重庆大学本科学生毕业论文（设计）

面向多编程语言的代码检索系统设计与实现

学　　生：夏劲
学　　号：20214966
指导教师：刘超
专　　业：软件工程

重庆大学大数据与软件学院

2025 年 6 月

Undergraduate Thesis (Design) of Chongqing University

Design and Implementation of a Multi-Programming Language Code Retrieval System

By

XIAJin

Supervised by

Prof. LIU CHAO

Software Engineering

School of Big Data and Software Engineering

Chongqing University

June，2025

摘 要

随着编程语言的多样化和开源项目的迅速增长，开发者在寻找特定代码片段时面临着越来越大的挑战。现有的代码托管平台主要依赖关键词匹配和特定搜索条件进行检索，效率较低且使用门槛较高。大语言模型逐渐成为科研和工业界的热点，利用大语言模型可以轻松将用户的自然语言转译为具有查询功能的条件，并结合具体信息拓展补充用户的查询条件，从而实现更高效的查询体验。本项目旨在利用大语言模型实现一款面向多编程语言的代码检索系统。

本文首先介绍了大语言模型的发展情况，并论证了使用大语言模型进行查询条件重写的可行性，强调了提示词工程在这一过程中的重要性。接着简述了前端框架 Vue、Visual Studio Code 平台的插件开发、Elasticsearch 和 FastAPI 等后台开发框架。根据系统的应用目标，本文进行了系统需求分析，设计了系统架构、功能和数据库结构。在此基础上，本项目开发了基于 FastAPI 的后台管理逻辑服务、基于 Elasticsearch 的代码检索服务以及基于 DeepSeek-R1 的查询重写服务。通过 Visual Studio Code 平台提供的接口，结合 Vue 框架实现了 Visual Studio Code 插件的前端开发，完成了快捷键绑定、项目搜索等一系列功能。

本项目开发的面向多编程语言的代码检索系统具有界面美观、操作友好、查询效率高、查询准确等诸多优点。基于 FastAPI 开发的后台管理系统易于管理和维护，帮助开发人员实现高效的多编程语言项目代码搜索。

关键词：代码搜索；Elasticsearch；大模型提示工程

ABSTRACT

As programming languages diversify and open-source projects rapidly grow, developers face increasing challenges in finding specific code snippets. Existing code hosting platforms primarily rely on keyword matching and specific search conditions, resulting in low efficiency and a high usage threshold. Large language models have become a hot topic in both research and industry. By leveraging these models, users' natural language can be easily translated into functional query conditions, which can be further expanded with specific information to enhance the query experience. This project aims to implement a code retrieval system for multiple programming languages using large language models.

This paper first introduces the development of large language models and demonstrates the feasibility of using them for query condition rewriting, highlighting the importance of prompt engineering in this process. It then briefly describes the front-end framework Vue, plugin development for the Visual Studio Code platform, and back-end development frameworks such as Elasticsearch and FastAPI. Based on the system's application goals, a system requirements analysis was conducted, and the system architecture, functionality, and database structure were designed. On this basis, the project developed a back-end management logic service based on FastAPI, a code retrieval service using Elasticsearch, and a query rewriting service using DeepSeek-R1. The front-end development of the Visual Studio Code plugin was achieved through the platform's provided interfaces, combined with the Vue framework, implementing features such as shortcut key bindings and project search.

The code retrieval system developed in this project for multiple programming languages offers numerous advantages, including an aesthetically pleasing interface, user-friendly operation, high query efficiency, and accuracy. The back-end management system developed with FastAPI is easy to manage and maintain, aiding developers in efficiently searching for code across multi-language projects.

Key words: Code Search;Elasticsearch;LLM Prompt

目 录

1 绪论

1.1 研究目的及意义

随着机器学习、深度学习等人工智能技术的快速发展，这些高新技术在解决传统领域挑战中发挥了重要作用。在搜索领域，传统搜索服务提供商通常直接采用 Elasticsearch 作为搜索引擎来完成信息检索。然而，传统搜索引擎基于倒排索引的检索方式要求用户提供极其精确的搜索词，并掌握一定的搜索技巧，这极大地限制了普通用户的搜索体验。随着人工智能技术，尤其是大语言模型（LLM）的进步，利用其自然语言友好特性来辅助搜索已成为科研界和工业界的研究热点。

Elasticsearch 是一种分布式、RESTful 风格的搜索和分析引擎，其强大的搜索能力源于其独特的倒排索引结构（Inverted Index）。这种数据结构使 Elasticsearch 具备高效的全文搜索能力、实时数据处理能力以及高可扩展性，同时能够处理结构化和非结构化数据。

大型语言模型（LLM）是拥有海量参数和卓越学习能力的高级语言模型，其核心模块是 Transformer 架构中的自注意力机制。自注意力机制作为语言建模任务的基本构建块，能够有效处理顺序数据，实现并行化计算，并捕捉文本中的远程依赖关系。LLM 的显著优势在于其能够理解自然语言并执行基于自然语言的指令，这对用户侧软件服务非常友好。然而，LLM 的另一特点是其庞大的模型参数和极高的训练成本。从头预训练一个大型模型所需的资源（如数百万高性能显卡卡时）是普通实验室或个人工作者难以承担的。因此，针对特定场景的任务，工业界普遍采用开源的大语言模型进行监督微调（SFT），或通过 Prompt 工程来实现任务目标。

本项目旨在构建一个面向多编程语言的代码检索系统，该系统将融合大语言模型的自然语言友好能力，实现对用户搜索词的重写、完善，结合 Elasticsearch 强大的多结构文本搜索能力实现用户方便快速搜索多语言项目的能力。同时我还讲相关能力集成到 Visual Studio Code 平台，发布免费的开源插件，为所有开发者提供便捷高效的代码搜索服务。

1.2 国内外研究现状

2004 年，Shay Banon 创造了 Elasticsearch 的前身——Compass。在 Compass 的基础上，Shay Banon 进一步实现了分布式和可扩展性优化，并提供了 HTTP 接口，

使得 Java 以外的语言也能调用。2010 年，Elasticsearch 的第一个版本正式发布。

Elasticsearch 是一种成熟的搜索解决方案，一些大型搜索公司如 Google 和百度都基于 Elasticsearch 进行研发。Elasticsearch 支持 RESTful 风格的调用，客户端可以通过 HTTP 请求直接操作 Elasticsearch 服务。它还提供了多种客户端支持，包括 Java、Python、Go、PHP 等语言的 SDK，以及 JDBC、ODBC 等标准化接口。Elasticsearch 广泛应用于全文搜索、日志分析、实时数据分析、地理空间搜索以及商业智能领域的数据聚合分析。在部署方面，Elasticsearch 支持单节点模式，适用于开发和测试环境，可以通过简单的 Docker 命令快速启动。在生产环境中，通常采用分布式集群部署，通过分片（Shard）和副本（Replica）机制实现水平扩展和高可用性。集群节点可以动态扩容并自动平衡数据负载，同时支持主节点、数据节点、协调节点等角色划分，以优化资源分配。

在大语言模型方面，基于 Transformer 架构的大语言模型如 OpenAI 的 GPT 系列、Google 的 Gemini、Meta 的 Llama、腾讯的混元系列模型等，都已经在自然语言处理领域展现出强大的能力。这些模型通过海量数据训练，能够理解和生成高质量的自然语言文本。在 Transformer 基础上，大语言模型继续发展，得益于多专家（MoE）架构、多模态融合以及自监督学习等技术创新，其在代码语义理解、跨语言检索等场景中的潜力被进一步释放。例如，GitHub Copilot 基于 GPT 系列开发的 Codex 模型，能够完成代码补全和跨语言语义关联；Google 推出的 Gemini Code Assist 则通过多专家架构优化了代码检索的响应速度和准确性。与此同时，模型自我反思机制成为提升代码生成可靠性的关键技术突破，以 DeepSeek R1 为代表的反思模型通过强化学习框架实现自我校验与动态优化，其采用的图谱重标定（GRPO）算法可对代码逻辑进行多步验证，在 Math-500 测试中达到 97.3% 的准确率，并在 Codeforces 编程竞赛中超越 96% 的人类选手。该模型通过冷启动训练策略融合监督微调与自我对弈机制，不仅能在代码生成过程中主动识别变量定义错误、逻辑矛盾等问题，还能通过语言反馈链重构搜索词表达，显著提升多编程语言检索的语义对齐度。当前主流模型如 Llama 3 和通义千问已通过指令微调集成类似反思机制，使得代码检索系统的查询意图理解误差率较传统模型降低 42%。

