南京大學

研 究 生 毕 业 论 文

（申请硕士专业学位）

论 文 题 目 基于结构嵌入分析的智能代码推荐系统的设计与实现

作 者 姓 名 韩奇

专 业 名 称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授

2020 年 5 月 23 日

学　　号 ：**MF1832050**

论文答辩日期 ：**2020 年 5 月 23 日**

指 导 教 师 ：　　**（签字）**

The Design and Implementation of Intelligent Code Recommendation System Based on Structure Embedding Analysis

By

Qi Han

Supervised by

Professor **Zhenyu Chen**

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2020

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于结构嵌入分析的智能代码推荐系统的设计与实现

工程硕士（软件工程领域） 专业 2018 级硕士生姓名： 韩奇

指导教师（姓名、职称）： 陈振宇 教授

摘 要

作为开发人员，尤其是编程初学者，在进行日常的软件开发时，经常需要完成一些不熟悉的编程任务。这时，他们可能通过搜索参考已存在代码片段或者复用之前的代码，来帮助完成这个不熟悉的编程任务。近年来，业界已有很多外部代码搜索工具来帮助开发人员。但是使用这些工具会增加开发者搜索筛选的时间，降低编码效率，所以开发出一款与编程一体的代码推荐系统迫在眉睫。

本文依托于慕测 WebIDE 系统进行更新与优化，实现了基于结构嵌入分析的智能代码推荐系统。用户进行编程练习或考试时，通过浏览器访问系统，使用智能编辑器进行在线编程，系统会监听用户编程行为，抓取代码信息进行分析，根据分析结果向用户推荐代码片段。代码推荐依赖大量源代码数据，因此首先需要基于大量的源代码构建语料库，并对其进行预处理。语料库中代码需要先进行词法和语法分析，构建解析树，获得令牌序列以代表代码结构特征。其次，对语料库中的所有令牌序列集，使用局部敏感哈希进行分类，减少比较次数，提高效率。接着，对相似度高的这些候选代码片段通过词嵌入训练模型，再进行句嵌入获得向量表示。最后，通过相似性分析算法对句向量进行比较，向用户推荐相似度值高于阈值且不同的前几位源代码片段，帮助用户根据现有源代码获得有价值的推荐，从而提高生产率，帮助缩短开发时间。

本系统根据主要功能和特性划分为五个模块，包括编辑器模块、语言服务协议模块、代码搜索模块、特征提取模块和特征序列分析模块。系统使用 MonacoEditor 作为主要编辑器基础，并遵循语言服务协议实现 Java、Python 等编程语言的实时代码补全、悬浮提示、跳转定义、引用查找、代码诊断等智能功能。为保障数据的可靠性和高可用性，系统使用搜索引擎进行数据采集处理存储。

目前该系统已替换原有慕测 WebIDE 系统上线使用。通过实验表明，系统提升了平均 82.72% 的用户的编程体验。智能代码推荐系统在用户开发时，有效地推荐代码片段帮助用户重用或纠正现有代码。该系统减少用户搜索筛选时间，缩短开发时间，提高用户编码效率。并且其简单易用的界面与交互风格，符合大多数用户习惯，提升和完善用户编程体验。

关键词：在线编辑器，代码推荐，局部敏感哈希，结构嵌入，相似度分析

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Intelligent Code Recommendation System Based on Structure Embedding Analysis

SPECIALIZATION: Software Engineering

POSTGRADUATE: Qi Han

MENTOR: Professor Zhenyu Chen

Abstract

As developers, especially beginners in programming, often need to complete some unfamiliar programming tasks in daily software development. At this time, they may help to complete this unfamiliar programming task by searching for existing code fragments or reusing the previous code. In recent years, there have been many external code search tools in the industry to help developers. However, using these tools will increase the time for developers to search and filter, and reduce the coding efficiency, so it is urgent to develop a code recommendation system that integrates with programming.

This thesis relies on the Mooctest WebIDE system to update and optimize, and implements an intelligent code recommendation system based on structure embedding analysis. When users perform programming exercises or exams, they access the system through a browser and use an intelligent editor for online programming. The system will monitor the user's programming behavior and grab code information for analysis. Recommend code fragments to users based on the analysis results. Code recommendation relies on a large amount of source code data, so it is first necessary to build a corpus based on a large amount of source code and preprocess it. The code in the corpus needs to perform lexical and grammatical analysis first, construct a parse tree, and obtain a token sequence to represent the structure characteristics of the code. Secondly, use local sensitive hash to classify all token sequence sets in the corpus to reduce the number of comparisons and improve efficiency. Then, the candidate code fragments with high similarity are trained by word embedding, and then sentence embedding is performed to obtain a vector representation. Finally, the similarity analysis algorithm compares the sentence vectors and recommends to the user a similarity value above the threshold and different first few source code fragments to help the user obtain valuable

recommendations based on the existing source code, thereby improving productivity and helping Reduce development time.

According to the main functions and characteristics, the system is divided into five modules, including editor module, language service agreement module, code search module, feature extraction module, and feature sequence analysis module. The system uses MonacoEditor as the main editor base, and follows the language service agreement to implement real-time code completion, floating prompts, jump definitions, reference search, code diagnosis and other intelligent functions in Java, Python and other programming languages. To ensure data reliability and high availability, the system uses a search engine for data collection, processing and storage.

At present, the system has replaced the original Mooctest WebIDE system for online use. Experiments show that the system improves the programming experience of 82.72% users on average. The intelligent code recommendation system effectively recommends code fragments to help users reuse or correct existing codes when they are developed. The system reduces user search time, shortens development time, and improves user coding efficiency. And its simple and easy-to-use interface and interactive style, in line with most user habits, improve and improve the user programming experience.

Keywords: WebIDE, Code Recommendation, Locality Sensitive Hashing, Structure Embedding, Similarity Analysis

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景与意义

近些年来，随着互联网的迅速发展和国家及人民对教育的重视程度的提高，网络教育成为一种热门趋势。网络教育即利用互联网、多媒体等现代化信息技术展开在线教育的模式。它克服了传统线下教育模式受时间、空间、环境等各方面因素影响的不足。对于需要进行编程学习的学生和需要进行编程教育的老师来说，一个功能完善、用户友好的在线服务平台是很重要的。

慕测平台是一个为高校教师和学生提供编程类练习和考试的服务平台，该平台对编程练习进行自动化评分和可视化展示。目前平台主要支持 Java 编程、Python 编程、开发者 Junit 测试等服务。

现今，开发人员在进行软件开发的过程中一般都需要使用集成开发环境（简称 IDE），例如 Eclipse[1]、IntelliJ IDEA[2]、Visual Studio Code[3] 等。这些 IDE 一般提供基本的代码分析、编译、调试等功能。除此以外，绝大多数本地 IDE 在代码编写时，还支持自动补全、错误诊断、定义跳转、悬浮提示等智能功能，这类针对语言特性的智能功能帮助用户快速进行开发工作并提升编程体验。随着容器技术的逐步流行以及 Web 应用的快速发展，大规模使用在线集成开发环境（简称 WebIDE）进行编程开发成为趋势。

慕测 WebIDE 系统是一个基于 Web 的支持在线编程的集成开发环境。该平台支持在线编程、软件测试等功能。慕测平台的学生用户在参加编程练习或考试时，可以使用慕测 WebIDE 进行在线编码。这为用户提供了简单易用的在线开发环境，解决本地编辑器安装复杂、配置困难、插件安装和更新等问题，提高用户体验。

但对于开发者而言，现有的 WebIDE 和传统的本地编辑器相比仍有很多不足之处，比如不支持所有编程语言，不支持完整语法特性，不支持丰富智能提示，不支持复杂大型项目等，这都大大降低了用户的编程体验。并且在编程过程中，开发者，特别是编程初学者，经常需要完成的是一些他们并不熟悉的编码任务。这时，用户一般通过复用之前的代码或者搜索参考已存在代码片段，并从

[1] https://www.eclipse.org/

[2] https://www.jetbrains.com/idea/

[3] https://code.visualstudio.com/

大量冗余、复杂的搜索结果中筛选出具有相似结构的代码来快速完成编程任务。这就增加了编程的整体时间开销，使得开发者编体验降低。

本论文旨在设计一个基于结构嵌入分析的智能代码推荐系统，为用户提供简单智能在线编辑环境和精确有效的代码推荐服务，从而减少用户站外搜索和筛选信息的时间消耗。本系统主要面向编程练习和学习用户，提供 Java、Python 代码补全、悬浮提示、跳转定义、引用查找、代码诊断等代码特性推荐功能，并为编程的用户提供了实时的代码片段推荐服务。系统在用户编码过程中进行高频率计算，以分析出一些局部相似性高的代码片段进行推荐。以此来帮助程序员在一定程度上完善其编写的代码片段，或通过增加一些其他代码来解决代码逻辑或使用规范等的错误。本系统通过智能推荐服务指导用户完善代码和完成所需功能，帮助用户进行编程学习，提升用户编程体验，提高用户编程效率。

1.2　国内外研究现状

本系统主要为在线编辑器实现代码特性提示和代码片段推荐等智能服务，所以本节将详细介绍目前在线集成开发环境应用现状和代码推荐服务研究状况。

1.2.1　在线集成开发环境的应用现状

目前有很多公司出于不同的需求都开发了在线集成开发环境，提供的功能也不尽相同。小众的有比如 CodeSandbox[4]、jsFiddle[5]、Ideone[6]、Codechef[7] 等平台都支持代码编译运行功能。

具体来说，CodeSandbox、jsFiddle 都是一个快速进行 web 开发的工具，支持项目管理，但不提供 Java 等主流编程语言的支持。Ideone 是一个在线代码编辑工具，支持代码编译运行和调试，支持包含 Java、Python、C++ 等在内的 60 多种编程语言，以及源代码的编译和在线执行操作。但操作界面设计简易且不够友好，只支持单文件运行，且编辑器只提供语法高亮的功能。Codechef 是主要用于算法、计算机编程和编程竞赛的平台，支持多文件创建和编辑，但不支持项目管理，编辑器仅提供语法高亮和简单的关键字、特定方法和代码片段的提示功能。还有很多仅支持极少数编程语言开发的 WebIDE，特别是欠缺对于主流编程语言如 Java、Python、C++ 等的支持。并且绝大多数编辑器不支持包括自动补全、定义跳转、悬浮提示在内的类本地编辑器的智能提示的功能。

[4]https://codesandbox.io/

[5]https://jsfiddle.net/

[6]https://www.ideone.com/

[7]https://www.codechef.com/ide

要考虑对开发项目的管理支持较好的平台系统，Eclipse Che[8]、AWS Cloud9[9]、Cloud Studio[10] 是目前被提及较多的主流 WebIDE。Eclipse Che 利用项目输入、工作区、模块化扩展插件的结构化支持 Codenvy 的引擎，提供了 Git 管理、工作区代理、协作开发、终端支持等服务，可以看作是 Eclipse 的线上版本。AWS Cloud9 是 Cloud9 被 AWS 收购之后的产物，提供了实时代码调试、终端支持、协作编码等服务。Cloud Studio 是 Coding 在原有的 WebIDE 基础上进行改进的产品，将服务集成于腾讯云，提供了 Git/SVN 代码托管、全功能终端支持、任务管理、开发协作等服务。

总结来说，这三大 WebIDE 都具备项目管理、文件管理、主流编程语言的编译运行等功能，并且编辑器支持本地 IDE 所有的语法高亮、自动补全、定义跳转、悬浮提示等智能功能，但还有改良的空间。

现有的这些 WebIDE 为我们开发、完善慕测 WebIDE 提供了思路与指导。为结合在线教育场景和为用户，尤其是编程初学者提供简单易用的编程练习平台，系统首先实现了对软件项目级别的支持、对文件的管理、代码编译运行、以及编辑器在基本语言高亮之外提供代码诊断、自动补全、跳转定义、悬浮提示、引用查找等智能功能。在此基础上，为了提高用户编码效率，减少用户学习路径，增加了方法级别的代码片段推荐、API 文档及示例搜索和代码片段及方法搜索等功能，使得 WebIDE 在开发者编程时更友善、易用，并提高用户的开发效率和学习兴趣。

1.2.2　代码推荐技术的研究现状

目前学术界已提出很多种用于代码补全或推荐的技术，大多数技术主要基于信息检索、模式挖掘、优化搜索、统计模型、深度学习等方法 [1]。

一些技术利用开发人员对需求的自然语言描述作为输入，例如 Gu 等人在 2018 年提出的代码描述嵌入神经模型网络 CODEnn [2]，通过将代码片段和自然语言描述语句同时嵌入高维向量空间，以使代码段及其对应的描述具有相似的向量，依据它们的获得的矢量检索与自然语言查询语句相关的代码段。并且此方法能够辨别在语义上有关联的单词，并且可以对查询语句中无关的关键字或噪声进行优化处理。他们在 2016 年提出的 DeepAPI [3]，是一种基于深度学习的方法来生成给定自然语言查询的 API 使用序列，采用了一种称为 RNN 编码器-解码器的神经语言模型。将一个字序列（用户查询）编码成一个固定长度的上下文向量，并基于上下文向量生成一个 API 序列。Raghothaman 等人在 2016 年提出的工具 SWIM [4]，在给定与 API 相关的自然语言查询时，引入结构化调

[8] https://www.eclipse.org/che/

[9] https://aws.amazon.com/cn/cloud9/

[10] https://studio.dev.tencent.com/

用序列来捕获 API 使用模式，推荐代码片段。

在对 API 使用代码推荐的方面的研究，例如基于搜索的 API 使用示例代码推荐有 Stylos 等人在 2006 年提出的用于查找 API 组件和示例的 Web 搜索工具 Mica [5]，它使用 GoogleWebAPI 查找相关页面，然后剖析这些页面的内容和结构以提取关联度最高的编程术语再对每个分析结果的类型进行分类。Wang 等人在 2011 年提出的基于 Web 搜索的 Java API 使用示例推荐系统 APIExample [6]，该工具从网络中收集相关的 web 页面，将嵌入在页面中的 Java 代码片段及其周围的描述性文本提取出来，通过前端标签结构的分析将它们再次组合成使用示例推荐给用户。例如基于用户代码的更改进行的 API 推荐有 Nguyen 等人在 2016 年提出的工具 APIREC [7]，通过对细粒度代码更改及这些更改所在的上下文进行统计学习从而实现推荐。例如基于代码结构的图模型，Nguyen 等人在 2012 年提出的一种基于图形的、面向模式的、上下文敏感的代码补全方法 GraPacc [8]，通过基于图形的模型来管理和表示多个变量，方法和控件结构的 API 使用模式。但它只能推荐 API 使用模式，且上下文匹配能力较弱。以及 Nguyen 在 2015 年提出的一种基于图的统计语言模型 GraLan [9]，该模型从源代码语料库中学习，并根据给定的观察（子）图来计算任何图的出现概率。但它仅支持对通用 API 调用和控制单元的预测，并且合成能力较弱。

总的来说，有利用 API 的使用模式 [8–13]，通过诸如频繁项集挖掘、成对关联、频繁子序列或子图挖掘等确定性算法进行挖掘。当需要推荐时，这些方法会分析上下文，如果上下文与先前标识的模式匹配，推荐器将建议模式中的其余 API 元素。这些工具对于挖掘的模式很有效，但是，它们不能推荐挖掘模式之外的任何代码，挖掘模式的数量也通常限制在几百个。还有基于统计模型进行推荐的技术，方法 [9, 14–19] 通过语言模型使用统计学习来推荐下一个令牌，包括 API 调用。它们的原理依赖于源代码的规则性 [15]，一个模型可以从一个大型语料库中统计地学习代码模式。然后，它可以预测哪个标记可能跟随给定的代码元素序列。该方法的一个关键限制是，很难确定哪些令牌属于特定于项目的代码习惯用法，从而产生用于推荐的噪声。

在代码到代码的搜索推荐方面 [14, 20]，可以使用部分代码片段作为查询从语料库中检索相关代码片段。然而，这种代码到代码的搜索工具返回大量相关的代码片段，而不删除或聚合类似的代码片段，对于用户体验并不好。

除此以外，代码克隆检测器 [21–24] 是另一组可能用于检索推荐代码片段的技术。但是，代码克隆检测工具通常检索与查询片段几乎相同的代码片段。此类检索到的代码段可能并不总是包含可用于扩展查询段的额外代码。

可以发现，研究使用了各种方法，不同侧重点对代码推荐进行了探索和尝

试，不过不同技术点都有一些不足之处。本系统考虑到用户在线编辑环境的易用性和健壮性，提供用户主动代码片段和方法搜索和代码片段推荐及系统自动触发代码片段推荐的能力，推荐模型会对用户代码进行不断地学习以丰富语料库内容，提高推荐准确性，并且希望在一定程度上保证快速的反馈和响应，因此考虑更简易、轻量的实现方案，完成本系统的设计与实现。

1.3　本文主要工作内容

本文通过研究了解当前在线集成开发环境的发展情况，思考如何与教学场景更好的结合，并探讨了开发者在日常编码过程中的行为习惯。研究发现不论是有经验的程序员还是初学者，都需要搜索代码片段来进行重用。然而由于网页上的知识数量庞大且分散，通常需要花费很多时间，才能从网络搜索得到的大量数据中筛选出所需的示例。为了解决传统 WebIDE 不适应教学场景，用户检索代码时间过长等问题，我们提出了一个基于结构嵌入分析的智能代码推荐系统。其主要目标是为开发者，尤其是编程初学者提供一个简单、易用、友好、智能的在线集成开发环境，减少用户站搜索和筛选代码信息的耗时，提高用户开发效率并辅助编程教学，激发学生编程兴趣。

本论文重点讨论基于结构嵌入分析的智能代码推荐系统的设计以及具体实现，并对保障代码推荐功能的及时性和精确性提出解决方案。本系统前端使用 React 框架实现组件化开发，并以 Monaco Editor 为基础，辅助 Language Server Protocol 实现具有代码提示功能的智能化编辑器。后端基于 Spring Boot 框架开发，代码推荐服务的核心业务逻辑使用 Python 完成算法分析，作为独立的服务部署对接，保证 WebIDE 基本服务后端的可靠性和性能以及推荐服务的可移植性。为进行代码推荐，通过 Elastic Search 完成对海量语料数据实时快速地进行存储、搜索和管理。代码编译运行都是在独立的 Docker 容器中完成，Docker 内部环境独立于服务器，保证了隔离性和安全性，并通过前端合理的交互设计以及统一简洁的视觉设计，提高用户对于系统的使用体验。

代码片段推荐服务核心技术主要分为以下几个步骤。首先，通过构建抽象语法树并简化以提取代码结构信息。其次，使用局部哈希敏感算法进行海量语料数据近邻过滤，完成初步相似代码分类，获得候选数据集，这一步主要为缩短推荐时间。获得的 LSH 模型将被保存对外提供服务，减少每次模型构建和加载耗时。然后，通过对特征词使用 FastText 技术计算词向量，通过增加 N-gram 保留部分词序信息，考虑代词内部的形态特性，并且对于训练语料库中不存在的单词也能生成词向量，同时在输出使用分层 Softmax 也缩减了训练时间。对于语

