Java代码智能分析与优化系统（超详细项目说明）

> 当前文档日期：2025-12-21

1. 项目目标与整体架构

本项目实现了一个**C/S（客户端/服务端）**的“代码智能分析与优化系统”：

- 客户端：命令行程序，负责与用户交互、把用户输入封装为文本命令，通过 TCP Socket 发送给服务端。
- 服务端：监听端口接收多个客户端连接（线程池并发），解析命令后调用大模型（DeepSeek / 通义千问）进行推理，再把结果回写给客户端。
- 模型调用：服务端通过 `HttpURLConnection` 向外部大模型 API 发起 HTTP POST 请求。
- 会话与历史：服务端把每个客户端的对话会话记录写入本地 `logs/` 目录，支持 `history / search / restore`。

1.1 技术选型回答（答辩常问）

- **客户端与服务端网络交互：使用 Socket / ServerSocket（TCP）**
  - 见服务端监听：src/com/codeai/server/Server.java
  - 见客户端连接：src/com/codeai/client/Client.java
- **大模型调用：使用 HttpURLConnection（HTTP POST JSON）**
  - DeepSeek：src/com/codeai/model/DeepSeekModel.java
  - Qwen：src/com/codeai/model/QwenModel.java
- **JSON：未引入第三方库，手写 JSON 构造/解析**
  - src/com/codeai/util/SimpleJson.java

2. 目录结构与模块职责

- 入口与启动：
  - src/com/codeai/Main.java
- 客户端：
  - src/com/codeai/client/Client.java
- 服务端：
  - Server（监听 + 线程池）：src/com/codeai/server/Server.java
  - ClientHandler（协议解析 + 调模型 + 会话管理）：src/com/codeai/server/ClientHandler.java
- 模型抽象与实现：
  - 接口：src/com/codeai/model/LLMModel.java
  - 工厂：src/com/codeai/model/ModelFactory.java
  - DeepSeek：src/com/codeai/model/DeepSeekModel.java
  - Qwen：src/com/codeai/model/QwenModel.java

- 日志与会话：
  - src/com/codeai/log/LogManager.java

3. 配置文件（config.properties）详解

配置文件位于项目根目录：config.properties

关键配置项（为了安全起见，本文不展示真实 key 值；实际文件中应使用你自己的密钥）：

- DeepSeek：
  - `deepseek.api.key`：API Key
  - `deepseek.api.url`：API Endpoint
- Qwen：
  - `qwen.api.key`
  - `qwen.api.url`
- 服务端：
  - `server.port`：监听端口（默认 8888）
  - `server.thread.pool.size`：线程池大小（默认 10）
- 日志：
  - `log.directory`：日志目录（默认 logs）
  - `log.file.prefix`：文件前缀（当前代码里主要使用 session_*.txt 命名）

读取位置：入口程序在 src/com/codeai/Main.java 使用 `Properties.load(new FileInputStream("config.properties"))` 加载配置。

3.1 启动时的配置校验逻辑

- Main.java 校验 `deepseek.api.key` / `qwen.api.key` 是否为空或包含占位符字符串；否则直接 `System.exit(1)`。
- 之后创建：
  - `ModelFactory`：Main.java
  - `LogManager`：Main.java
  - `Server`：Main.java

4. 服务端实现（Server + ClientHandler）

4.1 Server：ServerSocket 监听 + 线程池并发

实现文件：src/com/codeai/server/Server.java

关键点：

1. 构造函数中创建线程池：
   - `Executors.newFixedThreadPool(poolSize)`：Server.java
2. `start()` 中创建 `ServerSocket`：
   - Server.java
3. `while(true)` 里阻塞 `accept()` 等待连接：

    - Server.java
4. 每个连接交给线程池执行 `ClientHandler`：
    - Server.java

答辩要点：

- 这里是**典型的“主线程 accept + 线程池处理连接”**模型。
- 多客户端同时连接时，线程池大小决定并发处理上限。

4.2 ClientHandler：协议、会话、模型调用的核心

实现文件：src/com/codeai/server/ClientHandler.java

4.2.1 会话内存结构

- `sessionMessages`：`List<Map<String,String>>`，存储本会话的所有消息（role/content）。
    - 初始化：ClientHandler.java
- `sessionStartTime`：用于计算会话时长。
- `sessionLogged`：防止重复写日志。
- `currentSessionId`：用于“restore 后追加写回同一个 session 文件”。

