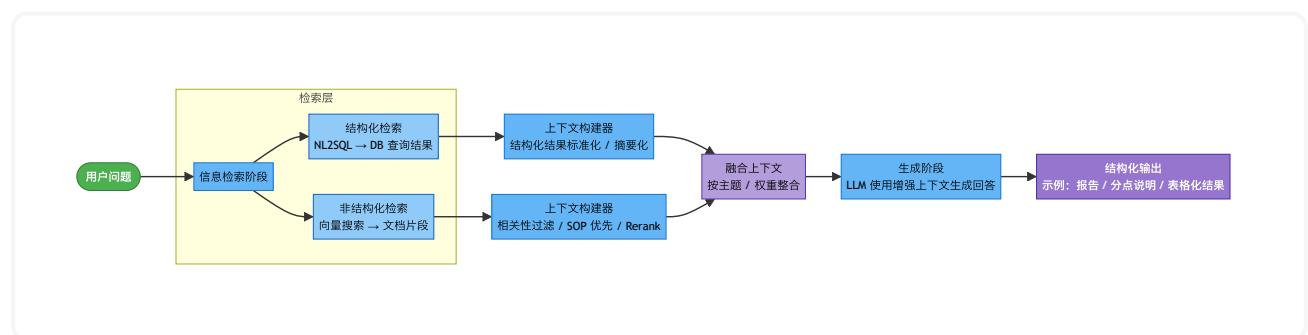
- i. 仅允许 **SELECT** 查询，禁止 **DROP / DELETE / UPDATE** 等修改操作。
- ii. 校验字段和表名是否真实存在，避免无效查询。
- iii. 自动加入必要的查询条件限制，如时间范围、索引字段，避免因无条件扫描导致数据库性能下降。

d. 字段命名差异的标准化与纠正机制

模型生成的字段名可能与数据库实际字段存在格式差异，例如将 `CRASH DATE` 错写为 `crash_date`。系统在解析阶段会进行字段标准化与映射修正，使生成的 SQL 能够正确匹配数据库 Schema，从而避免查询失败。

经过上述改进，NL2SQL 模块在准确性和安全性方面表现显著提升。一方面，SQL 生成的成功率和正确率大大提高——模型几乎不再产生语法错误查询，对复杂查询的语义理解也更加到位；即使偶有 AI 路径失败，规则路径的补充也保证了结果的可用性。另一方面，安全机制有效防止了危险操作和不合理查询的执行，实现了零破坏性操作事故。整体运行监测显示，通过 Schema 提示和字段映射，生成 SQL 能准确匹配数据库结构；通过安全验证和优化，查询性能良好且数据安全无虞。这些措施共同确保了数据库检索这一关键环节的可靠稳健，为混合检索架构的顺利运行奠定了基础。

5.2.3 RAG 上下文融合与输出连贯性



在混合检索获取多源信息后，如何将结构化数据和非结构化文本有效融合供生成模型使用，是一大技术难点。简单地拼接所有检索结果可能导致上下文冗余或矛盾，模型生成的回答可能缺乏连贯性：例如数据表结果和文本描述风格差异巨大，直接混合会使答案读起来不流畅；多篇文档内容若无筛选，可能信息冲突或干扰重点。此外，大模型对提供的上下文长度有限，需要精选关键信息。在保证答案精准的同时，还必须让输出表述连贯清晰、前后一致。

为解决上述上下文杂乱、信息冲突及回答不连贯等问题，本系统在 RAG 模块中对上下文构建和结果生成环节进行了系统化优化，主要体现在以下几个方面：

a. 基于问题类型的上下文整合策略设计

系统首先根据问题类型（如事实类、分析类、知识类）确定上下文构建方式：

对于分析性或综合性任务，系统会将 SQL 返回的结构化数据进行格式化，例如提取事故统计、趋势指标等关键数值，再与向量检索得到的相关文本内容按语义主题进行合并，形成逻辑连贯的上下文块。

对于纯知识问答场景，系统引入 **SOP 文档优先级排序机制**，按照语义匹配度与文档权威度筛选内容，确保模型优先看到最重要、最能回答问题的知识片段。

b. 生成阶段的连贯性引导

为避免模型生成内容时出现跳跃、重复或逻辑断裂，系统在 Prompt 中明确要求模型需要将多源信息“综合成统一的回答”。必要时，会引导模型采用段落式、步骤式或报告式的结构，以增强内容的条理性和可读性。

c. 特定场景下的格式化模板支持

对于业务要求严格的场景（如交通风险预警报告），系统提供了标准化的输出模板或示例，帮助模型按照指定格式（如标题、小节、项目符号等）生成结构化内容，使输出结果更加规范、专业、易读。

d. 生成结果的质量监测与迭代优

系统会对生成过程进行日志记录，当发现模型输出存在不连贯、信息遗漏或引用不充分等问题时，可通过调整提示词、优化上下文选择逻辑等方式进行迭代改进，持续提升 RAG 效果。

通过改进上下文融合方式和生成引导，系统的回答在保证准确详实的同时，显著提升了连贯性和易读性。整合后的上下文既涵盖关键数据又消除了噪音冗余，使生成模型能够产出前后一致、条理清楚的答案。例如，在综合分析某事故成因时，最终回答会先引用数据库精确数据佐证，然后结合检索到的知识进行解释，行文流畅自然，信息衔接紧密。针对风险报告等场景，输出严格按照预期结构呈现，段落清晰、要点分明，达到了专业报告的水准。

6. 快速开始

本章节将指导您如何快速启动和运行T-Agent。

6.1 系统要求

- a. **Docker Desktop** :4.0+ (包含 Docker Engine 和 Docker Compose)
- b. **Python** 3.8+ (用于知识库初始化)
- c. 内存 至少 8GB RAM (推荐 16GB)
- d. 磁盘空间 至少 10GB 可用空间

6.2 快速启动

6.2.1 开源代码地址

<https://github.com/Deemoowo/smartTransprotation>

6.2.2 克隆项目（如果尚未完成）



复制代码

```
git clone <项目地址>
cd smartTransportation
```

6.2.3 一键启动系统

Windows 用户：双击 `start.bat` 文件，或在命令行中运行：



复制代码

```
start.bat
```

Mac/Linux 用户：在终端中运行：



复制代码

```
chmod +x start.sh
./start.sh
```

6.2.4 启动过程说明

启动脚本将自动执行以下步骤：

- a. 环境检查 - 验证 Docker 是否已安装并运行
- b. 镜像构建 - 构建应用 Docker 镜像
- c. 服务启动 - 启动所有必需的服务：

MySQL 数据库

Redis 缓存

Milvus 向量数据库

应用服务

- d. 等待就绪 - 等待所有服务完全启动（约30秒）
- e. 知识库初始化 - 导入 SOP 和专家知识到向量数据库

6.3 访问系统

启动完成后，可通过以下地址访问系统：<http://localhost:8080>

6.4 系统配置

环境变量配置:系统通过环境变量进行配置，在 `docker-compose.yml` 中定义：

a. **数据库:**

用户名: `{your_username}`

密码: `{your_password}`

数据库: `smart_transportation`

b. **Redis:** 默认配置

c. **Milvus:** 默认配置

修改端口:如需修改默认端口，请编辑 `docker-compose.yml` 文件中的 `ports` 部分。