料库中所有词进行嵌入获得的 FastText 模型将被保存对外提供服务，减少每次模型构建和加载耗时。接着，使用 SIF 模型进行句嵌入。最后，对之前的候选数据集通过与用户代码的句向量进行余弦相似度算法计算获得句向量相似性，将结果高于既定阈值的前五位相似度值不同的代码片段按照相似度值由高到低推荐给用户。这一步也将语料库中获得的高度重合代码剔除，保证所推荐代码片段的唯一性和精确性。代码推荐帮助用户减少站外搜索和筛选已有代码片段的时间，帮助用户提高开发效率，完善用户编程体验。

1.4　本文的组织结构

本文组织结构如下：

第一章是引言部分。本章阐明了项目的背景和意义，研究了在线集成开发环境的应用和代码推荐技术现状，以及本文为解决已存在问题提出的方案和主要工作内容。

第二章是技术概述。本章通过相关技术的对比，介绍了系统主要选择和使用的核心概念和技术方案。

第三章是系统需求分析与概要设计。本章将先从项目背景、系统开发目的和目标以及系统提供的功能等方面介绍系统整体情况。接着，本章分析了系统的功能和非功能需求，并基于此给出了系统用例图，通过详细的用例描述进行说明。再通过系统架构图和 4+1 视图给出系统的总体架构设计以及模块划分，并对各模块进行简单设计描述，包括架构设计、流程设计和概念类图设计。

第四章是系统详细设计与实现。本章以第三章为前提，对编辑器模块、语言服务协议模块、代码搜索模块、特征提取模块、特征序列模块进行了详细设计，绘制了顺序图和类图，并展示了关键代码片段。

第五章是系统测试与分析。本章将对系统测试环境进行说明，描述功能和性能测试过程以及最终结果，并就结果进行分析总结。

第六章是总结与展望。本章将总结项目开展过程和成果，并对系统的优点和存在的不足加以说明，并对项目的未来发展做出进一步的规划。

第二章　相关技术概述

本章对系统中使用的相关技术和解决方案以及选择原因进行简要概述。本系统使用对源代码构建抽象语法树，提取相关代码结构信息，使用 FastText 模型计算词向量进行特征提取，提取语义相关性，基于获得的词向量使用 SIF 加权词袋模型进行句嵌入，进一步将解析出的代码片段的特征序列表示为向量，在进行近邻代码片段过滤时，采用 LSH 提高效率。遵循 LSP 协议使得高效实现编辑器智能提示成为可能。

2.1　抽象语法树

2.1.1　抽象语法树简介

抽象语法树（Abstract Syntax Tree）是具有一定语义的源代码结构的树型表示。解析树是一种内部结构，由编译器或解释器在解析某些语言结构时创建。这种解析也被称为“语法分析”，因此这种结构也可被称为语法树。解析树的实际实现主要关注语法，包括空格、关键字、括号等细节。而抽象语法树 [25] 其实是解析树或语法树的一个简单版本，其之所以是抽象的，则是因为其并没有完全包含所有真实语法中的信息，它忽略了一些解析树中不重要的细节，只保留必要的节点，因此同对于一份源代码，抽象语法树更简洁，节点更少。

构建抽象语法树一般有以下三个常见用途：

第一，用作代码提示优化插件或工具。如代码语法检查 [26]，代码风格检查，代码的格式化，代码语法高亮等等之类。

第二，用作代码的混淆压缩工具。如 UglifyJS 等。

第三，用作更改代码结构的工具。如 Webpack 或 Rollup 用于项目构建和模块打包，将 TypeScript 或 JSX 等编译为原生 JavaScript。

2.1.2　抽象语法树描述

语法树生成流程见图 2.1。主要包含以下几个步骤：

第一步，词法分析。词法分析器会按照预定的规则将源文件中字符串形式的代码分割成一个个令牌（Token）并会移除空白符等信息，形成令牌（Tokens）序列。这里的单词一般包含关键字、标识符、界符、数字等。当在词法分析阶段时，分析器会逐词读取源代码，进行扫描。扫描过程中的每一个令牌会在遇到空格，操作符等特殊符号时结束。

第二步，语法分析。语法分析过程会依照特定的语法规则指导，将词法分析得到的令牌序列匹配组合成有用某种含义的语法短语并验证这些语法短语是否符合语法规则，这些语法短语最终会被转化成树状表达形式。

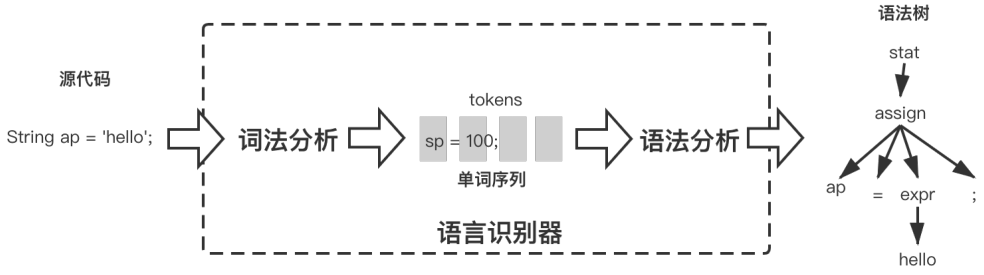

图 2.1: 语法树生成流程

2.1.3　抽象语法树在本系统中的应用

在本系统代码推荐部分，将使用 ANTLR 4 作为解析器生成工具对源代码进行分析。

ANTLR（ANother Tool for Language Recognition）是一种强大的公共域解析器生成器 [27]，结合了手工编码解析的灵活性和解析器生成器的便利性，用于对结构化文本或二进制文件如程序源代码、数据和配置文件进行读取、处理、执行或翻译。它广泛用于构建语言、工具和框架，如遗留代码转换器、特定于域的语言解释器、Java 到 C# 的转换器、wiki 呈现器、XML/HTML 解析器、DNA 模式识别器、字节码汇编器、语言格式美化，以及完整的编译器。

ANTLR 从一个称为语法的正式语言描述中生成一个程序（解析器），该程序（解析器）读取符合该格式的文件，并构建一个表示输入的数据结构（一个解析树）。

通过书写符合规范的文法规则和词法规则文件，可以使用 ANTLR 工具生成 Lexer 词法解析器和 Parser 语法解析器。官方提供了多种编程语言的文法规则和词法规则，便于使用，也可根据个人需求修改。本系统使用 ANTLR 4 构建解析树的原因如下：

第一，学习难度低。ANTLR 的 V4 版本在很大程度上简化了用于匹配语法结构的语法规则以一种更接近自然语言的形式解析语言。

第二，用户友好。当 ANTLR V4 将语法转换成为可执行文件、可读分析的

代码时，不会发生比如 ANTLR V3 中的模糊警告或 YACC 中的语法冲突等的状况影响用户判断。

第三，全新的解析技术，更高效。ANTLR V4 提出 Adaptive LL(*) 算法，在运行时进行语法分析。解析器会像 Java 使用 JIT 编译器那样升温，代码运行的时间越长，解析速度就越快。其优点是自适应算法比 v3 中的静态 LL(*) 语法分析算法强得多，不再需要静态分析语法，在运行时分析，也因此不需要考虑各种语法的可能性浪费时间。

2.2　FastText 技术介绍

词嵌入（Word Embedding）是将文本中的词转化为数字向量的一种方式，为了使用标准的机器学习算法来分析它们，需要将这些向量转化为数字作为输入。词嵌入的过程是将一个所有词数量维度的高维空间嵌入到另一个连续的向量空间中，其维数要低得多，主要将每个单词或短语映射到实数域中的一个向量，词嵌入的结果就生成词向量。

本系统中，为了实现相似代码片段的推荐需要计算代码片段之间的相似度。为了计算代码相似度，需要先获得代码片段的特征序列，然后根据特征值进行操作。因此如何获得及表示代码的特征向量是很重要的，这将极大影响相似度计算效果。FastText 模型提出字词嵌入方法考虑单词内部的形态特征，使得其因引入词的形态学信息在语法类比任务上效果较好。并且因为在低频词语上表现的相当好，准确率高且能构建训练词库以外的单词，资源消耗少等原因，本系统代码推荐部分使用 FastText 训练词向量。

2.2.1　词向量介绍

关于词向量的表示，目前主要有 One-hot 编码和分布式表示两种方法。

第一种是 One-hot 编码。这种方法是最基本的向量表示方法。One-hot 编码的思想是将要编码的对象去重后表示为二进制向量，并且在表示一个词时在整个向量上只有该词位置有效。向量长度为词汇大小，向量各位上除了该词的项为 1 外，其他位都为 0。例如，对于 n 个词汇量，排在第一位的是“a”，则“a”的向量表示为 n 维向量，仅有第一位为 1，其他位为 0，如下。

第 1 个单词：”a” 向量表示为 [100...000]

第 2 个单词：”an” 向量表示为 [010...000]

第 n 个单词：”zero” 向量表示为 [000...001]

也就是说，One-hot 编码的每个单词都是一个维度，彼此独立。这种方式非常方便，但假如单词之间存在着某种连续性的关联将无法体现，无法表现出词之间的相似性。

第二种是分布式表示。分布式表示（Distributed representation）通过找到一个变换函数使得每个词转化成一个向量，向量的每一个维度不再是只有 0 或者 1，而是通过连续的实数来体现各词之间不同的程度和上下文的关联性。这可以在一定程度上弥补 One-hot 表示的缺陷，向量之间的关系可以反映出单词之间的语义相关性。这基于分布式假说即相似的词会出现在相似的上下文环境下。这就表示词向量需要学习足够多的上下文信息来捕捉和了解一个词的语义信息，该词出现的频率也需要足够的高。其次维度过高或过低都会使得精度降低，过低的维度无法捕捉词汇间的不同意义，而过高的维度则会捕捉一些不利于泛化的噪声，即高方差问题。

分布式表示又包含了以下三种处理方法：

基于矩阵的分布表示。矩阵的每一行表示一个词，每列表示上下文，从矩阵中取得该词的上下文分布。

基于聚类的分布表示。通过聚类算法构造词的上下文关系，例如布朗聚类。

基于神经网络的分布表示。例如 Word2Vec 模型。

2.2.2　FastText 核心思想

对句子进行分类的一个基本方法是将它们表示为词袋（Bag-of-Words, BoW）并训练一个线性分类器。在机器学习范畴，分类的目标是聚集特征相似的对象。为此，线性分类器利用特征的线性组合进行分类决策。一个对象的特征一般使用特征值表示，在向量中的特征一般描述为特征向量。

当数据量很大时，随着类数量不断增大，计算代价也变高，FastText [28] 中使用基于哈夫曼编码树 [29] 的分层 softmax [30] 来优化运行时间，哈夫曼树是外部路径权值最小的完整二叉树，节点的权重以根节点为中心向外递减，使得训过程中的复杂度从 $O(kh)$ 下降到 $O(h\log_2 k)$，进一步加快了搜索速度，其中 k 为类数目，h 为维数。

词袋当中单词的次序是不变的，但是如果需要在计算时考虑词序会使得计算负担变大。FastText 使用 bag of n-grams 作为附加特性来捕获关于局部单词顺序的部分信息，并且在实践中和明确使用单词顺序的方法 [31] 有相当的效果。

Joulin 等人 [28] 在实验中证明了 FastText 训练词向量比传统方法要至少快 2-5 倍，特别是大数据量情况下，会有显著加速表现。

Zhang 等人 [32] 在 2017 年通过对线性模型，FastText 和卷积网络在编码层

面进行比较，得出 FastText 使用字符级 N-gram 编码提供最佳效果的结论，并且对于英文文本，FastText 在使用单词级 N-gram 的处理时效果最好，不过当特征过多时容易出现过拟合。

近年来，使用神经网络技术训练模型变得越来越流行 [33–35]。尽管这些模型在实践中取得了出色的性能，但由于它们在训练和测试时需要花费大量时间，这使得当在海量数据集上使用时受到限制，效果变差。FastText 通过分层 softmax 加快了模型速度，并引入 n-grams 作为附加特性保留了局部的单词顺序信息。

在本文中，为了对代码片段进行特征提取，提取语义相关信息，基于大数据语料库，使用 FastText 对所有方法级别的代码片段分析获得的令牌序列进行词嵌入，作为之后句嵌入的输入，方便后续进行相似性分析。

2.3　加权词袋模型 SIF

平滑倒词频 (Smooth Inverse Frequency，SIF) [36] 由 Arora 等人在 2017 年提出，通过实验发现，在文本相似性任务中，这种加权方法的性能提高了约 10% 到 30%，并且在大多数任务上，甚至优于 Wieting 等人 [37] 测试的包括 RNN [38] 和 LSTM [38, 39] 在内的一些复杂监督方法。

SIF 的计算主要分为两步：第一步，计算句子中所有词向量的加权平均值；第二步，去除平均向量在其第一个主成分或奇异向量上的投影（“公共成分去除”），即使用 PCA 主成分分析或 SVD 奇异值分解训练出移除项，用于降维。此处，一个单词 k 的权重为 $a/(a+p(k))$，其中 a 为参数，$p(k)$ 为计量的单词频率；

SIF 非常适合领域适应性设置，即在各种语料库上训练的词向量用于计算不同实验中的句嵌入。但是当把一个句子看成一个整体，并且如果其中没有重复单词出现的话，SIF 加权就会与信息检索中的 TF-IDF 加权 [40, 41] 非常相似。

SIF 的句模型构建方法相对高效和快捷，通过移出句子的公共信息，突出各句向量之间的差异，因为是一种无监督学习，在大规模语料库的场景下，可以更好的利用，相较有监督的学习性能更好，这是一大优势。由于代码程序的本身的特殊性，每行代码内部都遵循一定顺序并且本文采取的代码片段是方法级别，因此在获得句向量时，即使存在语序颠倒的问题，对相似性判断和推荐的影响较低，因此本系统使用该方法。

2.4　局部敏感哈希 LSH

2.4.1　LSH 基本思想

局部敏感哈希（locality sensitive hashing，LSH）是一种常用的近似近邻 (Approximate Nearest Neighbour, ANN) 搜索算法，一般用于处理在高维空间中海量

数据情况下查找相似节点的问题 [42]，其示意图如图 2.2所示。局部敏感哈希的基本思想是将原数据空间中的相邻的和不相邻的数据点进行哈希，通过同样的转换，映射到另一个新的空间中，原本相邻的数据在新空间中相邻的可能性很大，而原空间中不相邻的数据点，在新空间中相邻的概率很小。

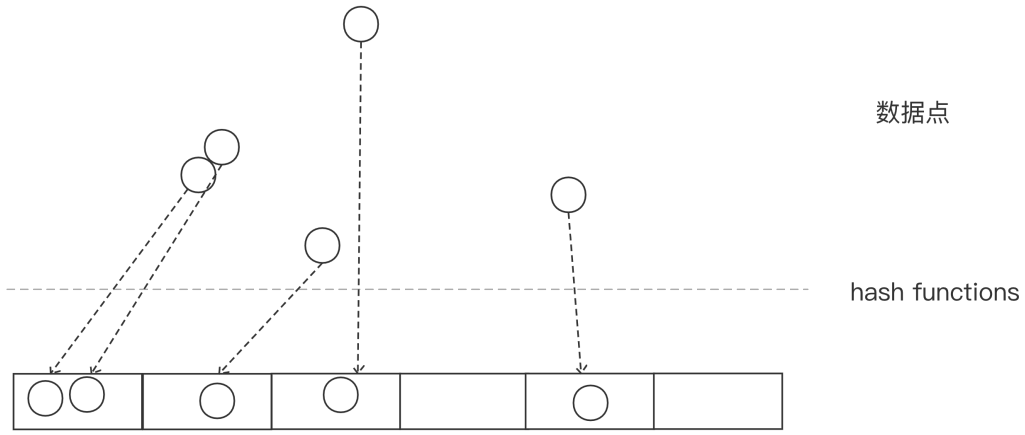

图 2.2: 局部敏感哈希示意图

因此需要一些有具备一定要求的哈希函数用于数据的变换，使得相似的数据可以被归为一类，不相似的数据被分到别的类中。这些哈希函数需要符合以下两个要求，即符合要求的 LSH 函数定义 [43] 为：

1. 如果 $D(x, y) \leq d_1$，则 $h(x) = h(y)$ 的概率至少为 p_1;

2. 如果 $D(x, y) \geq d_2$，则 $h(x) = h(y)$ 的概率至多为 p_2;

其中，$D(x, y)$ 表示 x、y 间的距离度量，h 函数为哈希函数，$h(x)$、$h(y)$ 表示对 x、y 进行哈希转换。

满足以上条件的函数称之是 (d_1,d_2,p_1,p_2)-sensitive 的。

一个或多个 (d_1,d_2,p_1,p_2)-sensitive 的 hash 函数将被用于对原数据点组成的矩阵进行哈希降维获得“签名矩阵（Signature Matrix）”，再通过对“签名矩阵”进行哈希，就可以将所有原数据点进行分类，得到每个数据点组中被划分到哪个类中，与它相似的数据有哪些。

2.4.2　LSH hash 函数-余弦距离

余弦相似度使用两个向量之间的角度的余弦值来衡量它们之间的差异。与距离测量相比，余弦相似度在方向上比在距离或长度上更关注两个向量之间的差异。余弦相似度普遍用于正空间，即取值范围为 0 到 1，用来判别两个向量之间的夹角大小，当向量之间的夹角越小时，代表这两个向量越相似。具体地说，当两个向量夹角为 90 度时，余弦相似度的值为 0，代表完全不相似；当两个向量夹角为 0 度时，余弦相似度的值为 1，代表非常相似甚至一样。

对于给定的两个向量 X、Y，余弦相似度计算公式如 2.1所示：

$$\text{sim} = \cos(\theta) = \frac{X \cdot Y}{||X||||Y||} = \frac{\sum_{i=1}^{n} X_i \times Y_i}{\sqrt{\sum_{i=1}^{n} (X_i)^2} \times \sqrt{\sum_{i=1}^{n} (Y_i)^2}} \tag{2.1}$$

其中，θ 表示 X、Y 向量的夹角，X_i、Y_i 表示 X、Y 向量的各分量。

余弦距离对应的 LSH hash 函数如 2.2所示：

$$H(V) = sign(V * R) \tag{2.2}$$

其中，R 是一个随机向量，$V * R$ 可以认为是 V 在 R 上的投影。其是 (d_1,d_2,(180-d_1)180,(180-d_2)/180)-sensitive 的。

2.4.3　LSH 在本文中的使用

线性搜索 K 最近邻 (K-NearestNeighbor，KNN) 随着数据量的增加，开销也在增大。所以为了在实际工程中应用，一般转化为近似最近邻搜索问题。一种方法是将所有的数据用一个随机投影树进行分割，将每次搜索和计算的点数缩减到一个可接受的范围内，接着建立若干个随机投影树形成一个随机投影森林，并以其综合结果作为最终结果。

随机投影 [44] 使用一种典型的自上而下的方法来构建森林。它与其他基于树木的方法有两个主要区别：一个是通过随机选择来分割节点，另一个是利用所有的学习样品来产生森林。

局部敏感随机投影森林（LSH Forest）[45] 即 LSH 与随机投影树的组合，增加了查询数据速度。

LSH 提供了一种在海量高维数据集中快速找到最接近查询数据点的一个或一些数据点的方法。但是，LSH 并不保证一定能够找到最接近查询数据的数据，而是通过在对数据点降维之后找到近邻数据点的概率很大，数据的降维也导致某些信息的丢失，因此 LSH 不能保证 100% 的相似度，但这对于本文的数据预