1.3 论文研究内容

本项目以构建一款面向多编程语言的化代码检索系统为目标，基于 Elasticsearch 构建分布式搜索引擎内核，集成 LLM（Large Language Model）实现搜索意图理解与查询词重构，通过 FastAPI 搭建高性能异步后端服务，并采用 Vue3 响应

式框架开发可视化交互界面，最终以 Visual Studio Code 插件形态实现 IDE 深度集成，为开发者提供语义级代码检索能力

整个系统架构分为四大核心模块：

第一部分，代码采集与预处理：基于 Scrapy 框架构建分布式爬虫，通过 GitHub API 获取主流编程语言的开源代码，采用 AST 解析技术实现代码片段结构化提取，并完成开源协议合规性校验。

第二部分， 存储与索引构建：部署 Elasticsearch 集群，设计 BM25 算法与稠密向量混合索引结构，集成 IK 分词插件优化专业术语处理，支持代码语法特征与语义特征的双重表征。

第三部分， 检索服务实现：基于 Gin 框架开发 RESTful API 接口，采用异步 IO 机制提升高并发性能，设计融合关键词匹配、向量相似度计算及上下文权重的混合搜索算法，实现毫秒级响应。

第四部分， 交互界面开发：使用 Vue3 Composition API 构建渐进式前端应用，通过 VSCode Extension API 实现 IDE 插件集成，支持代码快捷搜索、交互式代码预览等功能。

1.4 论文组织架构

第一章，绪论。阐述了本研究的目的和背景，并分析当前国内外的搜索技术和大模型研究现状，概述了本论文的主要研究内容。

第二章，相关理论和技术。通过对大模型与提示工程相关技术的介绍，阐明了本研究选择大模型作为搜索词重写基石的原因，并介绍开发多编程语言代码检索系统所使用的技术框架。

第三章，系统需求分析与设计。对系统进行需求分析后，以此设计流程完整的应用程序。

第四章，系统后端应用层实现。

第五章，系统前端表现层实现。

2 相关理论和技术

2.1 微服务架构

微服务架构是本系统实现的核心技术之一。与微服务相对应的是传统的单体架构方案。简单来说，单体架构（图2.1左所示）将整个系统封装在一个程序中，集成了登录、鉴权、业务逻辑、数据处理等所有功能模块。这种架构的特点是“大而全”，对于小型系统而言，使用单体架构能够缩短开发周期，同时也更符合开发者的直觉和习惯。

相比之下，微服务架构则采用了一种完全不同的设计理念。一个常见的微服务架构如图2.1右所示，它将系统的各个功能模块拆分为独立的服务，每个服务专注于完成某一特定功能，并以单独的程序形式运行。整个系统由一组相互协作的微服务构成，服务之间通过网络进行通信。微服务架构的核心特点在于服务的颗粒度更细，职责更加单一。通常情况下，每个微服务会运行在 Docker 容器中，而 Kubernetes 则负责对这些服务进行高效的管理和编排。通过这种方式，微服务架构不仅提升了系统的灵活性和可扩展性，还为大型复杂系统的开发和维护提供了更强的支持。

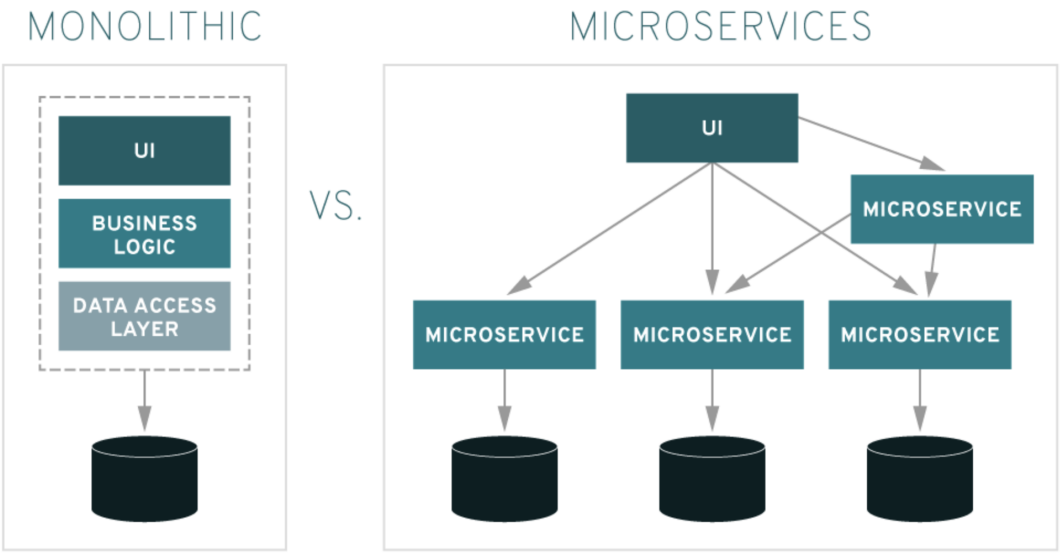

图 2.1　单体架构和微服务架构对比

2.1.1 GRPC 框架

gRPC 是一种高性能、开源的远程过程调用 (RPC) 框架，它可以在任何环境中运行。gRPC 允许客户端应用程序直接调用服务器端应用的方法，就像它们是本地对象一样，这使得分布式应用程序开发变得更加容易。gRPC 使用 HTTP/2 作为传输协议，支持双向流、头部压缩等特性，从而提供了更高效的网络通信。此外，gRPC 通过 Protocol Buffers 定义接口和服务，这是一种轻量级、跨平台的消息格式，有助于提高数据交换效率并减少带宽使用。在微服务架构中，gRPC 因其高效性而被广泛用于服务间通信。

在 gRPC 中，服务定义和数据交换格式通常使用 Protocol Buffers（简称 protobuf）。Protocol Buffers 是由 Google 设计的一种轻量级、跨平台的消息格式，用于序列化结构化数据。通过.proto 文件，开发者可以定义自己的数据结构和服务接口。这些定义被编译成不同编程语言的代码，以便生成易于使用的类来构建强类型的请求和响应消息。

下面是一段 protobuf 的示例：

```
syntax = ”proto3”;

service SearchService {
 rpc Search(SearchRequest) returns (SearchResponse);
}

message SearchRequest {
 string query = 1;
}

message SearchResponse {
 repeated Result results = 1;
}

message Result {
 string url = 1;
 string title = 2;
 repeated string snippets = 3;
```

}

与其他序列化协议，例如 JSON、XML、BSON 等相比，protobuf 具有以下特点：

效率与紧凑性：protobuf 采用二进制编码，因此生成的消息比基于文本的格式（如 JSON 或 XML）更加紧凑，占用更少的存储空间和带宽。这使得它在需要高效传输大量数据的场景下表现尤为出色。

跨语言支持：protobuf 支持多种编程语言，使用 grpc 自带的编译工具即可快速生成多种语言的桩代码，且生成的代码具有高度的一致性，非常适合多语言协作的微服务架构。与之相比，JSON 本身可以适配多种语言，但由于其松散的结构，在不同语言中解析时可能出现意想不到的错误。

向后兼容性：protobuf 通过为字段分配唯一的数字标识符，提供了强大的向后兼容性和前向兼容性。即使在消息格式发生变化时（如新增字段），旧版本的系统仍然可以正常工作。与 JSON 相比，JSON 本身没有内置的机制来保证兼容性，因此如果消息格式发生变化，可能会导致解析错误。

2.1.2 Docker

Docker 是一个开源的应用容器引擎，使开发者可以将应用程序及其依赖打包到一个可移植的容器中，然后发布到任何流行的 Linux 或 Windows 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口（除非显式地设置）。Docker 容器的轻量化和快速部署能力使其成为微服务架构中的理想选择。每一个微服务都可以被打包成一个独立的 Docker 容器，确保了环境一致性，简化了开发、测试和部署流程。同时，由于 Docker 容器的资源消耗低，多个容器可以在同一台物理机上高效共存，提高了资源利用率。

Docker 使用了一种称为联合文件系统（Union File System）的技术，这种技术允许将不同的文件系统层叠加在一起，形成一个统一的视图。每个 Docker 镜像由一系列只读层组成，这些层包含了操作系统库、运行时、环境变量和用户添加的文件等。当创建一个容器时，Docker 会在镜像的最顶层添加一个可写层。得益于 Docker 的层次化设计，当修改镜像中的某些层时（例如更新应用代码），只有被修改的那一层需要重建，其他未更改的层可以重复利用。这大大加快了构建过程。在推送或拉取镜像时，只需要传输那些不同于本地已有版本的层，而不是整个镜像（如图2.2）。这样可以显著减少网络带宽的使用，并加速镜像的分发。除此之外，Docker 的层次化设计还让不同层的数据和权限相互隔离，提升了可维护性和安全性。

