财顾报告 Agent：技术架构与选型说明

> 本文基于《财顾报告智能化生成产品 MVP 版本需求文档》的理解，目标：**可落地**、**业务体验友好**。

0. 术语澄清

表述	本文含义
后端用 Java	指**业务主后端**（统一接入、任务与状态、租户与权限、持久化、行内集成、导出编排等）采用 **Java（Spring Boot）**；**不**表示「大模型、RAG、Agent 编排也必须在 Java 里重做一份」。
算法 / Agent 端	独立的 **Python 服务**，承担报告生成链路中的 **多段 LLM 调用、结构化大纲解析、RAG、提示词渲染与试验协议（如 LangGraph）** 等，与 Java **进程分离、接口契约化**。
前端 Web	**浏览器 SPA**（如 React），只对接 **Java 暴露的 API / SSE**，**不直接调用 Python**（便于安全、网关、审计统一落在 Java）。

因此整体是 **Web + Java 业务后端 + Python 算法（Agent）服务** 的三端协同；您说的「Web 和 Java 之间还要有一个算法端」更准确的调用关系是：

Web → Java（门户与编排所有者）→ Python（Agent 执行 LLM/RAG）→ Java 落库 / 继续状态机

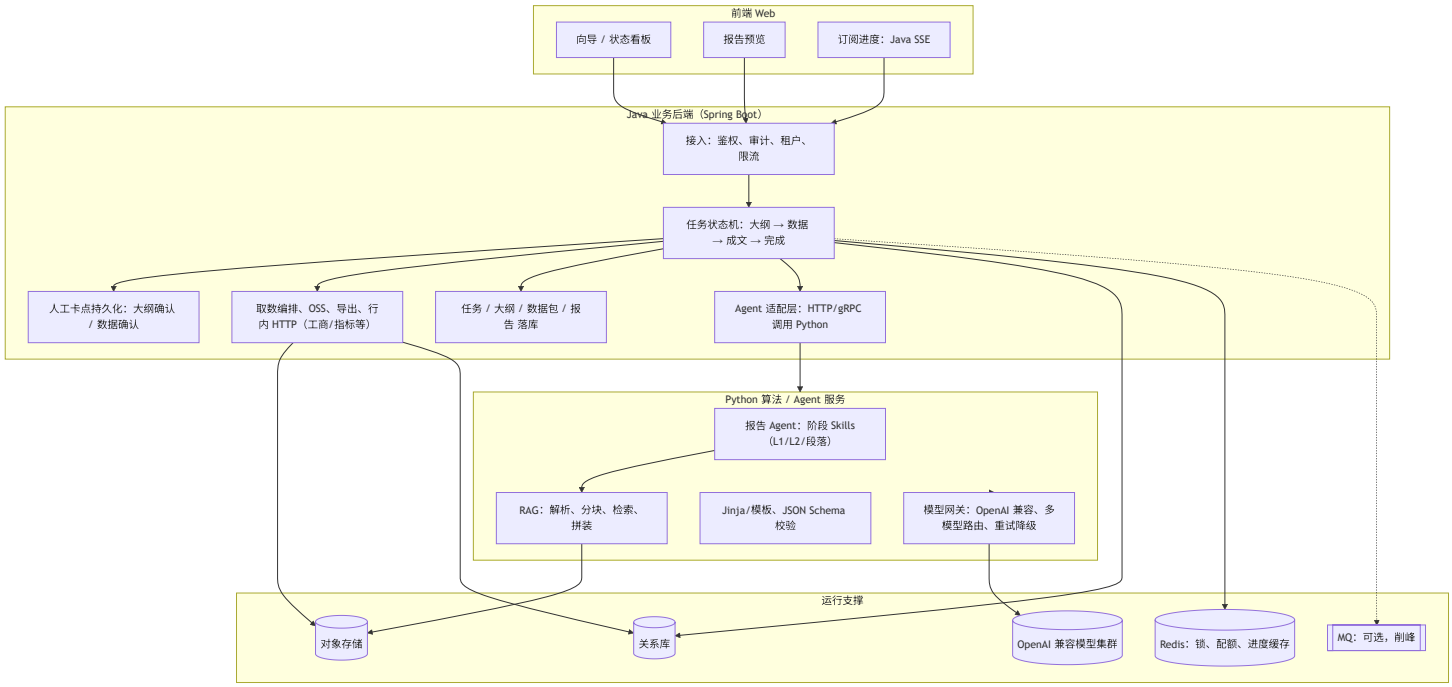
而不是把 Python 塞在浏览器与 Java「中间」给前端直连。

1. 架构设计原则

原则	架构含义
主链路优先	以 **Java 侧任务状态机** 为权威；Python 为**无状态或弱状态**的「阶段执行器」，输出结构化结果供 Java 持久化
受控生成	大纲确认、数据确认仍在 **Java / 前端** 闭环；Python 不绕过人工卡点
弱配置、强运行	MVP 预置知识库；Agent 侧可先用代码 + 模板仓库，逐步配置化