处理已经达到了预期的效果，并且后期的数据嵌入及相似性比较将提取相似度高的数据，弥补 LSH 的误差。

在本文中，通过构建局部敏感随机投影森林，占用更少空间，增加查询速度，将前 100 位与查询数据相似的数据作为候选数据集进行接下来的嵌入以及相似度比较功能工作，由于语料数据庞大，这极大地缩减了需要比对的代码片段，提高了效率，缩短了代码推荐反馈时间。

2.5　语言服务器协议 LSP

2.5.1　LSP 简介

语言服务器协议（Language Server Protocol，LSP)[1] 是由 Microsoft 创建的，用于定义编程语言分析器所使用的通用语言。现今，包括 Codenvy、Red Hat 和 Sourcegraph 在内的很多公司都加入支持这一协议，并且越来越多的编辑器开发社区和编程语言社区也加入进来。详细来说，它定义了编辑器或集成开发环境与语言服务器之间使用的协议，语言服务器提供自动完成、转到定义、查找所有引用等语言特性功能。语言服务器协议用于工具（客户端）和语言智能提供程序商（服务器）之间，以将这些特性集成到工具中。

LSP 为多种编辑器，IDE 或客户端（m）的多种编程语言（n）提供高水平的支持，将原本 $m \times n$ 复杂性问题降低成了更简单的 $m + n$ 问题。例如，在传统做法中，需要为 VSCode、Sublime Text、Vim 等 IDE 分别创建 Python 插件、Java 插件及其他编程语言插件。而是现在针对每种语言，LSP 使语言社区可以将精力集中在一个高性能的语言服务器上，该服务器可以提供代码补全，悬停提示，跳转定义，查找引用等功能，而编辑者和客户社区可以专注于构建单个，高性能，直观和惯用的扩展插件，可以与任何语言服务器进行通信，以立即提供深度语言支持。

LSP 的主要想法是使该服务器和开发工具如何通信的协议标准化。这样一来，单一语言服务器可以在多种开发工具，这反过来又可以支持多种语言以最小的代价重新使用。因此，对于语言提供商和工具供应商来说 LSP 都是极具价值的。

2.5.2　LSP 原理

语言服务器作为单独的流程运行，并且开发工具（客户端）使用基于 JSON-RPC 的语言协议与服务器进行通信。LSP 基本协议由标头和内容部分（与 HTTP 相比）组成。标头和内容部分以“\r\n”分隔。

[1]https://microsoft.github.io/language-server-protocol/

标头部分由标头字段组成。每个标头字段均包含一个名称和一个值，并以“:”（冒号和空格）分隔。每个标题字段均以“\r\n”结尾。考虑到最后一个标头字段和整个标头本身都以“\r\n”结尾，并且至少有一个标头是强制性的，这意味着两个“\r\n”序列始终紧接在消息的内容部分之前。

内容部分包含消息的实际内容。消息的内容部分的请求、响应和通知使用 JSON-RPC 描述。使用 Content-Type 字段中提供的字符集对内容部分进行编码。默认为 utf-8，这是目前唯一支持的编码。如果服务器或客户端收到的标头与 utf-8 编码不同，则应以错误响应。

如图 2.3是工具（客户端）和语言服务器在例行编辑会话期间通信的示例。

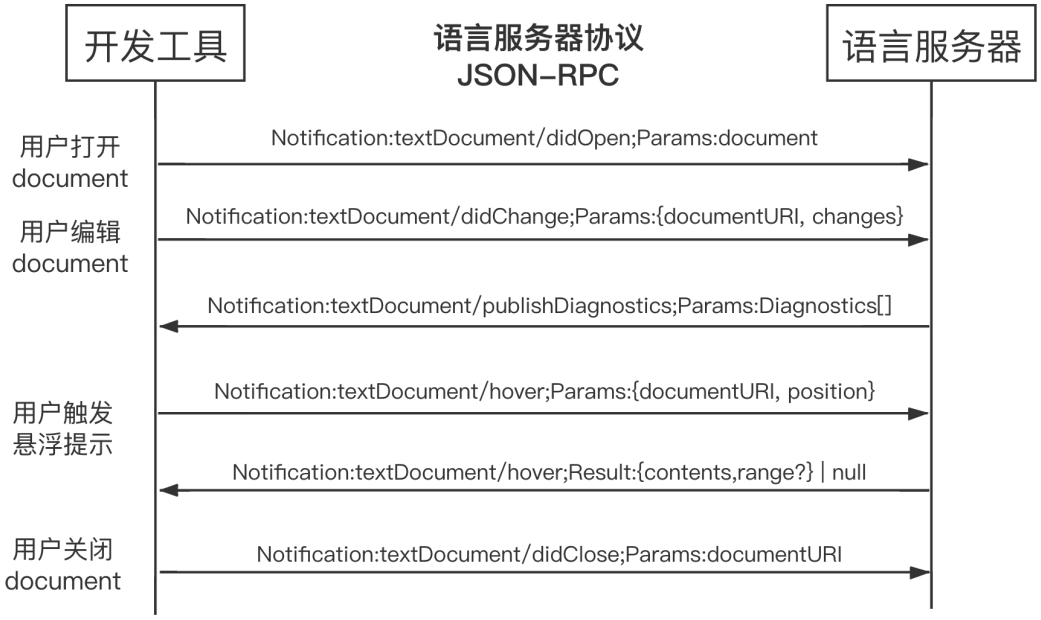

图 2.3: LSP 客户端与服务器端通信会话示例

具体每一步通信的描述如下：

用户在工具（客户端）中打开一个文件（称为文档）：该工具通知语言服务器文档已打开（“textDocument/didOpen”）。从现在开始，关于文档最新内容不再存在于文件系统中，而是由该工具保存在内存中，内容在工具和语言服务器之间同步。

用户进行编辑：该工具将文件更改（“textDocument/didChange”）操作通知服务器，并且语言服务器会更新文件的语言表示。发生这种情况时，语言服务器

会分析此信息，并将检测到的错误和警告（“textDocument/publishDiagnostics”）通知工具。

用户在打开的文档的某个符号上执行“悬浮提示”：该工具发送带有两个参数（文档 URI 和“悬浮提示”的文本位置）的“textDocument/hover”请求到服务器。服务器以文档 URI 和提示内容作为响应。

用户关闭文档（文件）：该工具会发送“textDocument/didClose”通知，通知语言服务器该文档现在不再在内存中。这时，文件内容将会更新到文件系统。

本文在 Monaco Editor 中对不同编程语言遵循语言服务器协议实现编程语言特性，每个用户将使用单独的语言服务器对他提供服务，保证准确性和独立性。

2.6　本章小结

本章主要对在系统实现时选择和采用的技术和解决方案进行简单介绍和描述。首先，介绍了源代码语法解析过程中重要的数据结构抽象语法树。接着，介绍了用于源代码片段嵌入分析，提取特征向量的词嵌入 FastText 技术和句嵌入 SIF 加权模型。然后，介绍了用于缩短代码推荐反馈时间的局部敏感哈希方法 LSH。最后，介绍了帮助实现编程语言特性的语言服务器协议 LSP。

第三章　智能代码推荐系统的需求分析与概要设计

本章节将概述智能代码推荐系统的需求与设计。首先，介绍项目背景、项目整体情况、系统开发目的和目标以及系统提供的功能。接着，从功能和非功能角度分析系统的需求并以此得出用例。其次，介绍系统总体设计，包括系统总体架构设计、4+1 视图和系统模块划分。最后，依次对系统各模块的架构设计与概念类图进行详细描述。

3.1　系统整体概述

由于本地编程工具和插件安装复杂、更新频繁、开发环境搭建较困难，经常成为阻碍用户学习开发的原因。而随着容器技术进入主流以及 Web 应用的快速发展，使用 WebIDE 进行编程开发成为可能。WebIDE 的免安装、免环境配置、无需考虑兼容性和软件升级等优势，使得它在编程教学场景下的利用率更高、适应性更强。不同背景、不同目的进行编程学习的用户都可以快速进行上手。然而，现有的 WebIDE 在代码编辑过程中缺少完善的智能提示功能。并且在编程过程中，开发者特别是编程初学者，经常需要完成的是一些他们并不熟悉的编码任务。一般情况下，用户都会通过查阅书籍或浏览器检索来获取自己需要的资料，这就消耗了用户的编程时间，降低了开发效率。如何获取优质的代码片段，减少代码搜索时间，帮助用户进行编程学习是一个亟待解决的问题。

系统提供智能代码推荐服务，通过智能化的编程系统对开发者的编程行为和环境进行持续观察，以推荐、信息可视化等方式为用户提供智能化服务支持。

系统旨在提供友善、精确的智能代码推荐服务，为用户提供在线智能开发环境，以提高开发效率、完善编程体验。系统主要划分为五个模块，编辑器模块主要用于提供完善的编程语言编辑环境，负责编辑器文件管理的功能。语言服务协议模块主要用于提供代码诊断、自动补全、跳转定义、悬浮提示、引用查找等编程语言特性服务功能。代码搜索模块主要提供代码片段或方法的搜索以及搜索结果列表的查看功能。特征提取模块主要提供预料数据爬取和源码语法解析的功能。特征序列分析模块主要提供将特征序列转化为向量以及相似度分析的功能。

系统前端基于 React 框架开发，后端主要由 Spring Boot 框架搭建，代码推荐核心服务基于 Flask 框架开发。系统主要数据存储于 Mysql，NAS 存储文件用于共享，辅助 Elastic Search 搜索引擎实现对大数据的查询以提高系统性能。

3.2　系统需求分析

3.2.1　功能性需求

智能代码推荐系统的主要功能性需求分为编辑器文件管理、编辑器语言服务、系统设置、搜索代码、查看搜索代码结果、代码推荐六部分。具体如表3.1所示。

表 3.1: 功能需求列表

需求 ID	需求名称	需求描述
R1	编辑器文件管理	用户能够在编辑器中打开单个文件、打开多个文件、查看文件内容、修改文件内容、关闭文件、保存文件。系统在用户进行代码编辑时能够自动保存文件。
R2	编辑器语言服务	用户在编辑器内编辑 Java 或 Python 代码时，系统能够对用户正在编写的代码进行自动补全，能够对用户鼠标停留的字段进行悬浮提示，能够对代码中使用方法的地方进行跳转定义，能够提供每个方法被引用次数，能够对代码诊断进行错误提示。
R3	系统设置	用户能够对整个系统的主题、语言进行修改。
R4	搜索代码	1. 用户能够点击代码搜索按钮打开代码搜索面板，在输入框中输入代码片段或方法名称点击按钮进行搜索。 2. 用户打开代码进行操作时，系统能够主动获取用户正在编辑的代码语言，并在搜索框内容为空的时候主动获取鼠标停留的代码字段进行搜索。
R5	查看搜索代码结果	用户能够在代码搜索面板查看搜索结果列表，包括源代码行及行数，主要编程语言，相关总代码行数，源代码项目名称和地址等信息。
R6	代码推荐	用户进行代码编辑时系统能够主动获取代码，向用户推荐超过阈值的前五位不同相似度代码片段。

3.2.2　非功能性需求

除了上述功能性需求外，系统还需满足的非功能性需求，如表 3.2所示。

表 3.2: 非功能需求列表

安全性	系统应保证仅有一个统一入口。
	系统应保证仅有一名管理员可以更改权限。
	系统应保证数据在传输过程中进行加密保护，不被他人偷窃、篡改。
健壮性	系统内部错误，不能导致应保证在任何情况下都不会崩溃
可靠性	当遇到故障时，应保证能在 1 个小时内可以修复，恢复系统能够正常运转。
	系统故障发生概率应保证维持在 2% 以下。
可维护性	应使用日志记录系统，保证课追溯历史系统使用情况。
可扩展性	系统类似组件应保证统一设计，保证在别处简单修改后即可使用。
易用性	应保证用户界面美观、交互友善、提示信息明确。
	应保证新用户可以在 15 分钟内熟练使用系统所有功能。
	应保证系统所有功能操作在用户最多三次操作后完成。
性能	系统页面加载和渲染需要在 5 秒内完成。
	系统主要功能响应时间不能超过 1.5 秒，特殊功能除外。
	系统能同时支持超过 1000 人访问，200 人运行提交操作。
兼容性	系统为 web 应用，应保证兼容各主流浏览器及其主要版本。

智能代码推荐系统为满足在考试或练习期间大量学生同时在线进行操作的状况，需要大量的硬件资源和网络资源，因此特别要求系统的可扩展性，健壮性和高可用性。为满足用于在线编辑的功能，需要考虑系统的易用性，保证页面美观、操作简单，减少用户系统学习时间。系统的性能需要考量系统响应时间、并发数、资源利用率等，提升系统性能有助于提高用户体验，提高资源利用率，减少成本。随着业务的丰富、流程变多，系统的可维护性和可扩展性也需要保证。

3.2.3　系统用例图

根据上述功能性需求和非功能性需求分析，获得系统用例图如图 3.1所示。本系统主要用户为老师和学生，但用户在系统新功能使用中并无区别，因此此处统一分析。

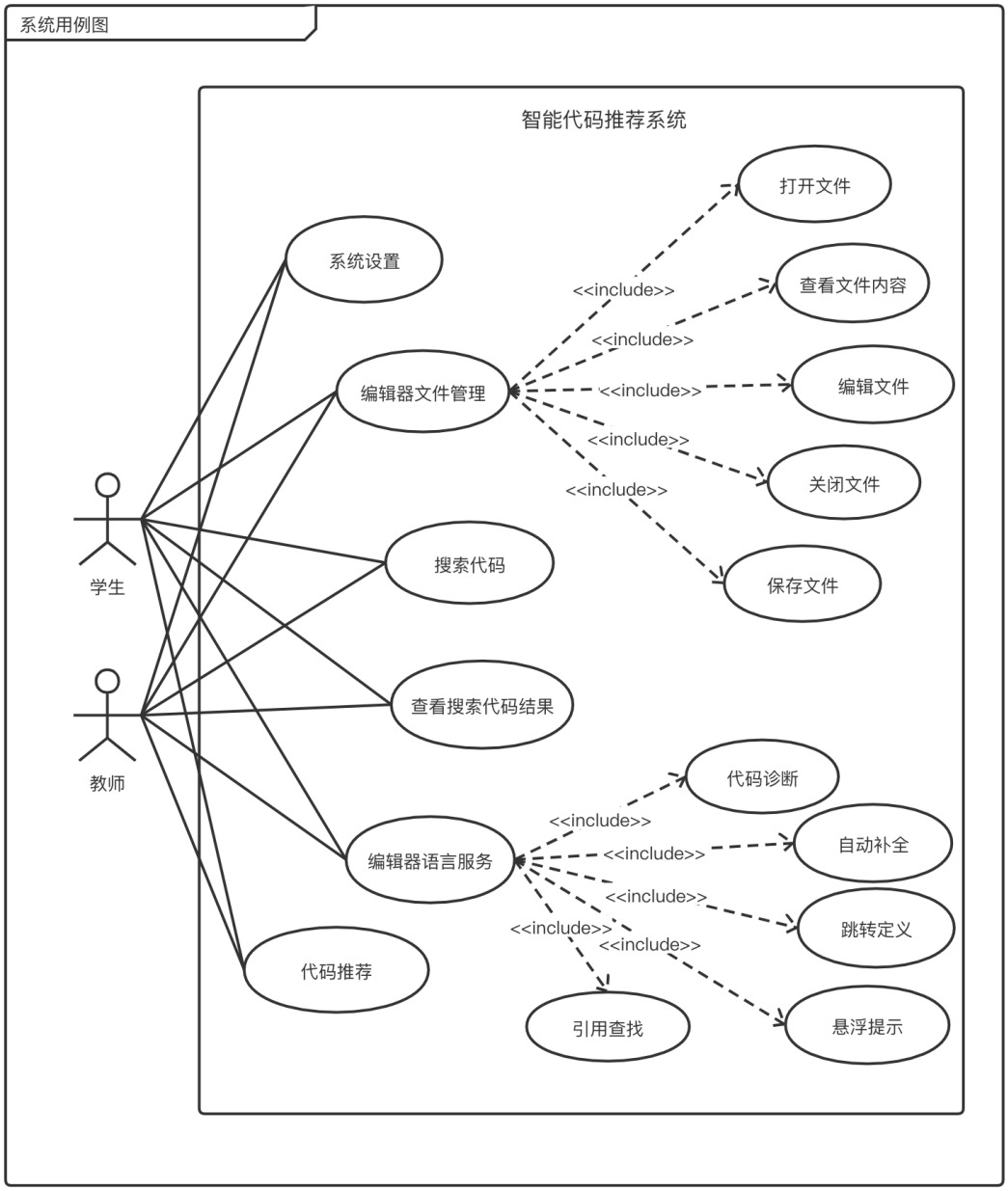

图 3.1: 系统用例图

本系统主要有 6 个用例，分别为系统设置、编辑器文件管理、搜索代码、查看搜索代码结果、编辑器语言服务、代码推荐。

3.2.4　系统用例描述

本小节将就上述 6 个用例进行详细描述，包括其参与者、触发条件、前置条件、后置条件、优先级、正常流程、扩展流程、特殊需求。

表 3.3: 系统设置用例描述

ID	UC1	名称	系统设置
参与者	学生或老师，目标是方便用户对系统的主题、语言进行自定义		
触发条件	用户想要更改系统设置		
前置条件	用户已被认证并且具有该权限		
后置条件	系统对更改进行保存应用		
优先级	中		
正常流程	1. 用户点击“设置”按钮； 2. 系统显示设置控制面板； 3. 用户点击“通用”标签； 4. 系统显示“通用”面板； 5. 用户点击“语言”下拉框中“中文”或“英文”选项； 6. 系统在下拉框中保存选项； 7. 用户点击“确认”； 8. 系统保存用户选项并更新界面语言； 9. 用户点击“样式”标签； 10. 系统显示“样式”面板； 11. 用户点击“UI 主题”下拉框，点击“黑色”或“默认”选项； 12. 系统在下拉框中保存选项并更新页面样式风格；		
扩展流程	无		
特殊需求	1. 当页面刷新后，仍然保留用户系统设置；		

系统设置功能主要是让用户对整个系统的主题、语言有自定义能力，该功能主要为辅助功能，帮助用户进行个性化界面和配置的设定，用于提高系统风格的多样性以及友好性。其用例描述如表 3.3所示。