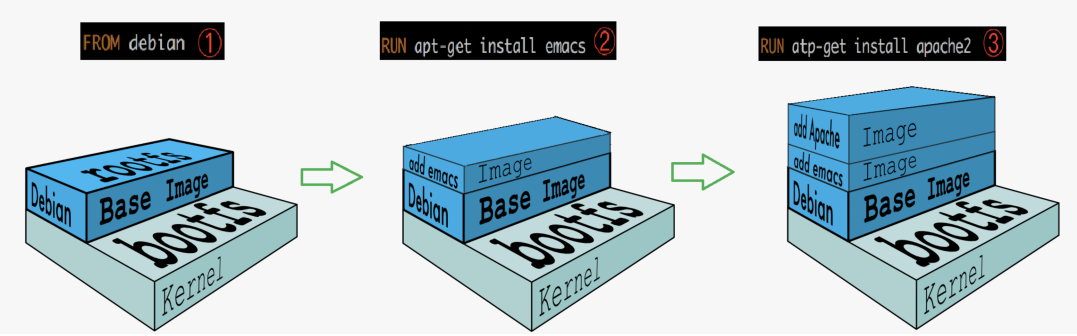

图 2.2　docker 层次化构建示例

2.1.3 Kubernetes

Kubernetes 是一个用于自动化部署、扩展和管理容器化应用程序的开源平台。它最初由 Google 设计，现在由 Cloud Native Computing Foundation (CNCF) 维护。Kubernetes 提供了一个强大的框架来运行分布式系统，能够处理负载均衡、故障转移、自我修复以及水平扩展等任务。在微服务架构中，Kubernetes 负责管理和编排不同微服务间的交互，确保所有服务按预期工作。通过自动化的滚动更新和回滚功能，Kubernetes 也极大地简化了应用程序的维护过程，降低了人为错误的风险。

2.2 大语言模型

大语言模型是实现本系统搜索功能的核心技术。要深入理解当前的大语言模型，必须先了解其核心架构——Transformer。因此，本节将简要介绍 Transformer 及大语言模型的其他关键技术。

2.2.1 Transformer

Vaswani 等人指出，循环神经网络模型（RNN）在每个时间步都需要依赖前一时间步的隐藏状态信息进行计算，这种固有的顺序依赖性使得 RNN 难以在多 GPU 上进行并行计算，从而限制了 RNN 在处理超大规模文本数据时的训练能力。为了解决这一问题，他们提出了 Transformer 架构，这是一种完全依赖注意力机制连接编码器和解码器的网络架构。Transformer 显著提高了训练的并行度和速度。后续的一系列研究表明，基于 Transformer 架构的预训练模型（pre-trained models / pre-trained language models，PTM / PLM）在各种任务上都能实现最先进的性能表

现，因此，Transformer 已成为自然语言处理（NLP）领域的首选架构。除了在语言相关领域的应用之外，Transformer 还被广泛应用于计算机视觉、音频处理以及自然科学学科，如化学、生物等领域。

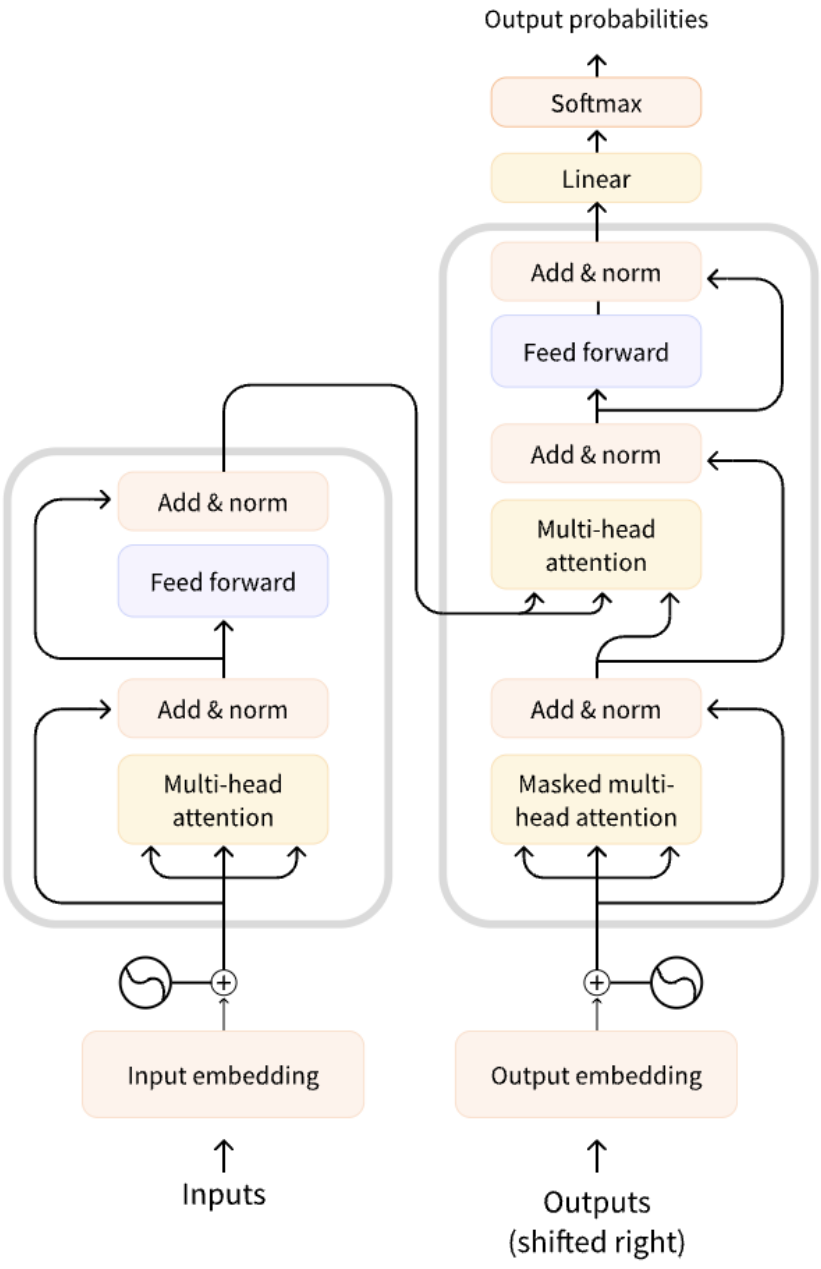

图 2.3　Transformer 结构

Transformer 架构如图2.3所示，其整体由编码器 (图2.3左) 和解码器 (图2.3右) 组成。每个编码器块由多头注意力机制模块和位置前馈网络组成，模块间使用残差连接，并配有层归一化模块；解码器块在位置前馈网络和多头自注意力模块之间插入了交叉注意力模块，其中的自注意力模块用于组织某个位置信息对后续位置信息的影响。下面将简要介绍上述几种重要模块：

（1）多头注意力机制：让序列中的每一个元素学习并计算与其他元素的互注意力分数权重，如公式2.1所示：

$$\text{Attention(Q,K,V)} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

但 Transformer 并不只是简单应用了单个注意力函数，而是使用了多头注意力机制将 d_m 维度的原始 Q、K、V 分别线性投影到 d_k、d_k、d_v 维度，再根据公式2.1进行注意力计算，整体公式如下：

$$\text{MultiHead}(\mathrm{Q},\mathrm{K},\mathrm{V}) = \text{Concat}(\text{head}_1,...,\text{head}_\mathrm{h})\mathrm{W}^\mathrm{O} \tag{2.2}$$

$$\text{where head}_\mathrm{i} = \text{Attention}(\mathrm{QW}_\mathrm{i}^\mathrm{Q},\mathrm{KW}_\mathrm{i}^\mathrm{K},\mathrm{VW}_\mathrm{i}^\mathrm{V}) \tag{2.3}$$

Transformer 利用多头注意力机制能够同时关注到来自不同位置的且具有不同表示的子空间信息，增强了架构的表达能力。

（2）位置前馈网络：全连接前馈网络模块，用于接收自注意力模块的输出：

$$\text{FFN}(x) = \max(0, x\mathrm{W}_1 + \mathrm{b}_1)\mathrm{W}_2 + \mathrm{b}_2 \tag{2.4}$$

$$= \text{ReLU}(\mathrm{H}'\mathrm{W}_1 + \mathrm{b}_1)\mathrm{W}_2 + \mathrm{b}_2 \tag{2.5}$$