原则	架构含义
可观测、可恢复	**TraceId 由 Java 注入**，透传 Python；各阶段输入输出以 Java 库表为准
职责边界清晰	**Java**：事务、安全、集成、对象存储指针、编排；**Python**：模型调用策略、解析、RAG、Prompt 组合

2. 总体技术架构图



典型调用序列（简化）

1. 前端创建任务、提交表单 → **Java** 写库。
2. Java 进入「待生成大纲」→ **Java** 组请求上下文（任务 ID、租户 ID、traceId、脱敏后的业务字段、知识库片段指针）→ **调用 Python**：`generate_outline_l1` / `generate_outline_l2` 等。
3. Python 调模型、校验 JSON、返回结构化结果 → **Java** 保存大纲草稿、转「待确认大纲」。
4. 用户在前端确认 → **Java** 更新「最终知识单元清单」。
5. 数据准备：**Java** 调行内接口 +（可选）请求 Python 只做 **RAG 检索** 或 **Java 自调向量服务**——以团队分工为准；MVP 常见是 **RAG 在 Python**。
6. 段落生成：Java 按知识单元循环或批处理 → **调用 Python** `generate_section` → Java 拼接、落「完整报告」。

原则：任务真相源（Source of Truth）在 Java；Python 专注 **可替换的「智能计算」实现**。

3. Java 与 Python 的契约建议

内容	建议
协议	**REST（OpenAPI）** 或内网 **gRPC**；MVP 优先 OpenAPI，便于联调与网关转发
认证	Java 使用 **服务间 Token**（mTLS 或 HMAC）调用 Python；**禁止**前端直连 Python
上下文	每个请求带 `X-Tenant-Id` 、 `X-Task-Id` 、 `X-Trace-Id` 、阶段枚举
幂等	Java 生成 `idempotency-key` ，Python 侧对耗时写操作可去重
敏感数据	敏感字段由 Java **脱敏/最小化**后下传；原文索引用 **OSS 预签名 URL 或 objectId**

4. 技术选型建议

4.1 前端 Web

层级	推荐
框架	**React 18+**
UI	**Ant Design Pro** 或 **Arco Design**
进度	**SSE**（由 Java 转发或聚合 Python 执行进度；或 Java 轮询 DB + SSE 推前端）

4.2 Java 业务后端

层级	推荐
框架	**Java 17 + Spring Boot 3**
编排	**Spring State Machine 或自研枚举状态机**
调用 Python	**WebClient / OpenFeign**，超时与熔断单独配置
API	**REST + OpenAPI**；长任务 **202 + taskId**

4.3 Python 算法 / Agent 服务

层级	推荐
框架	**Python 3.11 + FastAPI**

层级	推荐
Agent 编排	**LangGraph** 或 **显式 Handler 分阶段**（与需求文档三阶段一致即可）
LLM	**OpenAI 兼容 Client** + 自研 **ModelRouter**（多模型路由、配额可在网关或 Java 前置，Python 内也可再兜一层）
RAG	**LangChain / LlamaIndex 择一轻用** 或自研 pipeline