表 3.4: 编辑器文件管理用例描述

ID	UC2	名称	编辑器文件管理
参与者	学生或老师，目标是方便用户在编辑器区域对文件内容进行查看、修改、保存、对文件进行关闭操作		
触发条件	用户双击一个或多个文件		
前置条件	用户已被认证并且具有该权限		
后置条件	系统对文件进行更新		
优先级	高		
正常流程	1. 用户双击一个文件； 2. 系统在编辑器标签区域创建一个标签，并将文件内容显示在编辑器内； 3. 用户更改编辑器内文件内容； 4. 系统在文件标签处显示编辑状态，在 1 秒后去除编辑状态并保存当前文件内容； 5. 用户快捷键操作保存文件； 6. 系统去除文件标签处编辑状态并立即保存当前文件内容； 7. 用户在编辑器标签区右键“关闭标签”按钮； 8. 系统在编辑器标签区去掉这个文件标签，并将编辑器内容置为后一个标签的文件内容； 9. 用户点击编辑器标签区关闭按钮图标； 10. 系统在编辑器标签区去掉这个文件标签，并将编辑器内容置为后一个标签的文件内容；		
扩展流程	1a. 用户再次双击一个文件； 1. 系统在编辑器内显示当前文件内容，其他已被打开的文件只展示标签； 7a. 编辑器标签区只有一个文件的标签存在； 1. 系统在编辑器标签区去掉这个文件标签，并将编辑其内容清空； 9a. 编辑器标签区只有一个文件的标签存在； 1. 系统在编辑器标签区去掉这个文件标签，并将编辑其内容清空；		
特殊需求	无		

编辑器文件管理主要为文件操作及文件内容提供可视化功能。用户能够在编辑器中打开试题或习题文件夹中的一个文件或多个文件，进行文件内容查看、内容修改、保存，文件进行关闭等操作。其用例描述如表 3.4所示。

编辑器语言服务主要为用户提供编程语言特性功能。系统对用户的鼠标和键盘操作进行监听，当用户正在输入字符时自动补全。当用户鼠标在字段处停留时，系统会悬浮提示。系统对用户正在编辑的方法的被引用次数会标记提示。用户可以通过鼠标和键盘合作对代码中的方法体和定义进行跳转。系统对代码中的语法错误内容可以进行诊断提示。该服务通过提供更丰富的提示信息辅助用户编程，帮助用户提高编码效率，增强用户体验。其用例描述如表 3.5所示。

表 3.5: 编辑器语言服务用例描述

ID	UC3	名称	编辑器语言服务
参与者	学生或老师，目标是对代码进行自动补全、悬浮提示、引用标记、定义跳转、代码诊断等操作，辅助用户完成编码行为		
触发条件	用户在编辑器中进行键盘输入或鼠标操作		
前置条件	用户已被认证并且具有该权限，语言服务连接已建立		
后置条件	页面关闭时需要断开语言服务连接		
优先级	高		
正常流程	1. 用户在编辑器内打开文件； 2. 系统对代码中方法的引用次数进行标记显示; 3. 用户在编辑器内输入字符； 4. 系统弹出悬浮框，提示代码补全信息； 5. 用户鼠标在字段处停留 1s 及以上； 6. 系统弹出悬浮框，提示字段相关文档信息； 7. 用户通过鼠标和键盘组合 (Mac：Command+ 单击，Windows：Control+ 单击) 操作代码中的方法或类； 8. 系统编辑器光标自动跳转到该方法或类定义位置； 9. 用户在编辑器内输入有语法错误的代码； 10. 系统对有语法错误的代码位置进行红色波浪提示；		
扩展流程	1a. 语言服务连接尚未建立或建立失败； 1. 系统编辑器内不会有任何提示；		
特殊需求	无		

表 3.6: 搜索代码用例描述

ID	UC4	名称	搜索代码
参与者	学生或老师，目标是为用户提供代码片段或方法搜索的能力，帮助了解代码使用		
触发条件	用户点击“搜索”图标按钮		
前置条件	用户已被认证并且具有该权限，并且代码搜索功能开关已被开启		
后置条件	系统更新搜索结果内容		
优先级	高		
正常流程	1. 用户点击“代码搜索”标签； 2. 系统展示“代码搜索”面板； 3. 用户在输入框中输入代码片段或方法，并点击“搜索”图标按钮; 4. 系统显示搜索结果列表； 5. 用户在输入框为空的情况下，鼠标在代码字段处停留 3s； 6. 系统自动提取关键字填入输入框中并触发搜索请求，显示搜索结果列表； 7. 用户更改输入框内容； 8. 系统清空搜索结果列表；		
扩展流程	3a. 用户在输入框内输入空内容； 1. 系统在页面上提示“请输入搜索内容”； 3b.5a 用户搜索代码片段或方法返回结果为空或请求失败； 1. 系统在页面上提示“暂无内容”； 3c.5b 正在进行搜索请求，但结果未返回时； 1. 系统在页面上提示“正在搜索中，请稍候...” 5c. 当“代码搜索”面板没有被打开时； 1. 系统不进行任何搜索操作；		
特殊需求	无		

搜索代码功能主要有两种方式触发。第一种是用户主动在输入框内输入代码片段或方法名称进行主动搜索。第二种是用户在进行代码编辑操作时，系统会实时获取用户鼠标所在关键字段，在输入框为空的情况下，用户行为停留超

过 3s 的情况下自动触发搜索。这一功能可帮助用户在一定程度上减少搜索路径和站外搜索行为，将搜索操作都放在站内完成。该功能主要关注该操作能否简单快速的达成用户需求以及搜索结果是否能对用户有帮助和价值。其用例描述如表 3.6所示。

查看搜索代码结果主要功能是帮助用户快速获取与搜索输入相关的代码片段信息。搜索结果列表需要显示源代码片段、代码在源项目中行数、主要编程语言、所在项目相关代码总行数、代码片段所属项目名称和地址等信息。该功能主要关注搜索结果列表内容是否清晰易懂，与查询字段是否相关且准确。如果用户需要进一步了解该代码片段内容，可通过系统提供的源项目链接，进行项目访问，方便用户进行进一步学习了解。其用例描述如表 3.7所示。

表 3.7: 查看搜索代码结果用例描述

ID	UC5	名称	查看搜索代码结果
参与者	学生或老师，目标是获得代码片段或方法使用信息		
触发条件	用户点击“代码搜索”面板		
前置条件	用户已被认证并且具有该权限，搜索结果正确返回		
后置条件	无		
优先级	高		
正常流程	1. 用户点击“代码搜索”标签； 2. 系统展示“代码搜索”面板； 3. 用户点击代码所属项目名称； 4. 系统在浏览器新标签页中打开链接； 5. 用户点击“上一页”按钮； 6. 系统在页面上显示当前页上一页的搜索结果内容； 7. 用户点击“下一页”按钮； 8. 系统在页面上显示当前页下一页的搜索结果内容；		
扩展流程	5a. 如果当前页为第一页； 　1. 系统禁止点击操作，不作任何反馈； 7a. 如果当前页为最后一页； 　1. 系统禁止点击操作，不作任何反馈；		
特殊需求	1. 搜索结果一页仅显示 20 条数据； 2. 搜索关键词在代码中用黄色高亮；		

表 3.8: 代码推荐用例描述

ID	UC6	名称	代码推荐
参与者	学生或老师，目标是推荐高质量的相似代码片段帮助用户学习和完善代码		
触发条件	用户在编辑器内光标停留超过五秒或用户点击“推荐代码”按钮		
前置条件	用户已被认证并且具有该权限，并且代码推荐功能开关已被开启		
后置条件	无		
优先级	高		
正常流程	1. 用户光标在编辑器内停留超过五秒； 2. 系统获取用户光标上方或所在的方法体代码内容进行分析，在页面上展示前五名具有不同相似度的超过阈值的精确度高的代码片段及其所属项目名称、所属项目地址、所属类名、所属方法名、方法开始行、方法结束行等信息； 3. 用户点击“推荐代码”按钮； 4. 系统获取用户光标上方或所在的方法体代码内容进行分析，在页面上展示前五名具有不同相似度的超过阈值的精确度高的代码片段及其所属项目名称、所属项目地址、所属类名、所属方法名、方法开始行、方法结束行等信息；		
扩展流程	1a.3a. 当编辑器光标处于代码开头； 1. 系统对最后一个方法体进行代码片段推荐； 1b. 当编辑器内无方法体存在或系统分析后得到的代码推荐结果为空； 1. 系统不主动进行代码推荐； 3b. 当编辑器内无方法体存在或系统分析后得到的代码推荐结果为空； 1. 系统提示“暂无可推荐代码”；		
特殊需求	1. 主动代码推荐功能默认关闭； 2. 用户可以关闭或开启主动代码片段推荐功能；		

代码推荐用例描述如表 3.8所示。主要功能是在用户编写代码过程中，对用

户编辑行为暂停超过五秒或用户主动触发的情况下，系统进行用户代码分析，推荐与用户代码最相关的五个代码片段信息。被推荐的代码片段信息主要包括代码片段、代码所属项目名称、代码所属项目地址、代码所属类名、代码所属方法名、方法开始行、方法结束行等信息。该功能主要关注自动推荐是否及时，系统推荐的超阈值前五个具有不同相似度的代码片段精度如何，是否对用户有价值，是否能让用户学习，对现有代码进行补充和完善。

3.3　系统总体设计

需求分析部分给出系统的功能和非功能需求。在本节中，将结合需求进行系统概要设计。在需求分析的基础之上，进行体系结构的设计，以此对系统进行模块划分，再进行各模块的设计，确定其职责。具体地，本节将先描述系统总体架构设计，然后给出包括场景视图、逻辑视图、进程视图、物理部署视图、开发视图这五个不同角度的系统架构视图。并以此为依据划分出主要业务模块。

3.3.1　系统总体架构设计

智能代码推荐系统的整体架构见图 3.2。

本系统前后端分离，前端主要负责展示与交互，后端主要处理业务逻辑和数据。主要目的是增加灵活性，使得职责分离，降低耦合度，提高开发效率。

前端基于 React 框架开发，配合使用 Redux 和 Mobx 作为状态管理。采用 Webpack 进行项目管理和模块打包。前后端通信部分，系统使用 axios 进行封装用于请求处理。对于一些长连接请求和服务端主动推送需求，为兼容各主流浏览器使用 WebSocket，系统采用 SockJS 搭建全双工通信管道，搭配 Stomp 实现消息订阅。前端核心编辑器以 MonacoEditor 为基础，结合 React 进行封装，实现了组件化使用。系统遵循 LSP 语言服务协议创建语言服务客户端，通过 JSON-RPC 与服务器通信实现对编程语言的自动补全、跳转定义或悬停文档等功能的支持。

本系统服务器端主要采用 Spring Boot 框架，编码、配置和部署简易，且包含大量常用的第三方库和持续的社区支持，可以使开发者更加集中于业务逻辑，快速完成所需服务以及对接任务。代码片段推荐服务的核心逻辑与算法基于 Flask 框架实现。其中，LSH 局部敏感哈希模型和 FastText 词模型单独对外提供服务，方便一次加载和读取。系统通过 Docker 容器化为用户隔离出独立的代码运行环境，这保障了一定的灵活性、容错性和可扩展性。系统主要数据结构使用 MySQL 进行数据存储和持久化，语料数据使用 Elastic Search 进行存储和搜索，加快查询速度。

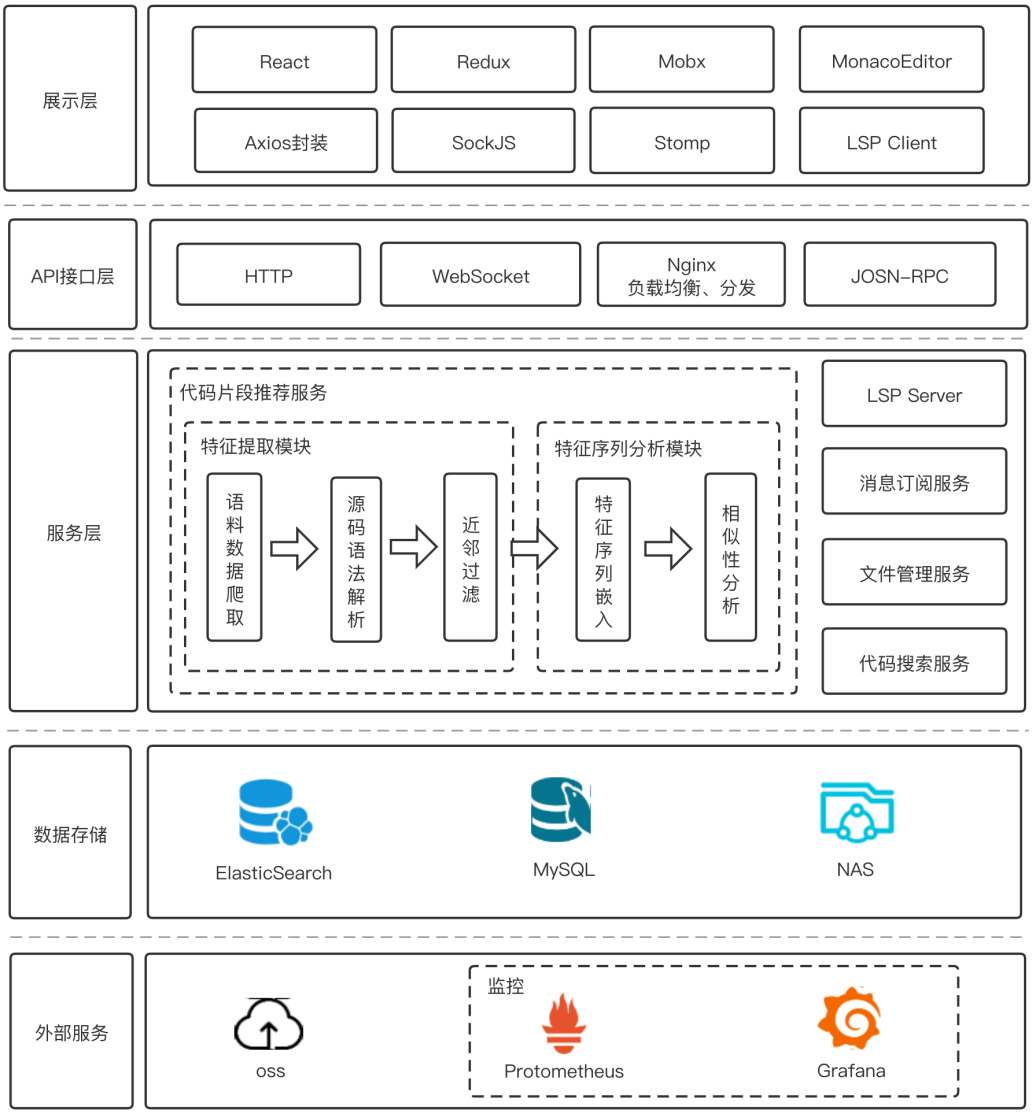

图 3.2: 系统架构图

本系统中前端和服务端使用 Nginx 实现负载均衡和反向代理。系统使用 Prometheus 配合 Grafana 提供监控以防服务器持续高负载。并使用 NAS 进行文件共享存储，OSS 用于编程试题的存储。为了保证数据可靠性和高可用性，系统利用 ElasticSearch 搜索引擎进行代码推荐相关语料数据和分析数据的采集处理和存储。

3.3.2　4+1 视图

本小节将从场景视图、逻辑视图、进程视图、开发视图、物理部署视图描述系统的总体架构设计。其中场景视图又可称为用例视图，联系其他四个视图，作为最重要的需求抽象，在 UML 中与用例图对应，因此本系统的场景视图如图 3.1所示。

逻辑视图涉众为最终用户，主要关注系统功能，通过 UML 总类图来表述，系统被分解为一组功能抽象，如图 3.3所示。

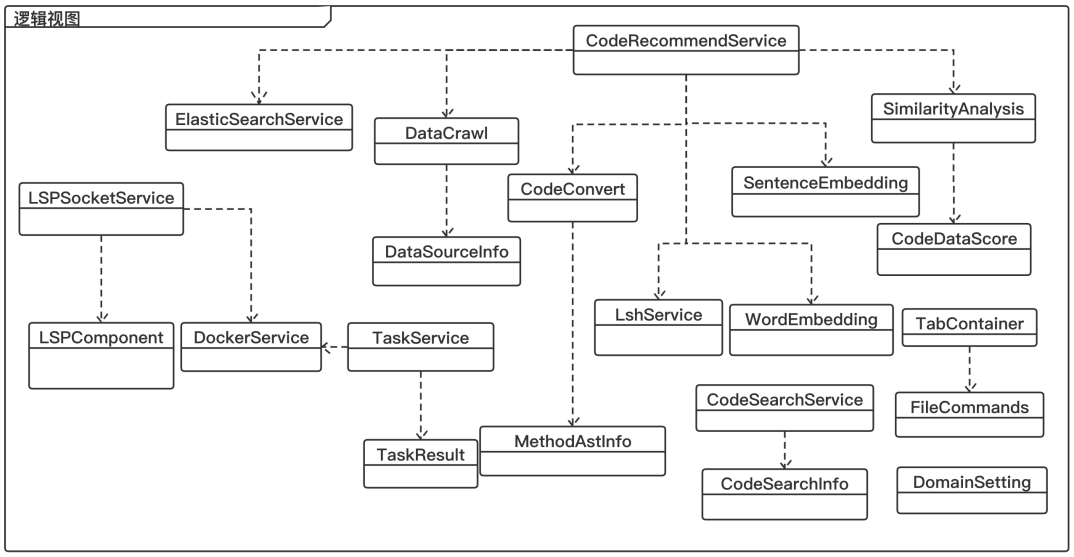

图 3.3: 逻辑视图

LSPSocketService 提供编辑器语言服务连接建立功能。LSPComponent 提供多编程语言语言服务消息处理功能。TaskService 提供项目运行和提交结果信息管理功能。TaskResult 提供提交的程序运行返回的结果。DockerService 提供容器管理的功能，包括容器创建、关闭、移除、获取等操作。TabContainer 提供编辑器内文件标签管理，包括新建标签、关闭标签、打开标签等功能。FileCommands 提供编辑器内文件打开、保存文件等功能。DomainSetting 提供系统设置更改的功能。其中 changeUITheme 负责系统主题的更改，updateSettingItem 负责系统语言的更改。CodeSearchService 提供代码片段和方法的搜索功能，CodeSearchInfo 提供搜索结果信息管理的服务。ElasticSearchService 提供搜索引擎增删改查服务。CodeRecommendService 需要调用多个功能来实现相似代码推荐。其中 DataCrawl 提供语料数据爬取功能。LshService 提供局部敏感哈希分类功能。CodeConvert

提供源码语法解析功能。WordEmbedding 提供特征词嵌入功能。SentenceEmbedding 提供特征序列句嵌入功能。SimilarityAnalysis 提供句向量相似度计算功能。CodeDataScore 提供相似代码推荐结果信息管理功能。

进程视图涉众为系统设计人员和集成人员，主要关注系统非功能需求，比如性能、可用性、健壮性、容错性和整体性等，旨在解决进程、线程、并发、异步、同步、通信等方面的问题。本系统进程视图如图 3.4所示。前端通过与主服务进程通信完成代码运行、提交、文件保存、建立语言服务连接、发起语言服务各类操作事件请求、代码搜索、代码推荐等功能。主服务进程通过调用异步进程完成任务的运行和提交以及代码推荐功能。其中，提交代码等数据存储需求会通过与 DataBase 通信完成。代码推荐的数据查询工作需要与搜索引擎通信完成。主服务进程与语言服务进程通信完成编程语言的事件通信。代码搜索功能由主服务进程与搜索服务通信完成，结果将直接返回用于前端展示。文件关闭、查看内容、编辑等功能的支持由前端独立完成。文件保存和打开功能由服务器与 NAS 通信完成数据获取与更新，保证一致性。