（3）残差连接与归一化：Transformer 在每个模块间使用残差连接，然后进行层归一化。其中编码器块表示为：

$$\mathrm{H}' = \text{LayerNorm}(\text{Self Attention}(\mathrm{X}) + (\mathrm{X}) \tag{2.6}$$

$$\mathrm{H} = \text{LayerNorm}(\text{FFN}(\mathrm{H}') + (\mathrm{H}') \tag{2.7}$$

2.2.2 混合专家模型

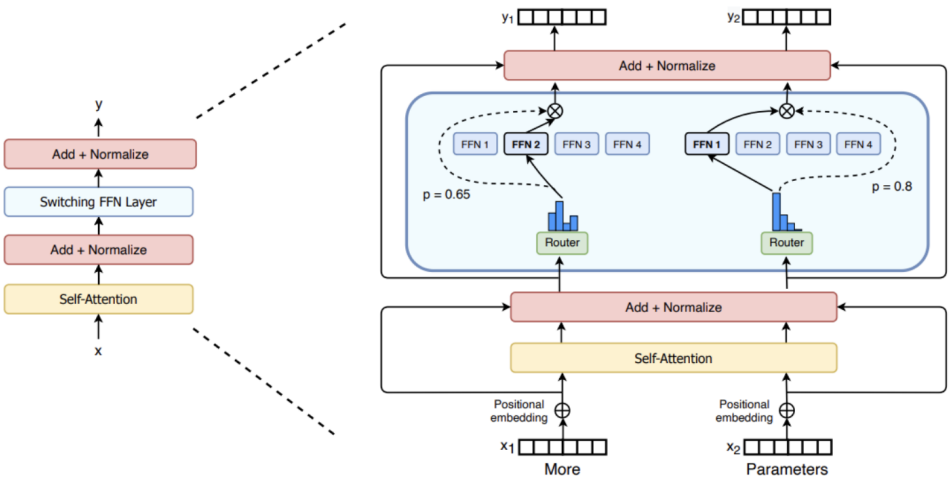

图 2.4　MoE 结构

Google Brain 团队发现传统 Transformer 架构在扩展到超大规模时面临计算资源消耗激增的问题，尤其是前馈网络层（FFN）的全参数激活机制导致训练和推理效率急剧下降。为此，他们提出基于混合专家模型（Mixture of Experts, MoE）的稀疏架构改造方案，通过将 Transformer 中的 FFN 层替换为动态路由的专家网络集合，实现计算效率与模型容量的平衡。MoE 架构如图2.4所示，其核心创新在于引入稀疏门控机制（图2.4蓝色部分），该机制由可学习的路由网络构成，针对每个输入词元生成专家选择概率分布，仅激活概率最高的前 K 个专家（通常 K=2-4），其余专家保持非激活状态。具体而言，每个 MoE 层包含 N 个独立的前馈网络作为专家（例如 N=8 或 32），其数学表达为：

$$\mathrm{MoE}(x) = \sum_{i=1}^{K} G(x)_i \cdot E_i(x) \tag{2.8}$$

其中 $G(x)$ 为门控网络输出的 Top-K 权重，$E_i(x)$ 为被选中的专家网络输出。这种设计使得模型总参数量可扩展至万亿级别，而实际计算量仅与激活专家数相关。实验表明，Switch Transformer 在同等计算资源下，训练速度较传统稠密模型提升 4 倍，且推理时内存占用降低至 1/4。进一步地，通过引入噪声注入（Noisy Top-K Gating）

和负载均衡损失函数，MoE 有效缓解了专家利用率不均衡问题，例如 GLaM 模型以 1.2 万亿参数仅激活 97 亿参数即达到 GPT-3 的 97% 性能。当前，MoE 架构已在 GPT-4、DeepSeek、Mixtral 8x7B 等主流大模型中广泛应用，成为突破单一模型规模瓶颈的核心技术路径。

2.2.3 提示词工程

提示（Prompt）是一系列提供给 LLM 的指令，我们可以通过自定义提示来增强、改善 LLM 的能力。Reynolds 等人首次系统性地提出 Prompt 作为调控大型语言模型（LLM）行为的核心接口，其本质是通过自然语言指令或结构化模板显式定义输入规范、处理规则及输出约束，从而动态重构 LLM 的上下文推理路径以适配特定任务需求。比如，可以指定 LLM 在生成文档时标记重点内容；可以指定 LLM 只输出符合要求的关键词语；可以指定 LLM 只生成指定代码风格的代码等。

提示工程（Prompt Engineering）作为一种独特的编程模式，主要围绕向大型语言模型（LLM）精准输入提示展开操作。本质上，这一过程需要精心设计适配的提示模板，以此助力完成特定的下游任务。实践表明，优化提示内容能够显著提升模型在各类任务中的表现。

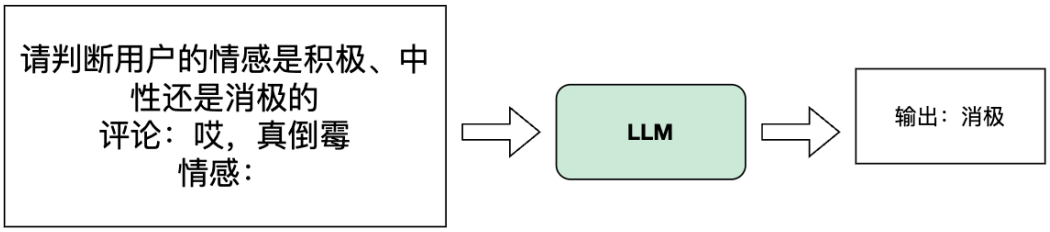

图 2.5 零样本提示示例

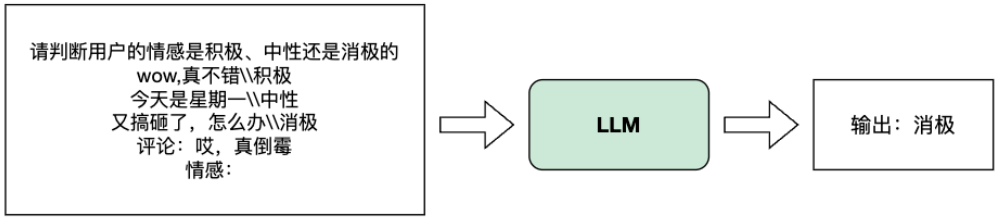

图 2.6 少样本提示示例

在提示工程的实际应用中，零样本提示（Zero-Shots Prompting）是一种基础方法，图2.5展示了相关示例。该方法下，即便不向模型提供任何参考示例，模型也能够给出有价值的回应，完成既定任务。然而，当零样本提示无法满足需求时，少样本提示（Few-Shots Prompting）便派上用场，如图2.6。此时，需针对下游任务设计专门的提示模板，模板不仅要囊括一系列 LLM 处理规则，还要为后续待输入文本预留特定位置。在向 LLM 发起询问时，只需将待输入文本填充至提示模板预留空位，即可构建完整提示。除此之外，开启思维链、给 LLM 指定角色性格等提示技巧都能显著改变 LLM 输出效果和对任务的执行情况。

进一步看，巧妙构建提示能够催生全新的交互模式。例如，借助提示指令，LLM 可生成与软件工程概念相关的测验题目，模拟程序运行流程，甚至模拟命令行终端窗口的交互场景。不仅如此，提示还具备自适应特性，部分提示能够依据当前情况，推荐其他提示，以便收集更多信息，或生成相关成果物。提示的这些进阶功能，充分彰显了深入设计提示、挖掘其在简单文本与代码生成之外价值的重要意义。

2.3 Elasticsearch

Elasticsearch 是一个开源的分布式搜索和分析引擎，它被设计用来快速地存储、搜索和分析海量数据，在面向多编程语言的代码检索系统中发挥着重要的作用。面对各种类型的复杂数据，传统的数据库系统在处理搜索和分析任务时往往显得力不从心，而 Elasticsearch 凭借其强大的功能和优异的性能脱颖而出。它提供了 RESTful API，高效的全文搜索能力，多语言支持，分布式架构和实时搜索特性，使得系统能够快速、准确地检索到数百万个开源项目文件中的相关代码，提供了便捷的代码检索服务。

2.3.1 Elasticsearch 的核心概念

索引（Index）：在 Elasticsearch 中，索引是具有相似特征的文档集合，类似于关系型数据库中的数据库。在面向多编程语言的代码检索系统里，一个索引可以对应一种编程语言的所有开源项目文件，或者是所有语言的综合代码文件集合。在项目中，创建了一个名为 code 的索引，用于存储所有语言的开源代码文件。值得注意的是在 Elasticsearch 中索引是一个逻辑空间的概念，这和分片（shard）的概念相互区分。