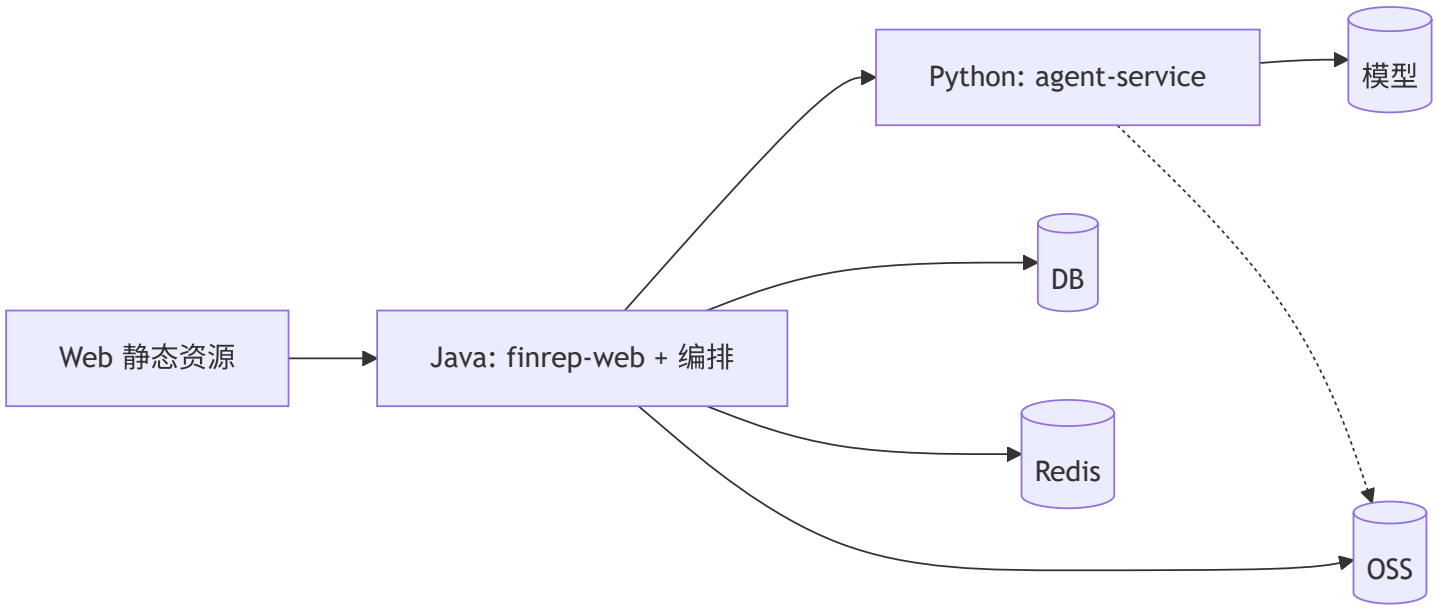
4.4 数据与中间件

组件	说明
主库	**MySQL 8 / PostgreSQL**，**租户 ID** 自设计起带入
Redis	锁、限流、配额计数
MQ	削峰时 **Java 投递**，可由 **Java Worker** 或 **Python Worker** 消费——**建议 MVP 由 Java 消费 MQ 再同步调 Python**，状态机不乱

4.5 导出与文档

- **Java**：**docx4j / POI**、与行内模板引擎集成较顺。
- **Python**：可直接生成中间 **Markdown** 交 Java 转正稿，或由 Java 全量组稿（看分工）。

5. 部署形态（三进程 / 三容器）



- **扩缩容**：Java API 与 **Python Agent** 可 **独立 HPA**；瓶颈常在 **LLM** 与 **Python CPU（解析/RAG）**。

6. 与需求文档的映射

模块	主要落点
任务状态、人工卡点、落库	**Java**
一级/二级大纲、段落级 LLM、JSON 校验、RAG	**Python**
行内 API、指标平台、OSS、导出	**Java**（Python 仅消费 OSS 只读链接时通过 Java 颁签）

7. 小结

- **先前若将全部 Agent 逻辑并入 Java**，与「**业务后端 Java + 算法 Agent Python**」的常见行内分工**不一致**；按您的界定，**Python 算法端应保留为独立服务**。
- **「单一后端 Java」** 宜理解为：**对外系统边界与主事务在 Java**；**Python 是内部「算法微服务」**，通过契约与 Java 集成。
- **前端只认 Java**，安全与多租户在 Java 收口，**算法端不直接暴露公网**。

8. 工程仓库建议

采用 **一个代码目录** 统一管理三端（与已实现目录 `finrep-report/` 对齐）：

子目录	说明
`finrep-report/frontend/`	Web 前端（React 等） 空骨架：`src/`（`pages`、`components`、`services` …）、`public/`
`finrep-report/backend/`	Java 业务后端多模块：`finrep-web`、`finrep-worker`、`finrep-infrastructure`（含 `integration/pythonagent` 调用算法服务）等；根包 `com.yuxin.finrep`
`finrep-report/algo/`	Python 算法 / Agent：`src/finrep_algo_agent/`（`api`、`skills`、`llm`、`rag`、`prompts`

子目录	说明
	等)

前端 **仅依赖 `backend` 的 API**； `backend` 经 **pythonagent** 适配层调用 **`algo`** 。三端在同一仓库内协作，部署仍可独立（三个制品/容器）。