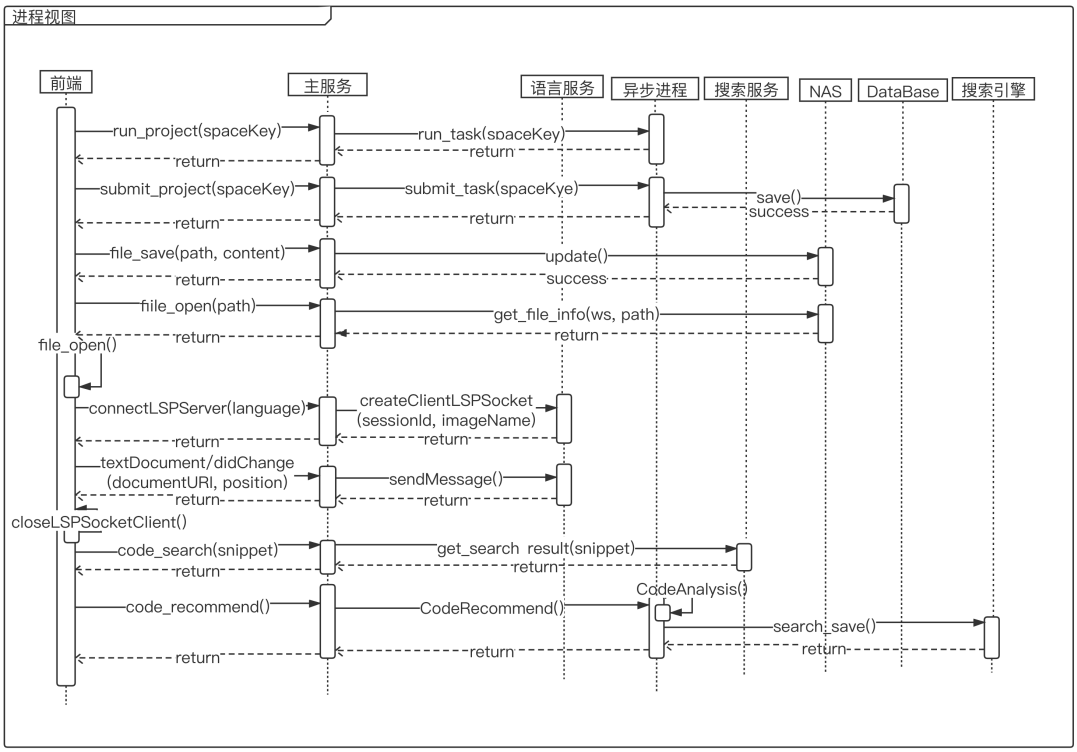

图 3.4: 进程视图

开发视图涉众为开发人员和产品经理，主要关注组织、可重用性、可移植

性、产品线，侧重于描述系统模块的组织以及开发环境结构。本系统开发视图如图 3.5所示。

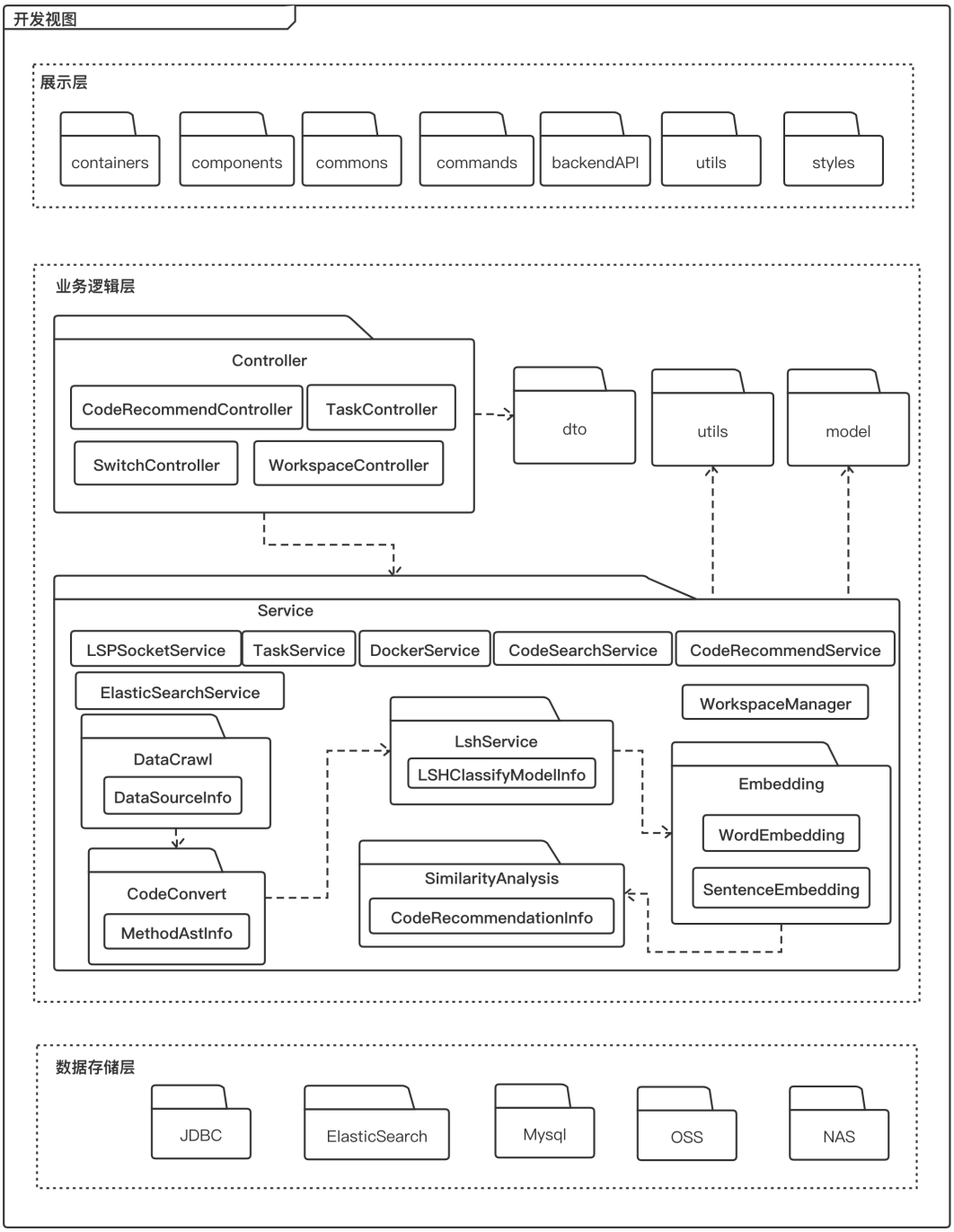

图 3.5: 开发视图

系统展示层划分为 Containers 容器组件、Components 展示组件、Commons 通用组件、Commands 命令集、backendAPI 接口请求封装、utils 通用方法、styles 样式。业务逻辑层依照系统业务功能进行划分。Controller 主要负责控制业务流程，调用特定 Service 类提供的接口对接收的前端请求进行处理，结果转化成 dto 用于服务层与展示层之前的传输。Service 包负责处理业务逻辑，调用数据层的接口，进行数据进行存储和插查询工作，实现语言服务建立、代码运行提交、文件管理、代码搜索、代码推荐相关的业务逻辑。DataCrawl 包负责源语料数据的爬取和筛选。CodeConvert 包负责源代码中方法体基本信息的构建。LshService 包负责实现局部敏感哈希算法。Embedding 包负责特征字嵌入和词嵌入的计算。SimilarityAnalysis 包负责向量相似性计算和筛选的实现。数据存储层主要提供数据的存储和高效检索，包括 ElasticSearch 对语料数据的搜索、Mysql 存储基本数据信息、OSS 存放题目信息、NAS 存放文件信息，保证数据一致性和持久化。

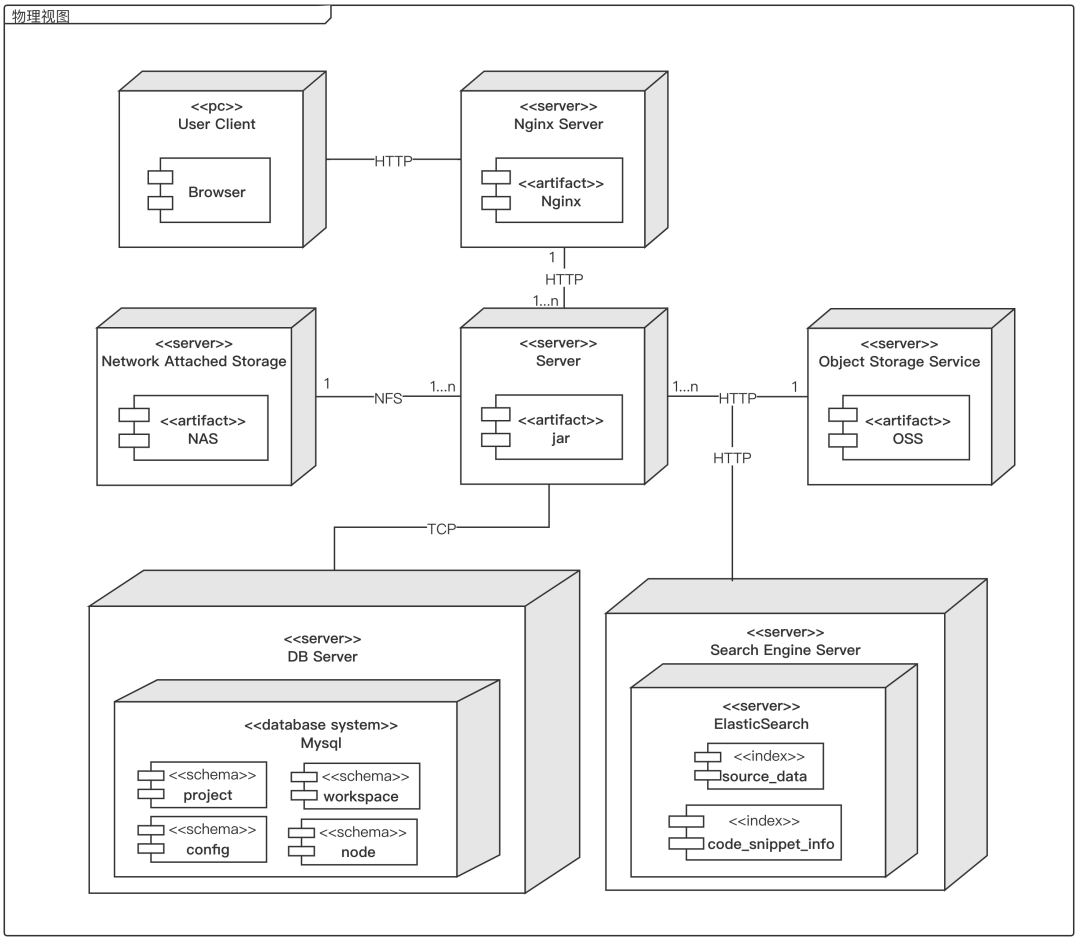

图 3.6: 物理视图

物理视图通过节点信息描述的是软件到硬件的映射，主要关注系统可伸缩性、性能和可用性。本系统物理视图如图 3.6所示。用户通过浏览器访问系统，通过 Nginx 实现负载均衡和反向代理，进行 HTTP 请求的转发，实现前后端分离。对于第一次页面访问，服务端从 OSS 获取试题信息。之后的文件操作包括获取、保存、删除等都是服务端从 NAS 获取及同步信息。应用服务端与数据库服务器通过 TCP 通信，完成数据的存储与查询。搜索引擎服务在大数据量查询时使用，以获得比关系型数据库更快的查询速度。

3.3.3　系统模块划分

智能代码推荐系统主要模块划分如图 3.7所示，包括编辑器模块、语言服务协议模块、代码搜索模块、特征提取模块、特征序列分析模块。

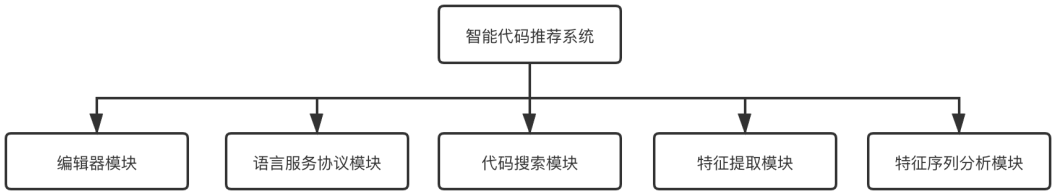

图 3.7: 系统模块划分图

编辑器模块以 Monaco Editor 为基础结合 React，进行封装，实现组件化使用，主要功能包括文本的显示、复制、粘贴、剪切、撤销、支持正则的查找、自动保存，以及自动识别文件类型、编辑器主题设置、编程语言高亮、简单代码提示和语法检查等。

语言服务协议模块使用 Monaco Editor 作为主要编辑器基础，并遵循 LSP 语言服务协议实现 Java、Python 语言客户端的单实例。系统通过监听用户鼠标和键入操作，提供实时代码补全、悬浮提示、跳转定义、引用标记、代码诊断等智能功能。

代码搜索模块主要通过自动获取用户编辑器中文件的编程语言以及提取用户鼠标关注点字段进行代码片段或方法搜索推荐，也支持用户主动输入触发搜索。搜索结果提供包括源代码行及行数，主要编程语言，相关总代码行数，源代码项目名称和地址等信息。

特征提取模块主要功能包括对数据爬取，源码进行词法分析、语法分析构建方法级抽象语法树，简化抽象语法树，获取方法令牌特征序列，进行近邻过

滤。语料库主要用于训练句嵌入并且用于代码片段推荐，因此需要高质量的代码作为来源。本系统数据来源主要为 GitHub 上标星较多的开源项目，且以 Java 编程语言为主。抽象语法树含有源代码语义信息，但对于较大的源代码来说，其中包含了一些对于训练无用的符号、变量名等信息。这使得语法树结构过于大，且希望推荐内容精炼的源代码片段，因此需要对抽象语法树进行简化，获得方法级简单解析树，这也有利于源代码的信息提取和保存。再使用局部敏感哈希对特征序列进行近邻过滤，可以帮助快速分类相似度高的代码，减少代码比较次数，缩减最终代码相似计算对比时间，以及推荐时间。

特征序列分析模块主要功能包括基于令牌特征序列生成词嵌入，生成句嵌入，相似性分析。其中嵌入的过程就是将单词转化为向量的过程，考虑并处理了词序丢失的问题。对每一个方法片段生成一个句向量进行最终相似性计算。相似性分析主要使用基于句嵌入的余弦相似度分析，选择高于阈值的前五位不同相似度代码片段作为最终代码片段推荐给用户。

3.4　智能代码推荐系统各模块设计

3.4.1　编辑器模块

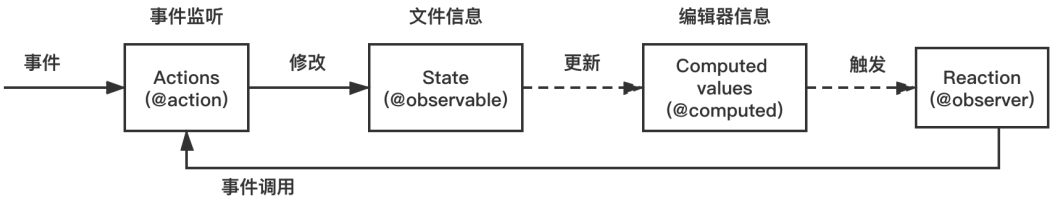

图 3.8: 编辑器模块数据流

编辑器模块的数据流设计如图 3.8所示。本系统前端选择 MobX[1] 作为状态管理工具，状态 state 被用于描绘 UI 界面，与 React 可以很好地搭配使用，作为 View 层的框架，React 通过虚拟 DOM 机制优化 UI 呈现，MobX 提供了一种将相应状态同步到 React 的机制。MobX 添加了 Observable 装饰器，使常规的 JavaScript 对象能够通过隐式订阅实现更改的自动跟踪。在编辑器模块中需要对文件信息 state 添加观察。当文件名，文件状态，文件内容等被 actions 修改时，自

[1]https://github.com/mobxjs/mobx

动更新用纯函数从 state 中推导出的值，包含编辑器内的相关信息，标签名、标签状态、编辑器内显示的文件内容、编辑器语言、编辑器主题等。Reaction 也会对 state 的变化做出响应，产生例如更新 UI 的副作用。编辑器模块需要将用户选择的文件在编辑器内呈现，当监听到对文件信息的更改就会更新编辑器相关信息，并且更新 UI。

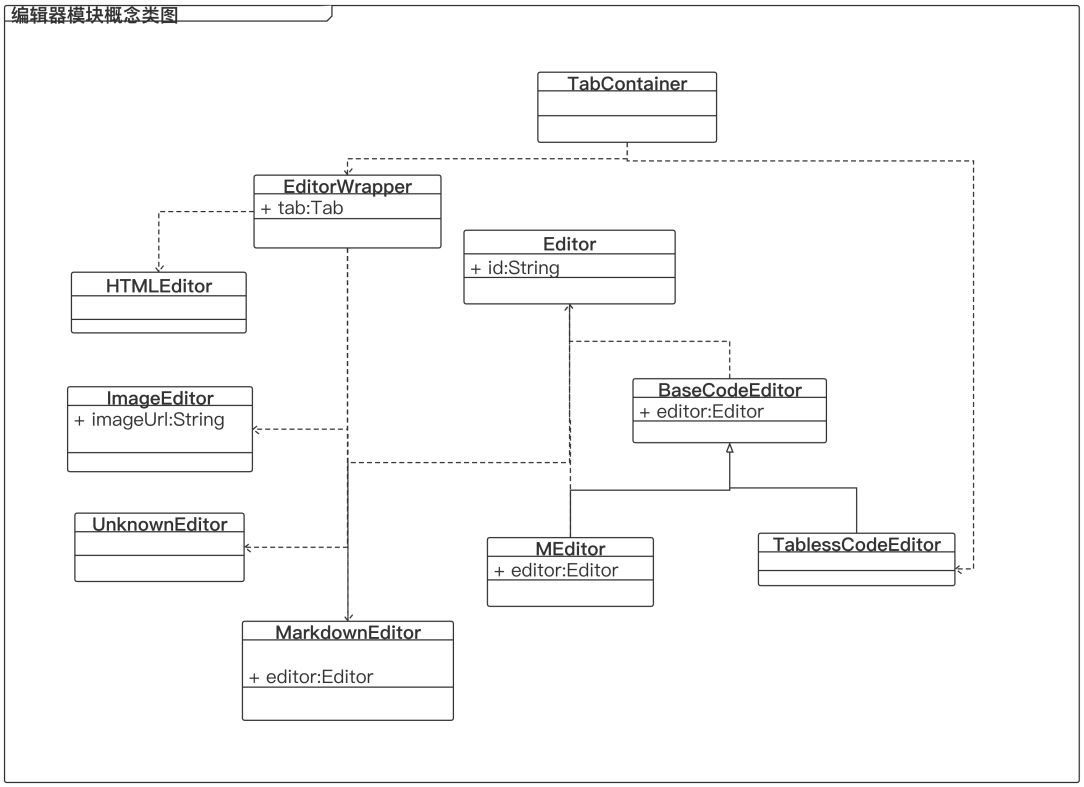

图 3.9: 编辑器模块概念类图

编辑器模块的概念类图见图 3.9。编辑器模块的主要类职责为：TabContainer，为 EditorWrapper 和 TablessCodeEditor 提供容器，每一个标签包含一个编辑器组件，控制标签内容，提供数据源，并绑定标签关闭事件；EditorWrapper，用于获取标签信息，根据编辑器类型，创建合适的编辑器元素对象；HTMLEditor，编辑器类型之一，用于显示 HTML 网页内容；ImageEditor，编辑器类型之一，用于显示图片文件；UnknownEditor，编辑器类型之一，用于处理无法在网页中显示内容的文件，提供下载链接；MarkdownEditor，编辑器类型之一，用于显示 Markdown 文件；Editor，创建 Monaco 编辑器实例，管理编辑器所有基本状态，