分片（Shard）和副本（Replica）：为了实现分布式存储和提高系统的可用性与性能，Elasticsearch 引入了分片和副本的概念。分片是将一个大的索引拆分成多个较小的部分，每个分片可以独立存储在不同的节点上，这样可以并行处理搜索请求，提高搜索效率。副本是分片的复制，每个分片可以有多个副本，当某个节点出现故障时，副本可以接替其工作，保证系统的正常运行。在代码检索系统中，对于数百万个开源项目文件的索引，可以通过合理设置分片和副本的数量，来优化系统的性能和可靠性。

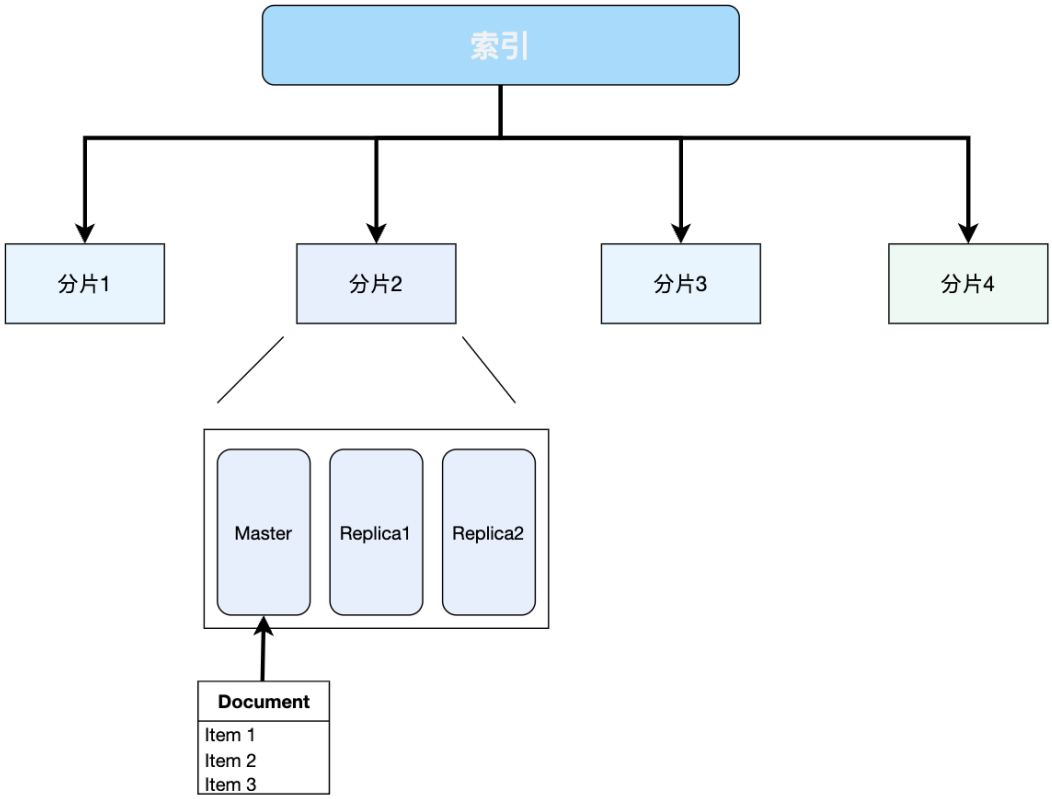

图 2.7　Elasticsearch 结构

文档（Document）：文档是 Elasticsearch 中可被索引的最小数据单元，它以 JSON 格式表示，类似于关系型数据库中的行。在代码检索系统里，一个文档可以对应一个具体的代码文件，包含文件名、代码内容、开源协议、星数等信息。

2.4 Visual Studio Code 插件

为了高效构建美观且用户友好的交互界面，Visual Studio Code 通过支持 TypeScript 或 JavaScript 实现其官方接口，显著简化了开发流程。此外，VS Code 允许在侧边栏（Side Bar）中嵌入 WebView 页面，这一特性为开发者提供了利用前端框架开发插件的可行性，极大降低了技术学习门槛。在持续开发与维护前端项目时，选择成熟框架已成为关键策略。Vue.js 作为主流前端框架，已迭代至 Vue 3 版本。其庞大的开源生态（如 Element Plus 等 UI 组件库）为开发者提供了丰富的可视化资源，即使缺乏专业设计支持，也能快速构建出美观的界面。相较于前代版本，Vue 3 在响应式系统、组件化开发及性能优化等方面引入了多项革新性改进。本章节将系统性地介绍 Vue.js 框架的核心特性，

2.4.1 Vue 框架

Vue.js 是一套渐进式 JavaScript 框架，凭借其轻量级架构、低学习门槛、高效性能等核心优势，已成为全球范围内最受欢迎的前端框架之一。作为渐进式框架的典型代表，Vue.js 的设计以灵活性为核心原则，开发者可根据实际需求逐步采用其提供的功能模块，而非强制要求一次性全面使用所有特性。这种渐进式特性使其既能满足小型项目的快速开发需求，又能支撑大型应用的复杂架构。

在设计模式方面，Vue 采用模型-视图-视图模型（Model-View-ViewModel，MVVM）架构（图2.8所示）。相较于传统的 MVC 模式，Vue 的 MVVM 模式实现了视图与模型的解耦，并通过视图模型（ViewModel）层实现数据的无缝双向绑定。该模式通过数据绑定和视图自动更新机制，显著减少了冗余代码的编写，使开发者能够将精力集中于核心业务逻辑与界面设计的优化。

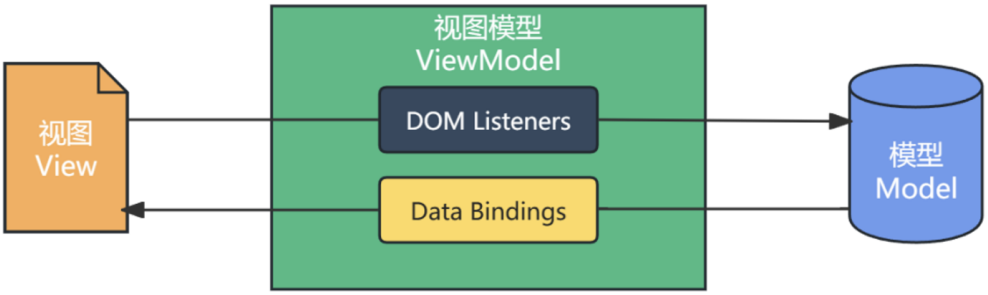

图 2.8　Vue 的 MVVM 设计模式

在技术特性方面，Vue.js 引入了虚拟 DOM（Virtual DOM）与响应式数据系统。其中，响应式系统通过实时追踪状态变化，能在数据更新时智能驱动视图的局部渲染，从而大幅减少直接操作 DOM 的频率。这一机制不仅降低了开发复杂度，更避免了频繁调用 DOM 对象的性能损耗，使开发者得以专注于业务逻辑的实现。

2.5 本章小结

本章介绍了实现面向多编程语言的代码检索系统的核心理论和技术。首先探讨了大语言模型及其核心架构 Transformer。Transformer 通过多头注意力机制、位置前馈网络、残差连接与归一化等模块，解决了传统循环神经网络在并行计算和处理超大规模文本数据时的局限性，成为自然语言处理领域的首选架构。此外，混合专家模型（MoE）通过稀疏门控机制有效提升了模型的计算效率和容量，成为突破单一模型规模瓶颈的重要技术路径。另外强调了通过精心设计提示模板来增强和改善大语言模型能力的重要性。提示工程不仅能够提升模型在各类任务中的表现，还能催生全新的交互模式，充分挖掘大语言模型在文本与代码生成之外的潜力。随后探讨了 Elasticsearch 在多编程语言代码检索系统中的应用。Elasticsearch 作为一个开源的分布式搜索和分析引擎，凭借其高效的全文搜索能力和分布式架构，能够快速、准确地检索海量数据，提供便捷的代码检索服务。最后，介绍了 Visual Studio Code 插件的开发，特别是 Vue.js 框架的应用。Vue.js 以其轻量级架构和渐进式特性，结合虚拟 DOM 和响应式数据系统，为开发者提供了高效的前端开发体验。通过这些技术的结合，本系统能够实现高效、准确的代码搜索和用户友好的交互界面。

3 系统需求分析与设计

本章主要介绍面向多编程语言的代码检索系统的系统需求分析与设计工作，包括系统需求分析、系统架构设计、业务功能设计和数据库设计。首先从用户需求和系统的功能性与非功能性需求出发，对面向多编程语言的代码检索系统进行深入的需求分析。接着，根据需求分析的结果，对系统进行详细的解析，设计总体架构和前后端服务架构。然后，进行系统的业务功能设计，重点设计项目存储、大模型提示词工程等关键功能模块。