4.2.2 I/O：基于 Socket 的字符流（UTF-8）

- 输入：`BufferedReader` 包装 `clientSocket.getInputStream()`
    - ClientHandler.java
- 输出：`PrintWriter` 包装 `clientSocket.getOutputStream()` 并设置 `autoFlush=true`
    - ClientHandler.java

4.2.3 欢迎信息 + 历史加载

- 欢迎菜单输出：
    - ClientHandler.java
- 加载历史：调用 `logManager.loadHistory(clientId)`
    - ClientHandler.java

4.2.4 主循环：读取命令 + 分派处理

- 主循环使用 `readCompleteCommand(in)` 读取客户端命令：
    - ClientHandler.java

支持命令类别：

1. 退出并保存关键词：`quit` / `q`

- 识别与处理入口：ClientHandler.java
- 退出时提示输入关键词（多行）：
    - 服务端提示：ClientHandler.java
    - 服务端读取关键词：再次调用 `readCompleteCommand(in)`：ClientHandler.java
- 保存会话：

    - 新会话 `logManager.logSession(...)`：ClientHandler.java
    - 恢复会话追加 `appendToSession(...)`：ClientHandler.java

2. 搜索：`search <keyword>`

- 入口：ClientHandler.java
- 调用：LogManager.searchSessions

3. 恢复：`restore <sessionId>`

- 入口：ClientHandler.java
- 加载消息：LogManager.loadSessionMessages
- 恢复后：
    - `sessionMessages.clear(); addAll(restored)`：ClientHandler.java
    - `currentSessionId=sessionId`：ClientHandler.java

4. 查看最近历史：`history`

- 入口：ClientHandler.java

5. 模型请求：`MODEL|PROMPT`

- 分割解析：`trimmedInput.split("\\|",2)`：ClientHandler.java
- 对比模式：`compare|...` 走 `handleCompareMode`：
    - ClientHandler.java
- 单模型：从 `ModelFactory.getModel(modelName)` 获取实现：
    - ClientHandler.java

4.2.5 单模型请求的“带上下文”实现

- 服务端把用户输入加入 `sessionMessages`：
    - ClientHandler.java
- 优先调用 `chatWithHistory(sessionMessages)`（上下文对话）：
    - ClientHandler.java
- 如果失败回退到 `chat(prompt)`（无上下文）：
    - ClientHandler.java
- 把模型输出加入会话：
    - ClientHandler.java
- 输出给客户端：
    - ClientHandler.java

4.2.6 compare 对比模式：并行调用两个模型 + 人工选择/合并

入口方法：

- ClientHandler.java

关键逻辑：

1. 同步把用户消息加入 session：

- ClientHandler.java

2. 用两个线程并行请求（deepseekThread / qwenThread）

- DeepSeek 线程创建：ClientHandler.java
- Qwen 线程创建：ClientHandler.java
- `start()`：ClientHandler.java
- `join()` 等待结束：ClientHandler.java

3. 输出时给每一行加编号（D1.. / Q1..）

- ClientHandler.java

4. 用户选择：

- prefer deepseek / prefer qwen / merge
- 读取选择使用 `in.readLine()`：
  - ClientHandler.java

5. merge 合并编辑模式

- `handleMergeMode(...)`：ClientHandler.java
- 合并输入格式示例：`D1-3,Q5-7,D10`
- 最终需要 `yes` 确认：
  - ClientHandler.java

4.2.7 连接断开时的会话自动保存

在 `finally` 里处理：

- ClientHandler.java

逻辑：

- 如果会话不为空且未保存：
  - 恢复会话：`appendToSession`
  - 新会话：`logSession`

4.3 多行命令协议（重点！答辩必问）

4.3.1 为什么需要多行协议

用户经常需要发送多行代码（含空行）。如果用“空行结束”，会误判代码中的空行，导致截断。

项目采用“**显式终止符**”方案：

- 客户端在多行消息末尾追加一行：`__END_OF_MULTILINE__`
- 服务端读取时遇到该行停止拼接

4.3.2 客户端发送（OutputStream 写入 UTF-8）

- 发送实现：`sendCommand(out, command, isMultiline)`
  - Client.java
- 多行条件：`isMultiline==true` 或 `command.contains("\n")`
  - Client.java