当状态值被更新时，会根据新的状态值重新渲染用户界面；MEditor，编辑器类型之一，为默认编辑器类型，自动识别文件扩展名，显示文本内容及相应类型特性；BaseCodeEditor，将 Editor 创建的 Monaco 编辑器实例绑定在 DOM 元素上，监听并处理编辑器通用事件，例如主题切换、鼠标移动事件监听等；Tablessc-CodeEditor，当没有文件被打开，即标签栏为空时，显示的无文件类型空编辑器。

3.4.2　语言服务协议模块

语言服务协议模块的通讯架构图如图 3.10所示。微软提供 LSP 语言服务器协议定义工具（客户端）和服务器端之间通信使用的协议，语言服务器提供了自动完成、转到定义、查找所有引用等语言特性功能。本系统使用 Monaco Editor 作为编辑器，为实现 Java 和 Python 编程原因的各语言特性，遵循 LSP 协议与 Java 语言服务器和 Python 语言服务器之间建立连接。LSP 协议基于 JSON-RPC，将语言服务器整合，可作为 SaaS 单独对外提供服务，WebIDE 后端仅作为请求转发中介，为减轻 WebIDE 后端压力。对于每一个用户，前端对每一种编程语言仅建立一个语言客户端，每一个语言客户端在自己的通信管道内与特定语言服务器进行通信，完成 LSP 通知的传输，保证每一个语言连接的唯一性，避免重复建立连接。

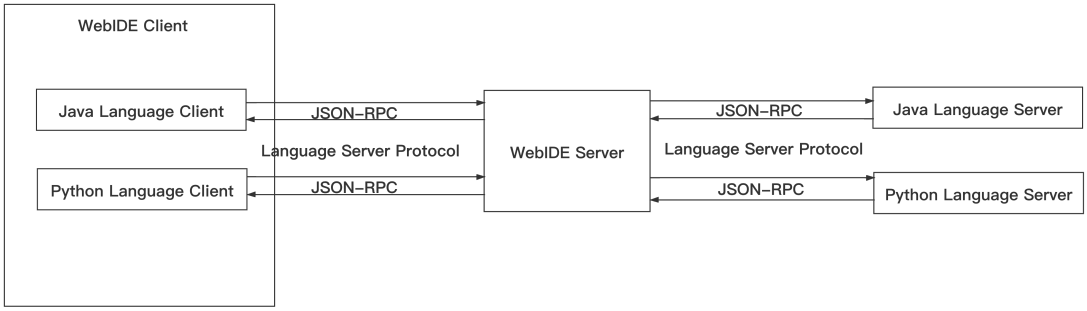

图 3.10: 语言服务协议模块的通讯架构图

语言服务协议模块的概念类图见图 3.11。语言服务协议模块的主要类职责为：LSPSocketClient，注册 Monaco 语言服务，创建客户端；MonacoServices，提供编辑器和文件根 URI，进行 workspace、languages、commands、window 实例化；BaseLanguageClient，抽象类，提供预览客户端基本模板；MonacoLanguage-Client，继承 BaseLanguageClient，从 JSON-RPC 创建语言客户端连接，并实现各类特性注册，定义 LSP 代理，实现与协议之间的转化器；MonacoCommands，提供 Monaco 编辑器注册指令的功能；MonacoLanguages，创建各类语言特性提

供者并向 Monaco 注册；ConsoleWindow，各类消息显示打印并创建输出通道；MonacoWorkspce，提供模型管理、文本文档操作监听、编辑应用等能力。

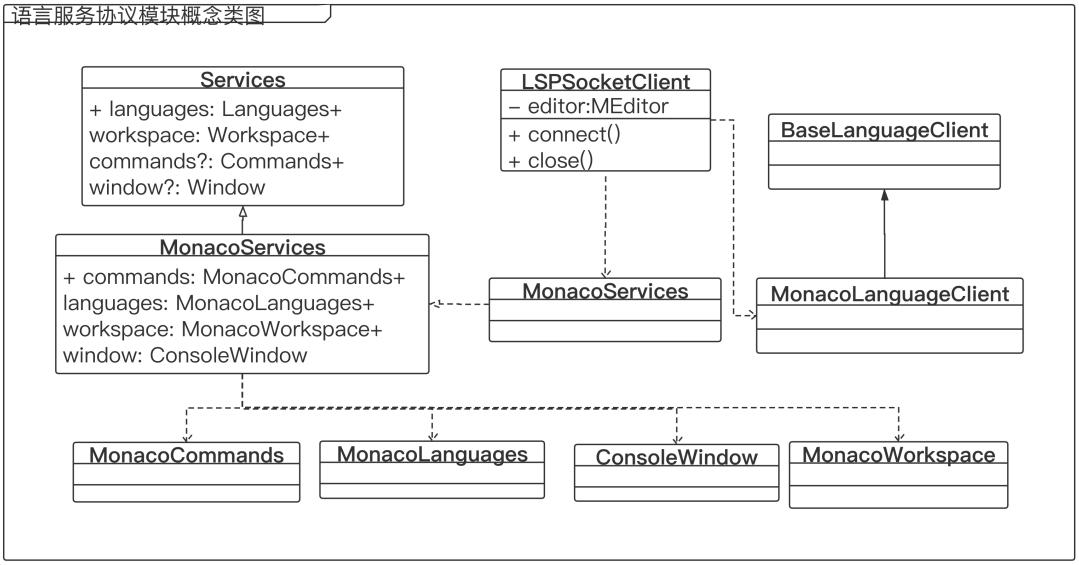

图 3.11: 语言服务协议模块概念类图

3.4.3　代码搜索模块

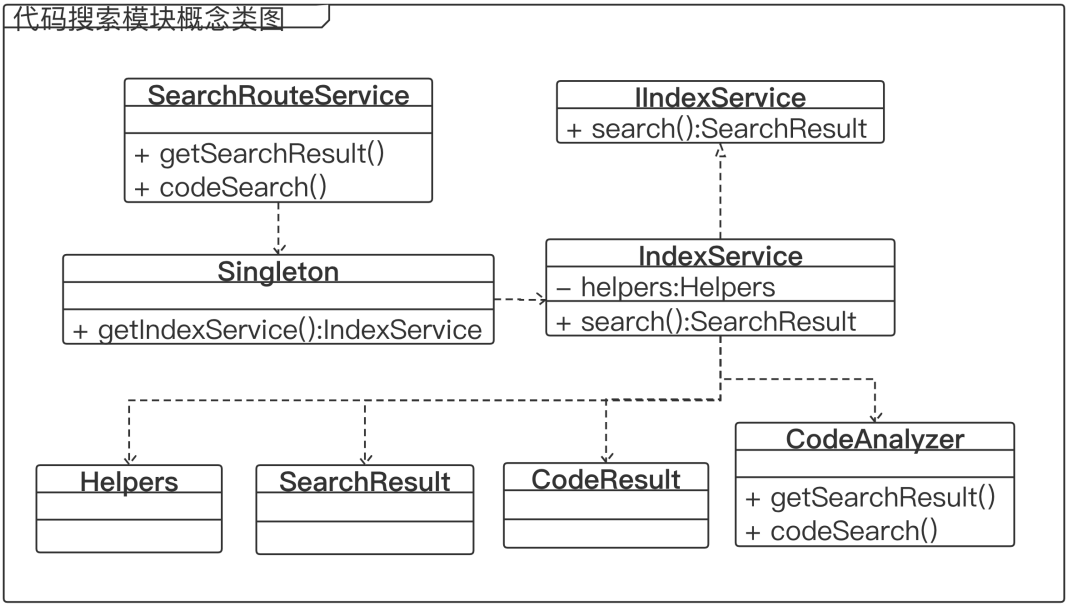

图 3.12: 代码搜索模块概念类图

代码搜索模块的概念类图见图 3.12。代码搜索模块的主要类职责为：SearchRoute-Service，负责处理请求参数和响应参数，获得搜索结果；Singleton，负责实现惰性单例模式，保证只有一个 IndexService 对象供全局访问，符合单一职责原则；IndexService，负责处理涉及到与索引有关的任务，包含搜索核心业务逻辑；CodeAnalyzer，负责实现自定义 Lucene 分析器，将字符串限制为 100 个字符；CodeResult，负责处理搜索后包含全文和元数据的结果；SearchResult，负责处理指定页码的包含语言、仓库、拥有者、代码方面的结果信息；Helpers，负责泛型帮助程序的通用处理方法。

3.4.4　特征提取模块

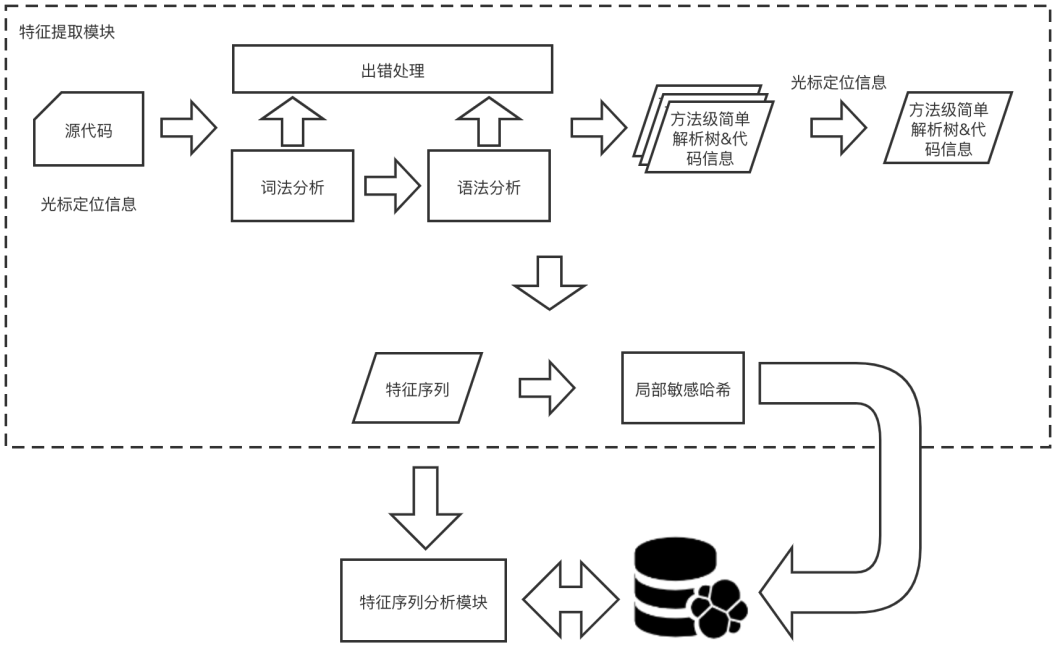

图 3.13: 特征提取模块架构图

特征提取模块的架构图如图 3.13所示。特征提取模块主要负责代码推荐的第一部分，为特征序列分析模块提供输入。在特征提取模块，需要完成包括语料库构建、源代码词法语法解析、近邻过滤等步骤。在语料库构建方面，首先抓取 github 上 star 数大于 100 的知名组织如 Alibaba、Tencent、Google 等的指定编程语言项目，接着下载并解压项目文件，之后的操作与实时处理用户代码一致。首先获取用户正在编写的源代码和光标位置，通过 ANTLR 4 进行源代码词法语法解

析，构建方法级的解析树、特征序列和相关的源代码信息。特征序列指的是文法中类名、函数名、运算符、表达式、及其他关键字信息组成的序列。接着，从获得的多个方法的解析树中，结合光标位置信息，提取用户正在关注的方法片段。当用户光标不在方法体内时，提取离光标所在行距离最近的方法体（优先位于上方的方法体）；当源代码文件不包含任何方法体时，不做任何处理。最后，为了减少源代码片段与语料库中代码比较次数，进行近邻过滤，对特征序列采用局部敏感哈希，提取前 100 的相似代码片段，作为候选代码片段，输入特征序列分析模块。

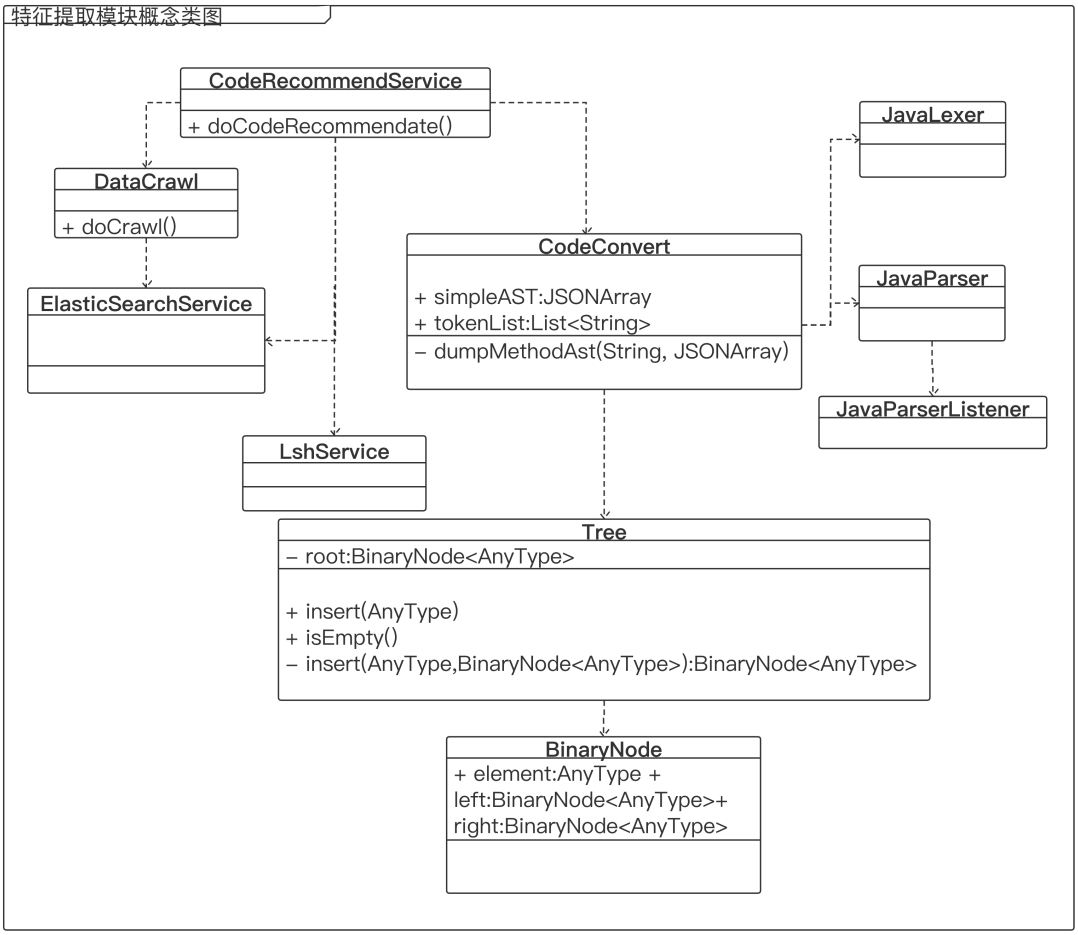

图 3.14: 特征提取模块概念类图

特征提取模块的概念类图见图 3.14。特征提取模块的主要类职责为：CodeRecommendService，负责发起代码推荐服务，获得推荐结果，构建语料库；DataCrawl，负责进行数据爬取、处理、下载、删选；ElasticSearchService，负责处理

与搜索引擎有关的所有操作，包括连接、关闭、增加、删除、查询等；LshService，负责执行局部敏感哈希方法，提供前 N 位的查询和大于阈值的查询；CodeConvert，负责从源代码提取简单信息，包括方法级别简单解析树、开始行、结束行、方法名、类名、特征序列等；JavaLexer，词法解析器，负责词法解析；JavaParser，语法解析器，负责语法解析；Tree，提供构建树的基本操作，包括插入、判断是否为空、获取深度、及树的基本遍历方法；TreeNode，提供树节点信息结构。

3.4.5　特征序列分析模块

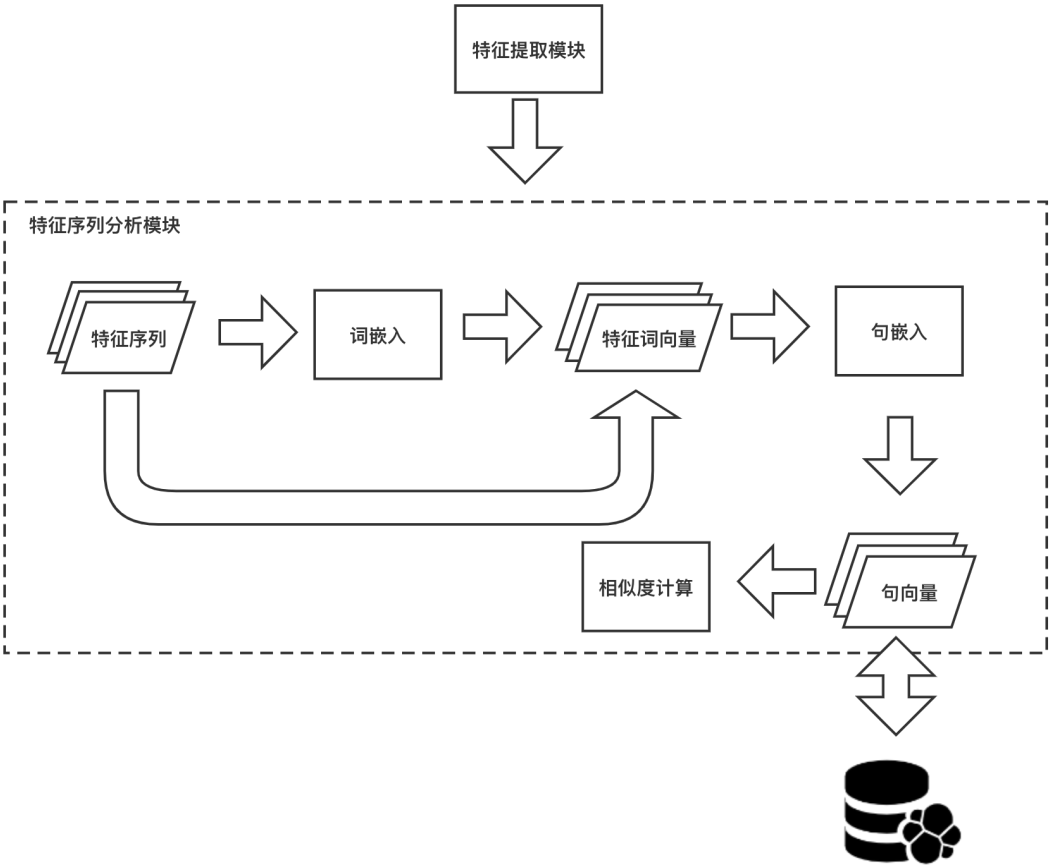

图 3.15: 特征序列分析模块架构图

特征序列分析模块的架构图如图 3.15所示。特征序列分析模块为代码推荐的第二部分，依赖特征提取模块的输出作为输入。特征序列分析模块需要首先将从特征提取模块获得的特征序列进行 FastText 词嵌入，在构建语料时，先离线计算了大量数据的词向量，获得了词向量模型，因此，在对用户当前编写的