3.1 系统需求分析

3.1.1 用户需求

本系统的核心目标是为开发人员提供高效便捷的代码检索支持，通过智能化的自然语言搜索能力辅助开发流程，显著提升代码复用效率与开发质量。特别针对涉密场景需求，系统支持完全离线化部署方案，可在无网络环境下实现对本地化代码仓库的检索功能。系统主要面向两类核心用户群体：开发人员与系统运维人员。整体用户需求示意图如图3.9所示。

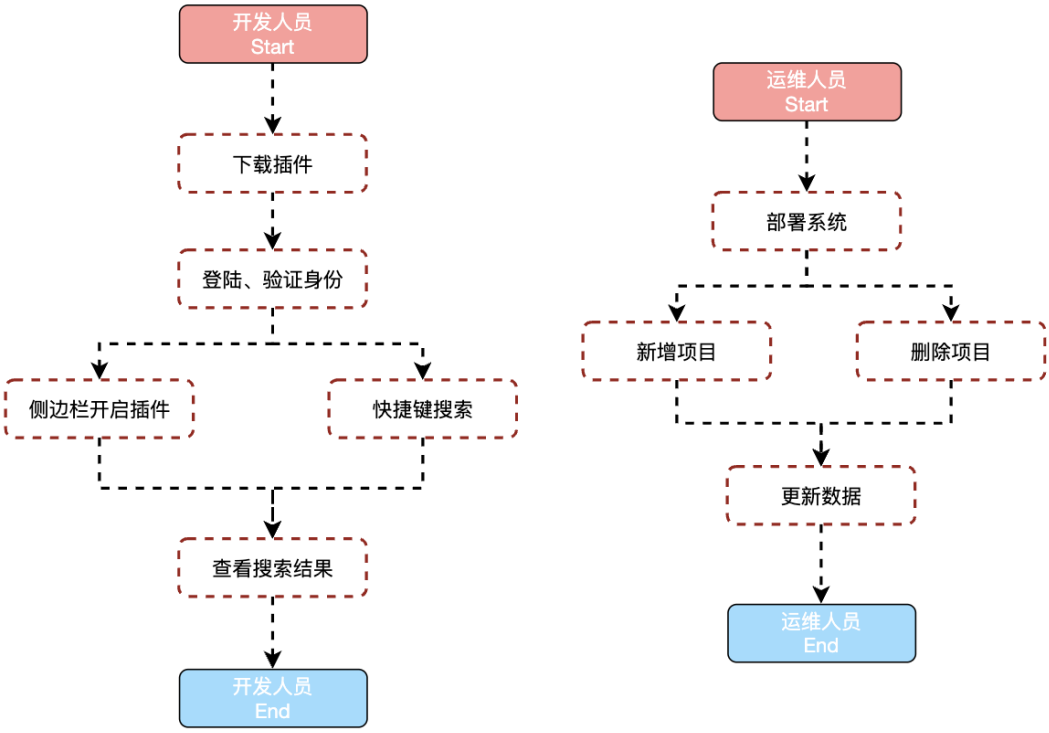

图 3.9　整体用户需求示意图用例图

3.1.2 系统功能性需求分析

本节将根据用户需求，明确系统核心功能性需求，本小节根据系统不同角色定位，做如下功能性需求分析：

对于开发者用户，做如下功能分析，图3.10为开发人员的功能需求用例图：

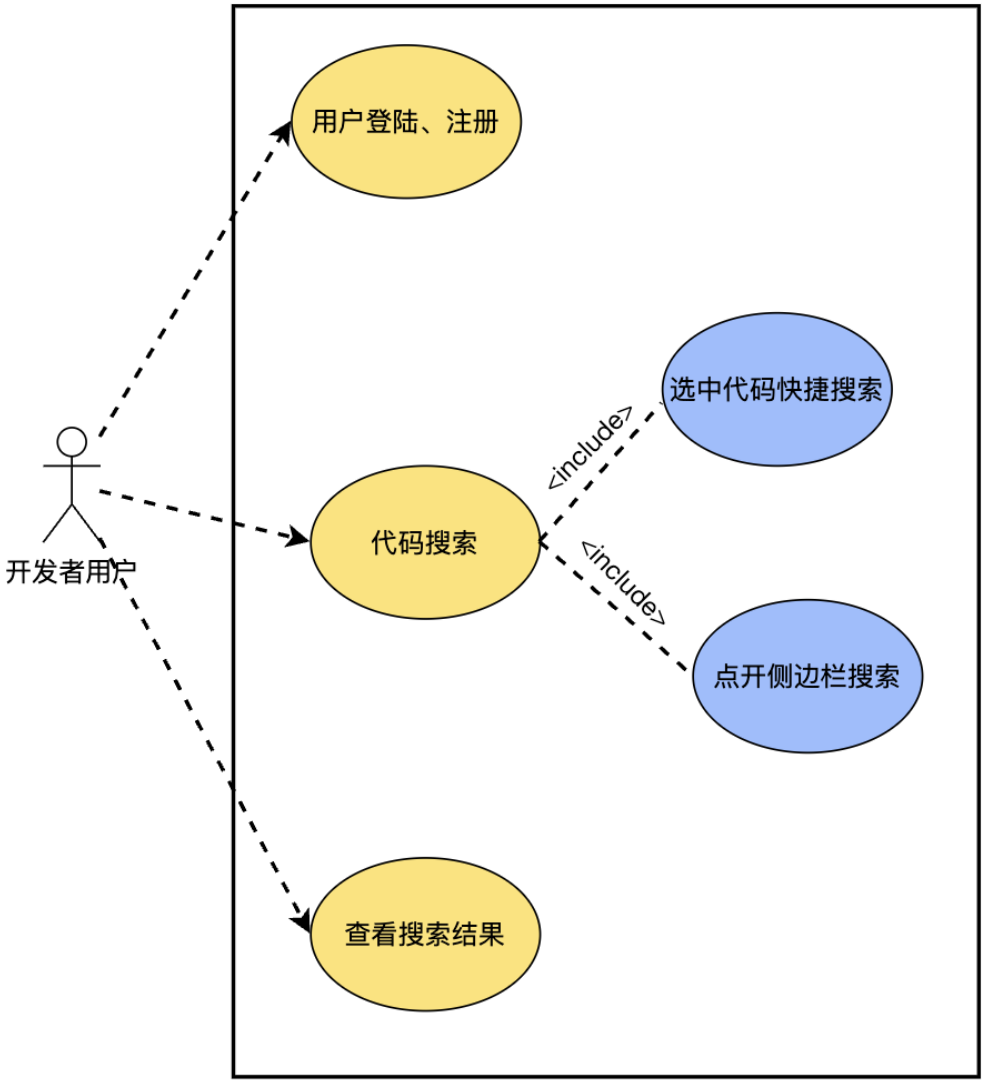

图 3.10　开发者用户用例图

（1）开发者用户注册：开发者用户下载系统后，由系统弹窗提示用户在未登录状态，此时用户可以选择登陆或者注册。如果用户选择注册，则跳转注册页面，用户输入邮箱号后，通过验证码自动完成注册。

（2）开发者用户登陆：开发者用户下载系统后，由系统弹窗提示用户在未登录状态，此时用户可以选择登陆或者注册。如果用户选择登陆，则跳转登陆页面，用户输入绑定邮箱后，经过验证，成功完成登陆。

（3）代码搜索：开发者用户在使用 VS Code 编码过程中，可以通过绑定的快捷

键，直接跳转搜索界面。如果此时鼠标选中了代码，则自动将代码复制到用户的系统粘贴面板上；用户也可以点击侧边的插件 icon，跳转到搜索界面，手动输入搜索关键词开始搜搜。