4.3.3 服务端接收（readCompleteCommand）

- 接收实现：ClientHandler.java

关键点：

- 先读第一行
- 然后“仅在缓冲区已有后续数据时”继续读（避免单行命令被阻塞）
- 读到 `__END_OF_MULTILINE__` 停止

> 项目里也有一份单独说明文件：
>
> - MULTILINE_IMPLEMENTATION.txt
> - TEST_MULTILINE.txt

5. 客户端实现（Client）

实现文件：src/com/codeai/client/Client.java

5.1 Socket 连接与线程模型

- 连接：`new Socket(host, port)`
  - Client.java
- 启动一个接收线程专门打印服务端输出：
  - Client.java

答辩要点：

- 主线程负责读用户输入并发送。
- 接收线程负责读服务端输出并实时打印。
- 这是“半双工命令行交互”的常见实现方式。

5.2 交互模式的输入规则

- 主要命令提示：
  - Client.java
- `quit/exit` 退出流程（两段多行发送）：
  1. 发送 quit（多行模式，确保服务端能完整接收）
     - Client.java
  2. 等待服务端提示“请输入关键词”
     - 标志位设置在接收线程：Client.java
     - 主线程等待：Client.java

3. 用户输入多行关键词，以 `---` / `__END__` 结束
    - Client.java
4. 发送关键词（多行模式）
    - Client.java

- `MODEL|PROMPT` 多行输入：
    - 只要第一行包含 `|`，客户端就提示继续输入，直到 `---` / `__END__`。
    - Client.java

5.3 directCommand 模式（可用于脚本/自动化）

- `Client.main` 支持第三个参数 `directCommand`
    - Client.java

6. 模型层（LLMModel + Factory + 两种实现）

6.1 抽象接口 LLMModel

文件：src/com/codeai/model/LLMModel.java

- `chat(prompt)`：单次请求（无上下文）
- `chatWithHistory(messages)`：带会话历史（上下文对话）
- `getModelName()`：显示名称

6.2 ModelFactory：集中创建与按名获取

文件：src/com/codeai/model/ModelFactory.java

- 在构造函数里创建两种模型实例并放入 Map：
    - ModelFactory.java
- `getModel(modelName)`：通过 key 获取
    - ModelFactory.java

6.3 DeepSeekModel：HttpURLConnection POST JSON

文件：src/com/codeai/model/DeepSeekModel.java

关键实现点：

- `URL url = new URL(apiUrl)`
- `HttpURLConnection conn = (HttpURLConnection) url.openConnection()`
- `POST` + `Content-Type: application/json` + `Authorization: Bearer <apiKey>`
    - DeepSeekModel.java
- 写请求体（UTF-8）：
    - DeepSeekModel.java
- 读响应 `HTTP_OK`：
    - DeepSeekModel.java
- 非 200：读 errorStream 并抛异常：

- DeepSeekModel.java

6.4 QwenModel：同样 HttpURLConnection POST JSON

文件：src/com/codeai/model/QwenModel.java

与 DeepSeek 的主要差异：

- 请求体结构不同：
  - DeepSeek: `{"model":"deepseek-chat","messages":[...]}`
  - Qwen: `{"model":"qwen-plus","input":{"messages":[...]}...}`
  - 见构建： SimpleJson.java
- 响应解析字段不同：
  - DeepSeek: 查找 `"content":"..."`
  - Qwen: 查找 `"text":"..."`
  - 见解析： SimpleJson.java

7. JSON 工具（SimpleJson）

文件：src/com/codeai/util/SimpleJson.java

7.1 构建请求 JSON

- DeepSeek：
  - 单次：SimpleJson.java
  - 带历史：SimpleJson.java
- Qwen：
  - 单次：SimpleJson.java
  - 带历史：SimpleJson.java