代码获得的特征向量进行词嵌入时，只需加载已训练好的模型。接着，由获得的特征向量使用 SIF 模型计算特征序列的句向量；最后，与候选代码片段进行余弦相似性比较，将前五位相似度高于 0.9 阈值的且相似度值不同的代码片段作为最终结果，推荐给用户。推荐给用户的信息，应该包含代码片段源码，代码片段所属方法名称，代码片段所属文件名称，原项目名称及地址，以及相似度值。并按相似度值按倒序排列推荐给用户进行参考学习。

特征序列分析模块的概念类图见图 3.16。特征序列分析模块的主要类职责为：CodeRecommendService，负责发起代码推荐服务，获得推荐结果，构建语料库；WordEmbedding，负责训练和计算特征词向量；SenntenceEmbedding，负责计算句向量；Util，负责设置 SIF 加权方案中要删除的主要成分数；SIFEmbedding，负责移除纠正项，进行数据降维；WordPrepare，负责数据处理，包括切分单词，实现平滑倒词频等；SimilarityAnalysis，负责计算向量相似度，并进行相同筛查。

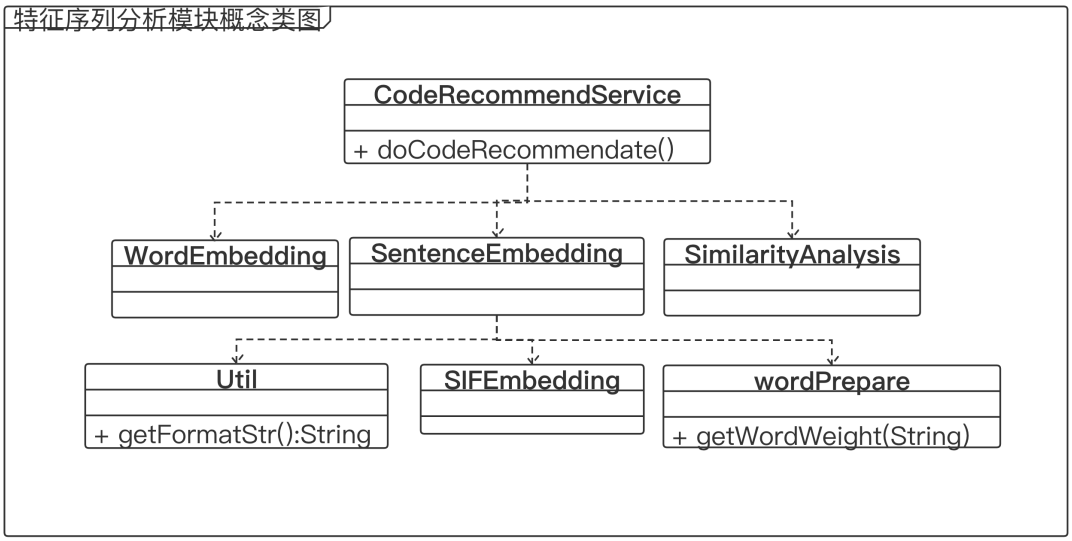

图 3.16: 特征序列分析模块概念类图

3.5　数据库实体设计

为保证存储的数据不可再分割（原子性）、皆为基础数据（原始性）、可与其他表派生出其他数据（演绎性）、相对稳定（稳定性）等特性，对系统需要进行持久化的数据进行设计，其中主要包含代码搜索模块在搜索分析时涉及到的数据库中存储 API 信息、代码信息和仓库信息的数据，以及用于代码推荐分析过程中，存储和管理在搜索引擎的方法级别代码片段结构信息。

表 3.9: 数据库 api 表

字段	类型	描述	备注
publickey	VARCHAR	公钥	主键，不为 null
privatekey	VARCHAR	私钥	不为 null
lastused	VARCHAR	最后使用时间	
data	VARCHAR	数据	

如表 3.9所示为数据库 api 表字段说明。该表主要存储 API 详细字段信息。作为 ApiResult 模型对象，用于控制层使用传值到对应的页面。公钥和私钥被创建用于访问 API。存储数据用于专门处理 API 方法，如验证，创建密钥，删除等。根据提供的公钥、HMAC 和查询字符串本身验证对 API 的请求。系统使用内存中的缓存以避免对数据库造成的超时连接或连接过多等影响。

如表 3.10所示为数据库 code 表字段说明。该表主要存储代码详细字段信息。包含代码所属仓库信息，仓库名称和仓库地址，代码所属文件信息，文件名，代码位置，哈希值，编程语言，空白行数，注释行数，总行数。作为搜索结果中 CodeResult 对象的部分有效信息，系统会将这些数据返回前端页面，展示给用户，因为这些代码信息数量大且几乎不变动，因此持久化下来使用。

表 3.10: 数据库 code 表

字段	类型	描述	备注
reponame	VARCHAR	代码所属仓库名称	
location	VARCHAR	代码位置	
filename	VARCHAR	代码所在文件名	
reponamelocation-filename	VARCHAR	代码所属仓库地址	唯一
hash	VARCHAR	哈希值	
simhash	VARCHAR	simhash 值	
languagename	VARCHAR	编程语言名	
blanklines	INTEGER	代码空白行数	
commentlines	INTEGER	代码注释行数	
lines	INTEGER	代码总行数	

如表 3.11所示为数据库 repo 表字段说明。该表主要存储仓库详细字段信

息。作为 RepoResult 模型属性，使用时，需要进行 equals 判断 name 字段是否已存在，在 UniqueRepoQueue 和其他一些地方来确保不多次添加同一信息。使用 getDirectoryName 方法对仓库名称属性进行判断，确定目录，否则它在文件系统上可能无效。

表 3.11: 数据库 repo 表

字段	类型	描述	备注
name	VARCHAR	仓库名称	主键，不为 null，唯一
scm	VARCHAR	仓库版本控制工具	默认为 git
url	VARCHAR	仓库路径	
username	VARCHAR	用户名	
password	VARCHAR	密码	
source	VARCHAR	仓库来源	
branch	VARCHAR	分支名	

表 3.12: 方法级代码片段数据结构

字段	类型	描述	备注
path	VARCHAR	所在文件路径	
class	VARCHAR	方法所属类名	
method	VARCHAR	方法名	
beginline	INTEGER	方法开始行	
endline	INTEGER	方法结束行	
ast	VARCHAR	简单解析树	保留解析树中保留字、运算符等，使用特殊符号统一代替非关键词 token 比如常量、变量名、字符 (括号、分号等)
tokenList	VARCHAR	特征序列	包含类名、函数名、运算符、表达式、及其他关键字信息组成的顺序序列，去掉无用的常量、变量名、字符 (括号、分号等）等

本系统使用 ANTLR 4 作为解析器生成工具对源代码进行分析，简化获得的

语法树，并提取了每个方法包括开始行、结束行、方法名、所属类名、特征序列等信息，具体如表 3.12。简单解析树用于还原方法代码，对于语义较弱的，例如变量名、字符等信息，进行统一替换。特征序列用于后续分析及相似性比较工作。

3.6　本章小结

本章首先对智能代码推荐系统进行了整体概述，介绍了系统开发背景和目标；其次，分析了系统功能和非功能需求，形成了系统用例图，并对每一个用例详细描述；接着，在系统需求分析的基础上，进行总体架构设计，对系统从逻辑视图、进程视图、开发视图和物理视图角度进行详细描述，并给出系统模块的划分以及描述；最后，对系统划分的编辑器模块、语言服务模块、代码搜索模块、特征提取模块、特征序列分析模块给出概要设计，包括结构设计、概念类图、通信图、数据设计等详细描述。为下一章系统的详细设计与实现打下坚实的基础。

第四章　智能代码推荐系统的详细设计与实现

4.1　编辑器模块设计

4.1.1　编辑器模块的详细设计

如图 4.1所示是编辑器模块的顺序图。用户鼠标双击文件树中的节点会触发 openNode 方法，首先判断该节点是否为文件夹。如果是文件夹，将会通过 fetchPath 方法获得文件夹下的所有节点信息。WorkspaceManager 主要获取文件内文件信息，TemporaryFileFilter 用于过滤隐藏文件，并将路径加入 WatchedPathStore，在被删除前将会持续观察变化。如果不是文件夹，即文件，将为文件在标签栏中创建标签。编辑器模块还将通过 onDidChangeModelContent 方法持续监听文件内容变化，即用户读文件的操作行为。当文件内容发生变化为 1 秒，将自动触发 saveFile 文件保存方法。通过 writeFile 方法，将最新文件内容和文件路径传递给 WorkspaceManager。WorkspaceManager 会将文件信息保存到 Workspace，更新本地磁盘内容。

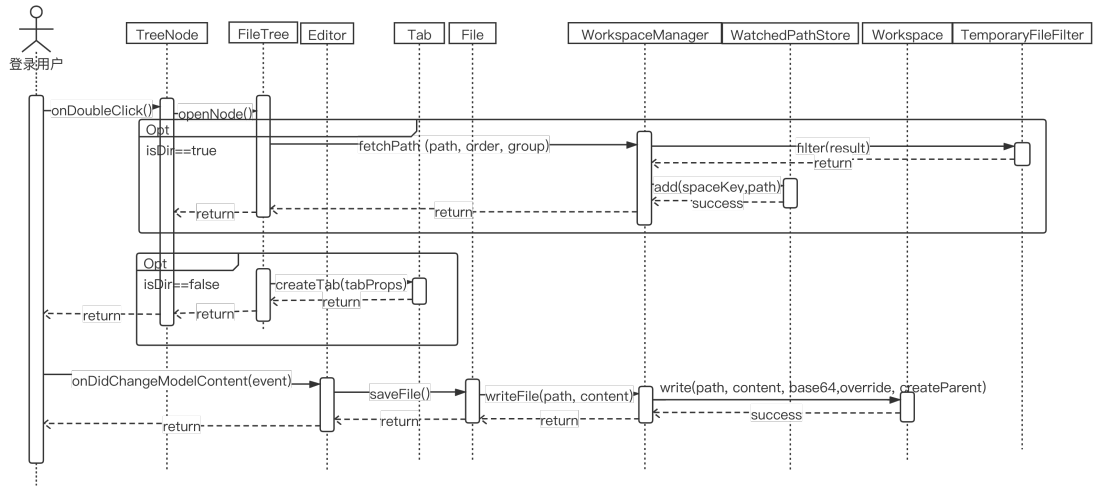

图 4.1: 编辑器模块顺序图

编辑器模块的主要类职责已在 3.4.2 节给出。编辑器模块完整类图如图 4.2所示，包含了类的属性和方法以及各类之间的调用关系。TabConatiner 为标签容器组件，当标签栏中没有标签时，需要显示 TablessCodeEditor，反之，则调用 Ed-

itorWrapper 判断编辑器类型显示相应组件。编辑器主要有五个类型，根据文件扩展名使用不同编辑器展示。HTMLEditor 用于显示 html 网页内容。ImageEditor 用于显示图片文件。UnknownEditor 用于处理无法在网页中显示内容的文件。MarkdownEditor 用于显示 Markdown 文件。以及作为默认编辑器，MEditor 用于显示编程语言文件。EditorWrapper 负责根据编辑器类型调用不同组件。MEditor 和 TablessCodeEditor 继承自 BaseCodeEditor，拥有其所有功能，包括编辑器监听鼠标移动、监听布局更新、监听主题变化等，并增加自己的新功能。TablessCodeEditor 作为无标签编辑器状态需要监听编辑器内内容变化，然后创建新标签。BseCodeEditor 类依赖 Editor 类，创建 Editor 对象，获得 Monaco Editor 实例初始化及其他编辑器的基本功能，包括对于光标位置、语言模式、主题、内容、文件等信息的获取和设置。

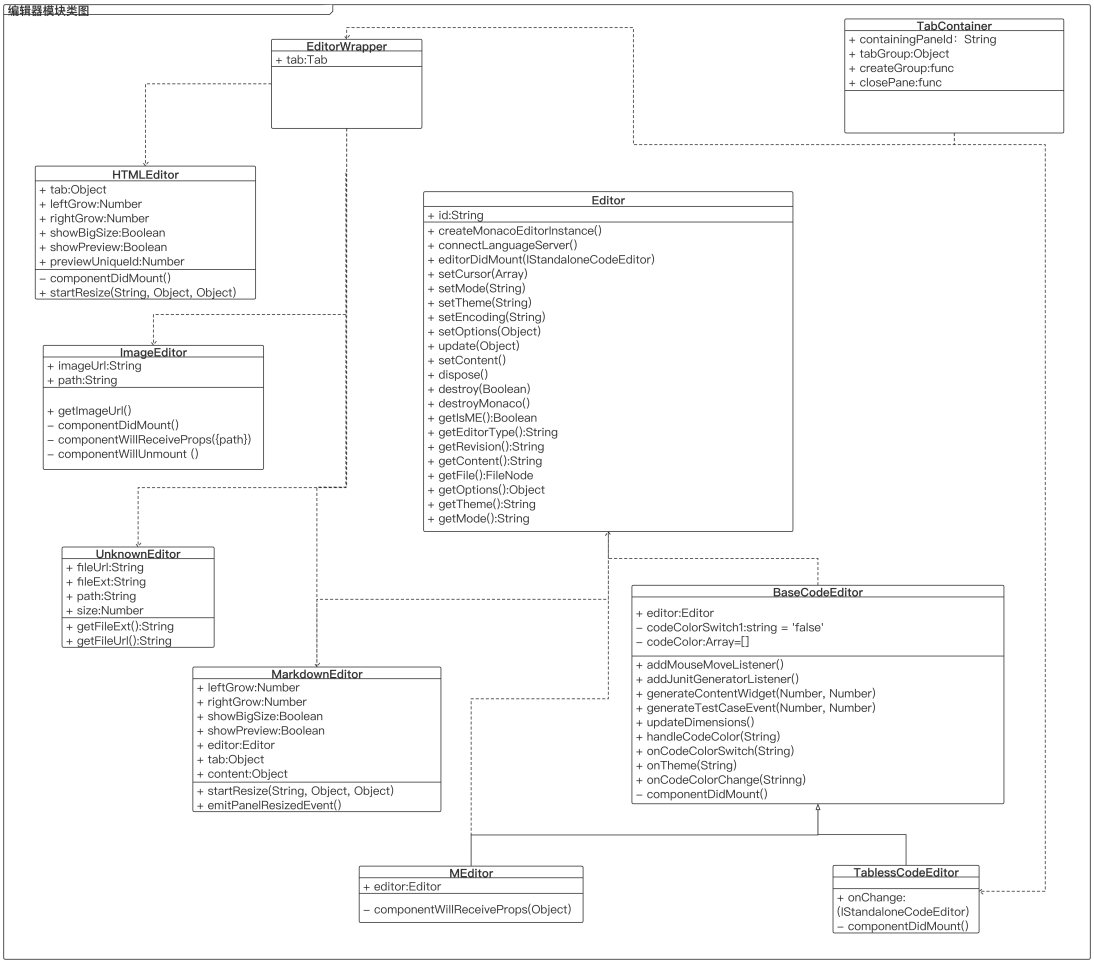

图 4.2: 编辑器模块完整的类图设计

4.1.2　编辑器模块的实现

各类型编辑器在前端使用不同的组件显示内容，Content-Type 头部一般用于标识资源的 MIME 类型，每一个文件节点拥有 contentType 类型用于记录文件扩展名，判别标签类型，当 contentType 包含 text 字段或为 application/xml 或为 application/xhtml+xml 则为 TEXT 类型，当 contentType 包含 image 字段，且等于 image/vnd.adobe.photoshop 时，为 UNKNOWN 类型，否则为 IMAGE 类型，其他情况下，默认为 UNKNOWN 类型。因此，共有 TEXT、IMAGE 和 UNKNOWN 三类标签类型。

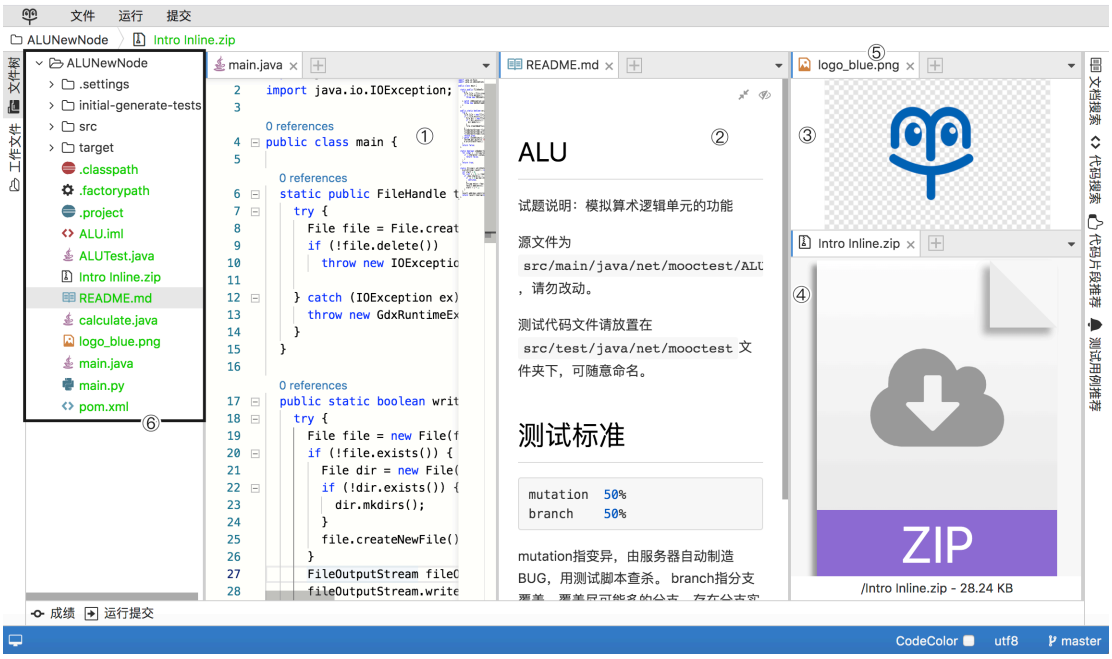

图 4.3: 各类型编辑器展示效果图

在判断编辑器类型时，结合使用标签类型进行判别。当标签类型为 IMAGE 时，使用 ImageEditor；当标签类型为 UNKNOWN 时，使用 UnknownEditor；当 contentType 为 text/html 时，使用 HTMLEditor；否则直接判断文件后缀名，当后缀名为 md 时，使用 MarkdownEditor；当后缀名为 png 或 jpg 或 jpeg 或 gif 时，使用 ImageEditor。