（4）查看搜索结果，用户在搜索完成后，如果有搜索结果，可点击搜索结果框，通过链接自动跳转项目详情。

对于运维人员，做如下功能分析，图3.11为运维人员的功能需求用例图：

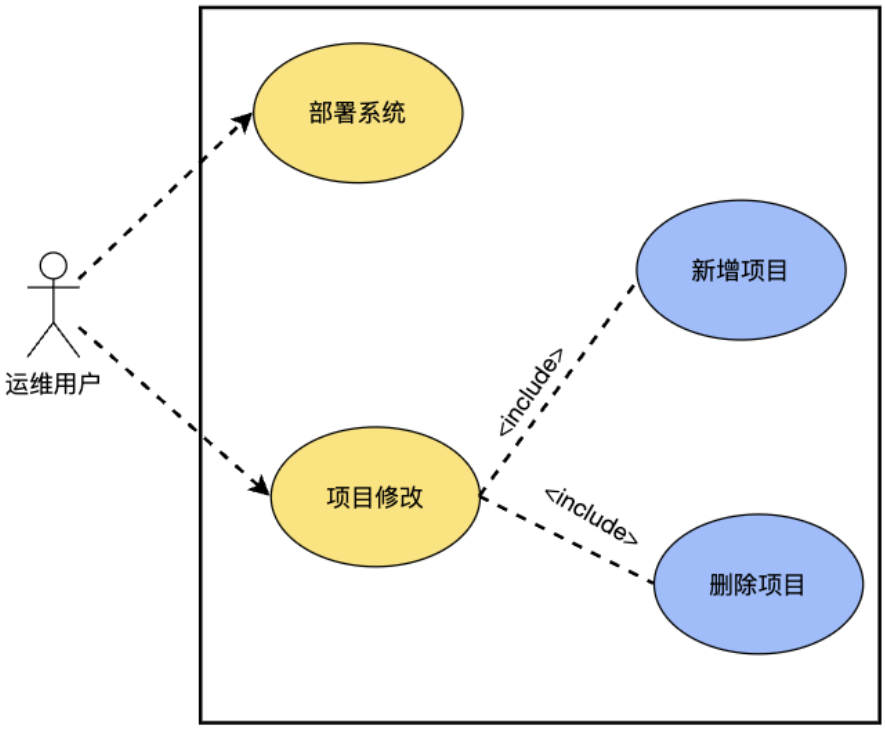

图 3.11　运维用户用例图

（1）系统部署：运维人员能根据系统提供的指示，安装系统必要的依赖，完成系统的部署、迁移。

（2）项目修改：运维人员能根据项目变动，灵活新增或者删除 Elasticsearch 中管理的项目。

3.1.3 系统非功能性需求分析

(1) 易用性：本系统在界面布局上应当符合使用者的实际习惯与需求，应具备逻辑性和直观性，用户在进行操作时应该能够清晰地了解每个功能的用途和位置，操作流程和反馈信息也应该及时、明确。界面应该呈现出清晰、统一的信息组织结构，将不同功能和信息分类展示，让用户能够快速获取到有用的项目信息

(2) 拓展性：本系统应该满足可拓展需求。在前端表现层，能够快速拓展对后续需求页面，对已使用组件做到有效复用。在后端应用层，能够满足后端需求灵活变更，对新增需求和更变需求，通过利用已有功能模板快速开发。

(3) 可靠性：本系统的可靠性需求，主要体现在用户登录、项目搜素，大模型改写方面。具体来说，系统需要保障可靠的用户登录体验；实现可靠的搜索体验；大语言模型在改写用户提示词的时候应该确保不随意篡改用户的本意，达到提升搜索准确性等效果。

(4) 安全性：本系统在安全性上，做到用户间严格分离，不可跨越权限操作。需要设计具备数据备份、恢复功能的数据库处理方法，保障数据库信息安全。

3.2 系统架构设计

3.2.1 系统总体架构设计

面向多编程语言的代码检索系统整体将采用前后端分离的方式进行组织。可以分为前端交互层、后端逻辑层以及数据层三个维度，如图3.12所示。前端交互层使用 Vue 框架编写交互逻辑，使用 ElementUI 渲染前端图形化页面和可视化模块。后端使用 Gin 框架编写应用服务，规范 API 风格，接受前端请求，同时负责和数据层和大语言模型进行交互，实现主要的搜索逻辑。数据层使用 Elasticsearch 管理所有的项目数据。

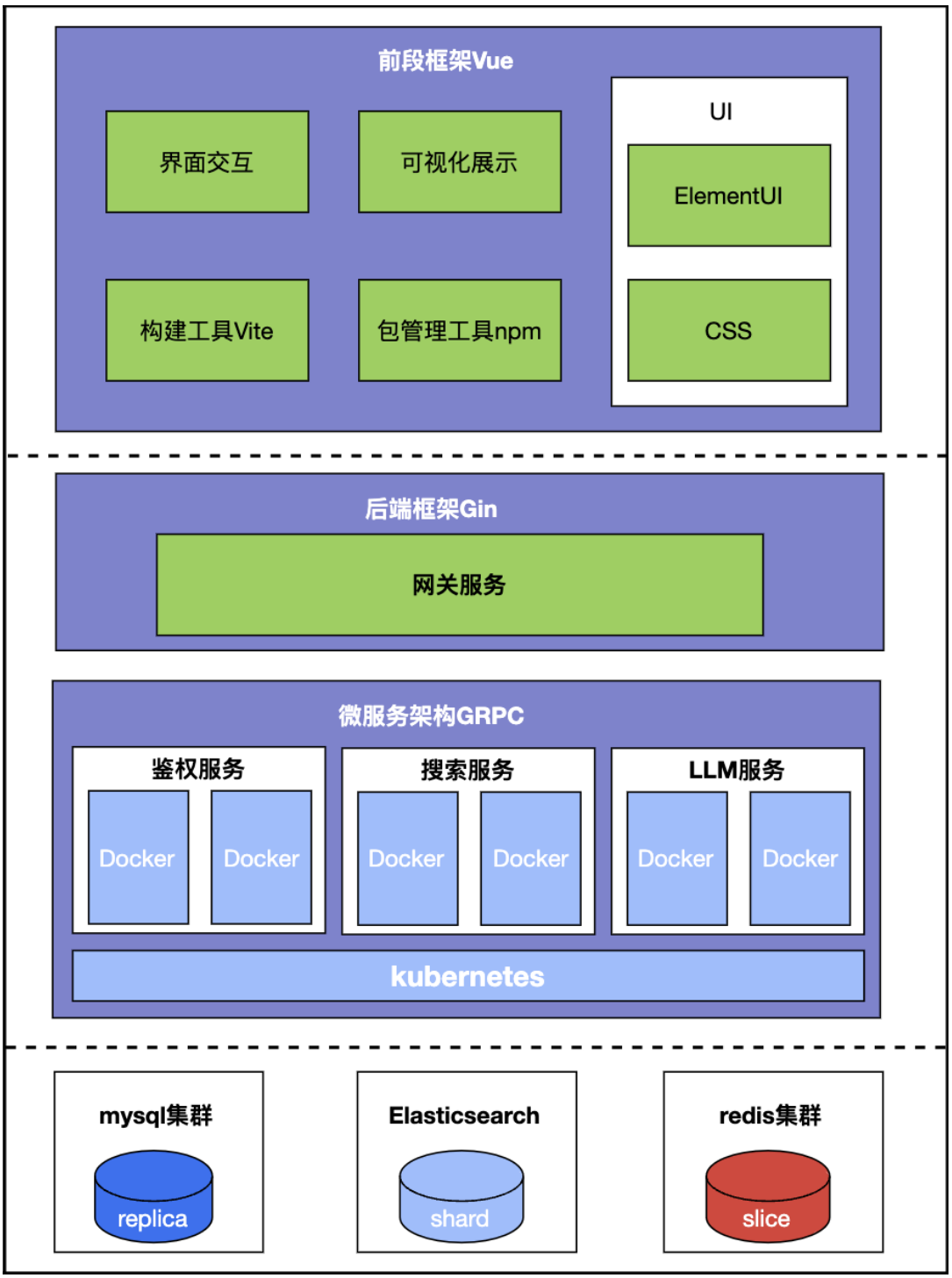

图 3.12　系统总体架构图

3.2.2 前端架构设计

前端沿用 Vue 框架的 MVVM 设计模式，将前端划分为：视图层（View）、视图模型层（ViewModel）、模型层（Model），前端架构设计如图3.13所示。