7.2 解析响应 JSON（字符串搜索 + 转义处理）

- DeepSeek 提取 `content`：
  - SimpleJson.java
- Qwen 提取 `text`：
  - SimpleJson.java
- `findJsonStringEnd`：处理转义引号：
  - SimpleJson.java

答辩要点：

- 这是“轻量无依赖”的实现，优点是无需引入 jar；缺点是对复杂 JSON 不够健壮。

8. 日志与会话（LogManager）

文件：src/com/codeai/log/LogManager.java

8.1 目录与命名规则

- 根目录来自 `config.properties` 的 `log.directory`：
  - 由 Main.java 读取并传入 `LogManager`
- 每个用户（userId=客户端 IP）独立一个目录：
  - LogManager.java
- 每个会话一个文件：`logs/<userId>/session_yyyyMMdd_HHmmss_SSS.txt`
  - LogManager.java

8.2 写会话 logSession

- 会话头部信息：会话ID/时间/时长/消息数/关键词
  - LogManager.java
- 逐条写消息：role 区分“用户/助手”
  - LogManager.java

8.3 加载最近历史 loadHistory

- 找到该用户目录下最新的 session 文件（按 lastModified）：
  - LogManager.java
- 整文件读出并返回：
  - LogManager.java

8.4 searchSessions：全文搜索 + 提取 sessionId/title

- 遍历该用户所有 session_*.txt
  - LogManager.java
- 如果文件内容包含关键字：
  - 提取 `会话ID:`
    - LogManager.java
  - 提取标题（优先关键词，否则时间）：
    - LogManager.java

8.5 restore：loadSessionMessages（把 txt 反解析为 role/content）

- 识别 `【用户】` / `【助手】` 开始新消息
  - LogManager.java
- 忽略分隔符行（---- / ====）
- 收集 content 行并拼接

8.6 appendToSession：把“完整消息列表”覆盖写回文件

- 注意：这里不是增量 append，而是 `new FileOutputStream(sessionFileName,false)` 全量覆盖
  - LogManager.java

9. 运行方式（答辩演示用）

9.1 编译

建议使用 UTF-8 编码编译（Windows 下避免中文乱码）：

- `javac -encoding UTF-8 -d bin src/com/codeai/Main.java src/com/codeai/server/*.java src/com/codeai/client/*.java src/com/codeai/model/*.java src/com/codeai/log/*.java src/com/codeai/util/*.java`

9.2 启动服务端

- `java -cp bin com.codeai.Main`

9.3 启动客户端

- 交互模式：`java -cp bin com.codeai.client.Client localhost 8888`
- directCommand：`java -cp bin com.codeai.client.Client localhost 8888 "deepseek|解释这段代码\n..."`

10. 常见追问与标准回答（建议背熟）

1. 问：客户端与服务端用的是什么通信方式?

- 答：TCP Socket。服务端 `ServerSocket.accept()`，客户端 `new Socket(host,port)`。
  - Server.java
  - Client.java

2. 问：模型调用是 Socket 还是 HTTP?

- 答：对外调用大模型是 HTTP（`HttpURLConnection`），并发送 JSON 请求体。
  - DeepSeekModel.java
  - QwenModel.java

3. 问：为什么要线程池?

- 答：避免每个连接都创建无限线程；线程池可控并发与资源占用。
  - Server.java

4. 问：对话上下文是怎么实现的?

- 答：服务端维护 `sessionMessages`，每次请求把用户/助手消息追加进去，然后把整个 messages 列表传给 `chatWithHistory`。
  - ClientHandler.java

5. 问：多行代码怎么保证不被截断?

- 答：客户端以 `---/__END__` 结束输入，然后网络层使用 `__END_OF_MULTILINE__` 作为真实协议结束符；服务端读到它停止拼接。
    - Client.java
    - ClientHandler.java

11. 已知限制（如被问到可坦诚说明）

- `SimpleJson` 是字符串级别解析，遇到复杂 JSON/嵌套字段可能不健壮。
- 会话日志是纯文本文件，search 是全文扫描；会话多了可能慢。
- API Key 当前在本地 `config.properties` 中读取，建议不要提交到公开仓库（应使用环境变量或本地忽略）。