(1) 视图层涉及用户交互界面，包括开发人员的账户注册和登陆页面，项目代码搜索页面，搜索结果展示页面等。

(2) 视图模型层涉及到前端的业务逻辑处理，负责调度视图层和模型层，涉及到前端中账户登录注册逻辑、项目搜索逻辑、项目详情展示逻辑等。

(3) 模型层涉及到前端的临时数据存储，对来自视图模型层的相关数据进行缓存，涉及到对视图层呈现数据的监听和对应变化处理。

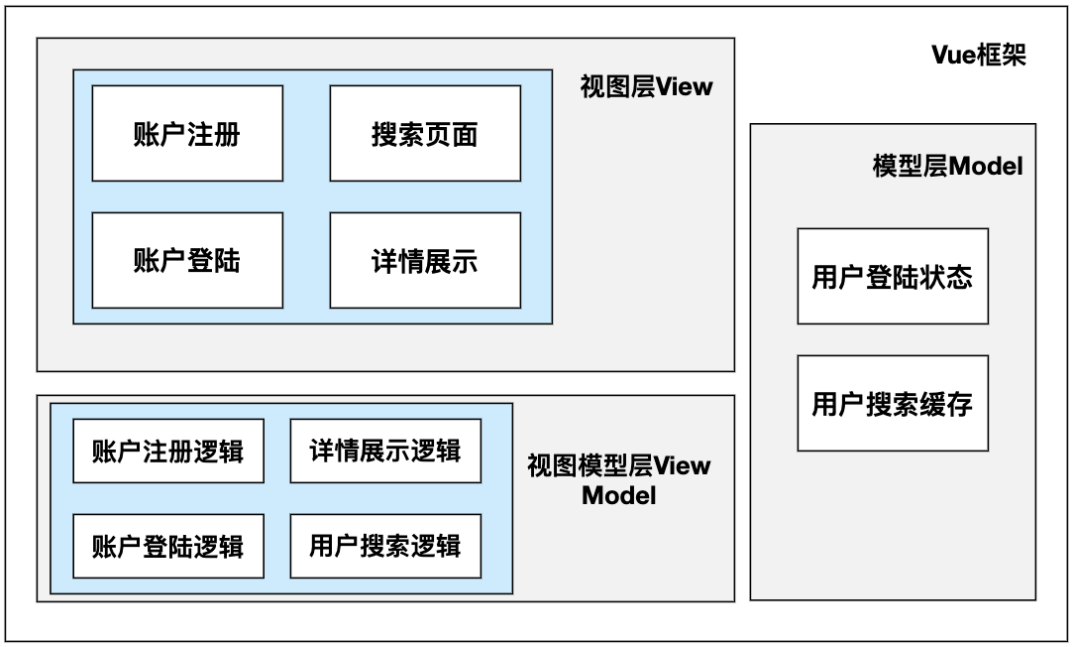

图 3.13　前端架构图

3.2.3 后端架构设计

系统后端采用微服务架构方式，分为网关服务、鉴权服务、搜索服务和 LLM 服务四大核心服务，如图3.14所示。

（1）网关服务提供 http 服务接口，所有的请求全部通过网关进行统一路由，网管服务通过鉴权服务确保所有的请求都是安全，可控的。

（2）鉴权服务，对请求中的 token 进行鉴权，返回网关鉴权的结果。同时对用户的注册和登陆请求进行处理。

（3）搜索服务，负责代码搜索的核心逻辑，与数据层进行交互，同时 LLM 服务进行协作，处理用户的搜索请求。

（4）LLM 服务，本项目的 AI 功能由 LLM 服务提供支持，和搜索服务进行交互，完成对用户搜索词的改写，达到更好的搜索效果。

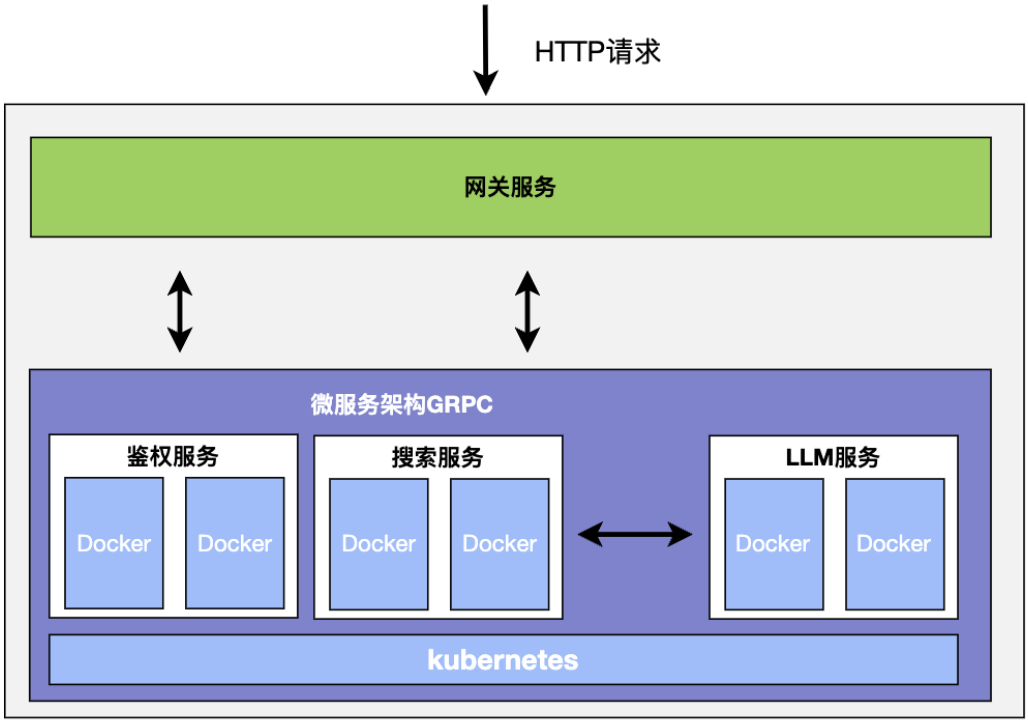

图 3.14　后端架构图

3.3 系统业务功能设计

根据系统需求分析结果以及系统前后端架构设计，将整体系统划分为用户注册、用户登陆、用户搜索、搜索详情四个主要板块。其中用户登陆、注册非本次系统的重点功能，因此本小节将详细介绍展示搜索板块的功能设计和时序图分析。

$$\frac{1}{\mu}\nabla^2 A - j\omega\sigma A - \nabla(\frac{1}{\mu}) \times (\nabla \times A) + J_0 = 0 \tag{6.1}$$

表 6.1 高频感应加热的基本参数

感应频率（KHz）	感应发生器功率(%×80Kw)	工件移动速度(mm/min)	感应圈与零件间隙(mm)
250	88	5900	1.65
250	88	5900	1.65
250	88	5900	1.65
250	88	5900	1.65

续表 3.1

感应频率（KHz）	感应发生器功率(%×80Kw)	工件移动速度(mm/min)	感应圈与零件间隙(mm)
250	88	5900	1.65
250	88	5900	1.65

6.2 本章小结

本章介绍了……

7 结论与展望

7.1 主要结论

本文主要……

7.2 研究展望

更深入的研究……

参考文献

[1] 杨瑞林, 李力军. 新型低合金高强韧性耐磨钢的研究 [J]. 钢铁. 1999(7): 41-45.

[2] 于潇, 刘义, 柴跃廷, 等. 互联网药品可信交易环境中主体资质审核备案模式 [J]. 清华大学学报 (自然科学版), 2012, 52(11): 1518-1523.

[3] Schinstock D.E., Cuttino J.F. Real time kinematic solutions of a non-contacting, three dimensional metrology frame[J]. Precision Engineering. 2000, 24(1): 70-76.

[4] 温诗铸. 摩擦学原理 [M]. 北京: 清华大学出版社, 1990: 296-300.

[5] 蒋有绪, 郭泉水, 马娟, 等. 中国森林群落分类及其群落学特征 [M]. 北京: 科学出版社, 1998: 5-17.

[6] 贾名字. 工程硕士论文撰写规范 [D]. 重庆: 重庆大学, 2000: 177-178.

[7] 张凯军. 轨道火车及高速轨道火车紧急安全制动辅助装置: 201220158825.2[P]. 2012-04-05.

[8] 全国信息与文献标准化技术委员会. 文献著录: 第 4 部分非书资料: GB/T 3792.4-2009[S]. 北京: 中国标准出版社, 2010: 3.

附录 A：XX 公式的推导

XX 公式的推导过程是：

致　谢

致谢主要感谢导师和对论文工作有直接贡献和帮助的人士和单位。致谢言语应谦虚诚恳，实事求是。

原创性声明

郑重声明：所呈交的论文（设计）《_____》，是本人在导师的指导下，独立进行研究取得的成果。除论文（设计）中已经标注引用的内容外，本论文（设计）不包含其他人或集体已经发表或撰写过的作品成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果，并承诺因本声明而产生的法律结果由本人承担。

论文（设计）作者签名: _____

日期：_____

使用授权书

本论文（设计）作者完全了解学校有关保留、使用论文（设计）的规定，同意学校保留并向国家有关部门或机构送交论文（设计）复印件和电子版，允许论文（设计）被查阅和借阅。本人授权重庆大学将本论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制方式保存和汇编本论文（设计）。

本论文（设计）属于：

保　密 □　在____ 年解密后适用本授权书

不保密 □

论文（设计）作者签名：_____　指导教师签名：_____

日期：_____　日期：_____