各类型编辑器在图 4.3中展示。1 为主要代码等相关编程语言文件编辑区域展示效果，使用 MEditor 组件。2 为 Markdown 文件内容展示效果，使用 MarkdownEditor 组件。3 为图片类文件展示效果，使用 ImageEditor 组件。4 为 ZIP 文件，使用 UnknownEditor 组件，可以点击下载到本地，还有展示 html 网页内容

的 HTMLEditor 组件。5 为文件标签展示效果，每个文件都会有一个标签，并显示文件名。6 为文件树显示区域，在文件树中双击文件，即可在对应类型的编辑器内显示文件内容。

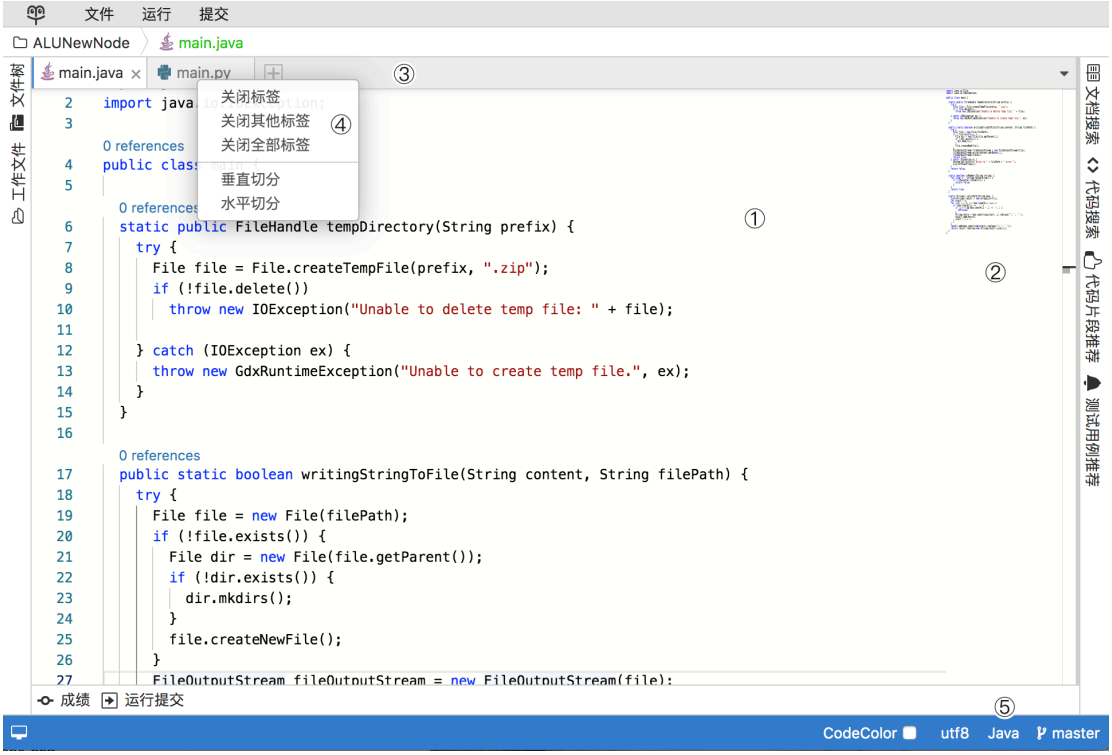

图 4.4: 编辑器主要编辑区域界面

图 4.4为编辑器的主体部分的界面，这里以最重要的的 MEditor 组件为例。1 为主要编辑区域，用户打开的文件内容会在该区域内展示，支持用户进行编辑操作。2 为代码整体预览部分，可以预览代码结构，点击可定位代码位置。3 为标签栏，所有被点击的文件都会在标签栏中显示，便于用户直接切换文件，标签显示为文件名。4 为标签提供的功能，对标签右键，可以显示该面板，提供包括关闭标签、关闭其他标签、关闭全部标签、垂直切分编辑区面板、水平切分编辑区面板的功能。5 为代码编辑语言显示，比如该页面中 main.java 文件为 Java 编程语言。

MEditor 集成了 Monaco Editor 的基本能力，并进行封装成为通用组件。如下表 4.1为 Monaco 编辑器对象数据结构。其中 content 为文件内容，被实时监听更新。当用户对文件进行编辑时，会更新此属性，因此可通过这一属性获取到最新的文件内容；cursorPosition 用于记录光标位置，包括行数 LineNumber 和列

数 Column，被实时监听更新，可用于获取最新的光标位置；theme 为编辑器主题，根据系统主题可在“vs-dark”和“vs”之间切换，为用户提供黑、白两套编辑氛围；model 为一个保存编辑状态的对象，包含语言信息，当前的编辑文本信息，标注信息等；me 为 Monaco Editor 实例，提供标准 Monaco Editor 能力，对于编辑器的其他功能操作需要在此基础上实现。

表 4.1: Monaco 编辑器对象数据结构

名称	作用	类型	操作
id	唯一标识 ID	String	
cursorPosition	光标位置信息	Object	get/set
tabId	标签唯一标识 ID	String	get/set
filePath	文件相对地址	String	get/set
theme	主题	String	get/set
options	配置信息	Object	get/set
tab	标签信息	Tab	get/set
file	文件信息	FileNode	get/set
content	文件内容	String	get/set
editorType	编辑器类型	String	get
isME	是否为 Monaco Editor 实例	Boolean	get
meDOM	编辑器文档对象模型	HTML DOM	
uri	文件绝对路径	String	
model	Monaco Editor 模型	TextModel	
me	Monaco Editor 实例	StandaloneEditor	update
modeInfo	编程语言和类型	Object	

4.2　语言服务协议模块设计

4.2.1　语言服务协议模块的详细设计

语言服务协议模块的顺序图如图 4.5所示。语言服务协议需要在工具（客户端）和服务器端之间使用，规范了语言工具和代码编辑器之间的通信。语言服务协议模块即由客户端的建立和服务器端的搭建以及它们之间的连接组成。Monaco Editor 作为客户端载体，当 Editor 初始化成功后，通过 connectLSPSocketClient 方法，发起服务器和客户端的连接。由于符合 LSP 的 Editor 编辑器可以轻松地和多个符合 LSP 的语言工具（服务器）进行通信，因此为了完成多编程语

言的提示功能的支持，主要集成包括 Java、Python 在内的平台主要编程语言的服务器。首先需要完成 LSPSocketClient 的安装、初始化，使用 install 方法，入参为编辑器实例和文件路径组成的 options 信息。MonacoServices 会主动调用 create 方法，创建 Monaco Editor 的客户端服务提供者，提供了语言特性注册、工作空间监控、指令管理、控制台窗口消息服务。安装注册完 MonacoServices 之后，LSP-SocketClient 还需要通过 createLanagugeClient 方法创建 MonacoLanguageClient，将连接 connection 传入，监控 LSP 连接情况。例如，Editor 通过 onDidChange-Content 方法监听文件内容发生变化，当发生变化时 MonacoServices 通过其注册的 doComplete 方法向语言服务器端请求，实现代码自动补全。

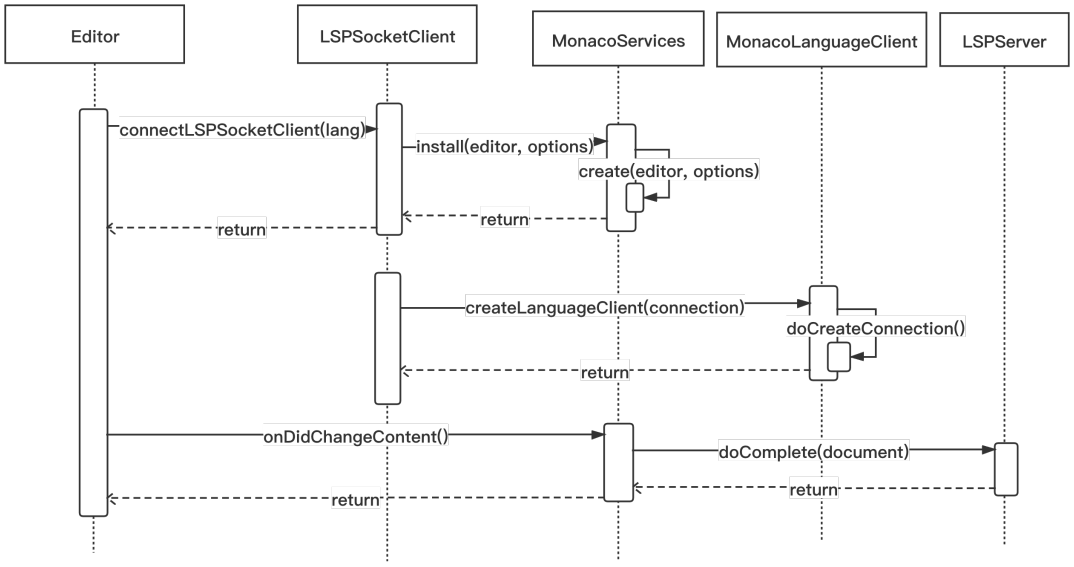

图 4.5: 语言服务协议模块顺序图

语言服务协议模块的主要类职责已在 3.5.2 节给出。语言服务协议模块完整的类图如图 4.6所示，包含了类的属性和方法以及各类之间的调用关系。LSP-SocketClient 用于创建 Monaco 语言客户端，依赖 MonacoServices 和 MonacoLanguageClient，提供和后端服务 Websocket 连接建立和断开的方法，根据编程语言确定请求地址，以及创建语言客户端的能力。MonacoServices 为 Monaco Editor 的服务提供者，用于安装注册指定编辑器实例，因此需要传入当前编辑器对象及编辑器内文件的地址。编辑器实例用于创建 MonacoCommands，提供对指令的注册。地址信息将会被用于 MonacoWorkspace 的创建，监控编辑器内工作空间的打开、关闭和变化。MonacoLanguages 提供了所有语言特性的能力注册，将

以 JSON-RPC 格式的信息与指定语言服务器进行通信。MonacoLanguageClient 为 Monaco Editor 语言客户端，可以传入客户端配置项主要包括工作空间地址、文档选择器、初始化配置信息等，并可处理客户端报错或关闭情况。其继承自 BaseLanguageClient，提供 start 方法，用于启动语言客户端。

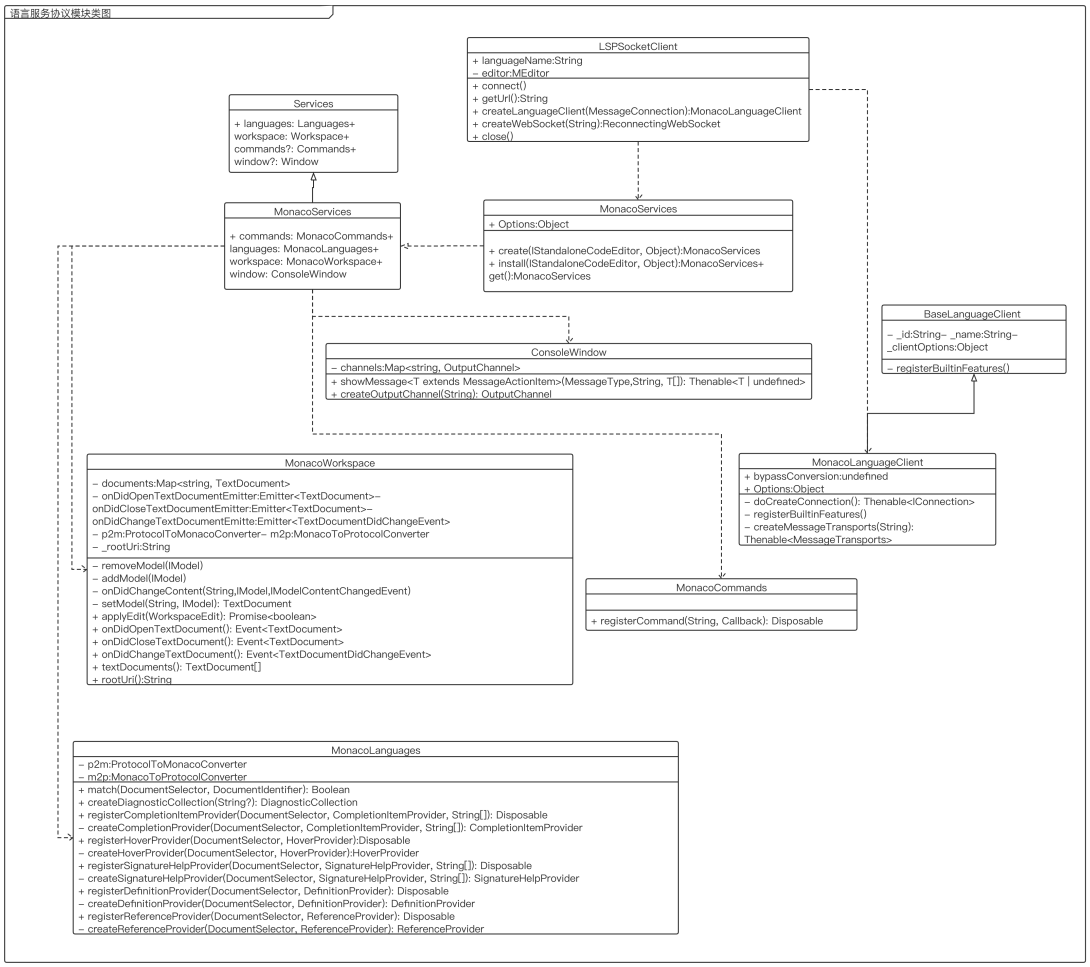

图 4.6: 语言服务协议模块完整的类图设计

4.2.2　语言服务协议模块的实现

本系统通过 WebSocket 实现 JSON-RPC 客户端和服务器之间的通信，连接建立的代码如图 4.7所示，保证客户端和服务器端的长连接。WebSocket 实现了浏览器和服务器之间的全双工通信，通信只建立在一次连接之上，可以一直保持连接状态。并且能以较少的控制开销进行通信，节省服务器资源和带宽，其持久的通信能力，也保证了更强的实时性。当 LSPSocketClient 类被实例化时，其构造方法 constructor 将会被执行，一般用于创建和初始化类创建的对象。这里

首先需要通过 MonacoServices 的 install 方法完成 Monaco 语言客户端服务的安装、注册，传入编辑器对象和文件根路径用于初始化。当 LSPSocketClient 对象的 connect 方法被调用时，将会根据当前编程语言，使用不同的 WebIDE 后台语言服务地址，并与之建立 WebSocket 连接。其连接 connection 将作为参数传入 MonacoLanguageClient 类中，用于创建语言客户端连接并启动，监听关闭请求。

```
class LSPSocketClient {
  constructor(props) {
    // install Monaco language client services  省略其他代码
    this.ms = MonacoServices.install(props.editor, { rootUri: config.fileUri });
  …}
  connect() {
   //省略获得特定语言服务器 URL 和建立 websocket 连接代码…
    listen({//当 websocket 建立后开始 listen
      webSocket,
      onConnection: connection => {
        //根据需求从 JSON-RPC 连接创建语言客户端连接并启动
        const languageClient = _self.createLanguageClient(connection);
        _self.disposable = languageClient.start();
        connection.onClose(() => {/监听  LSP ws  连接关闭
          _self.disposable.dispose();
          runInAction(() => config.lspSocketConnected = false)
        });}});
    }
   //省略断开 LSP ws 连接代码…
```

图 4.7: LSP 连接监听代码

客户端即编辑器，通过遵循语言服务协议，与 WebIDE 后台进行通信。WebIDE 后台将符合 JSON-RPC 格式的消息转发给特定语言服务器，并将响应结果返回以完成提供语言服务的功能。编辑器需要实现的包括语法高亮、自动补全、悬浮提示、引用次数查看、定义跳转、错误检测等在内的语言特性功能，由该编程语言特定的语言服务器提供。为辅助用户编程，完善用户体验并提高开发效率，为用户提供的语言特性功能如图 4.8效果。本系统支持各语言服务器拓展，因此无需局限于特定编程语言。用户在进行代码编辑时，会实时出现悬浮框对用户代码进行自动补全，并提供能够文档的描述信息查阅。当出现语法错误时，

会以红色波浪号形式进行提示，并以悬浮框告知错误原因，方便用户更正。每个方法的头一行，将显示该方法的引用次数，在文件根路径下的所有文件内调用该方法都会被记录。当用户将鼠标放置在字段上时，会出现悬浮框展示文档信息对该字段进行解释，帮助用户理解。对于调用的类中方法，或实例化对象，可以通过鼠标单击加上 Command（Mac）或 Control（Windows）来进行方法或类定义的跳转。

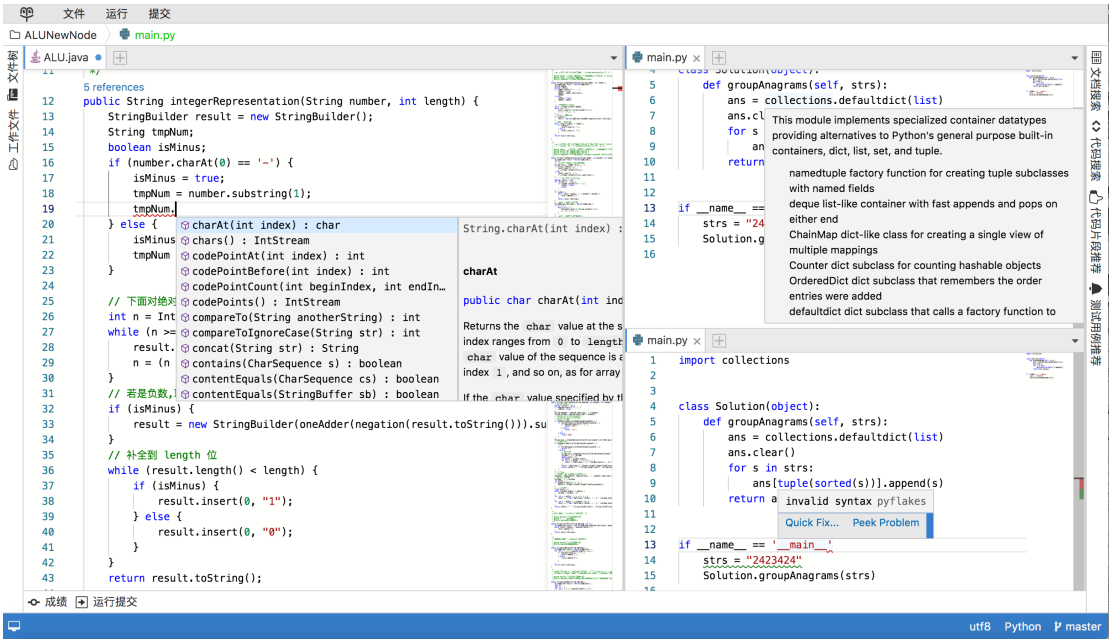

图 4.8: 编辑器语言特性界面

4.3　代码搜索模块设计

4.3.1　代码搜索模块的详细设计

代码搜索模块的顺序图如图 4.9所示。用户通过 searchCodeAction 触发搜索请求，SearchRouteService 包含搜索主逻辑，通过自调用 getSearchResult 方法获取搜索结果。获得搜索结果依赖 Singleton 的 getIndexService 方法获得 IndexService 单例，避免资源消耗。再触发 search 方法进行搜索，StatsService 用于显示基本统计数据，此处需要更新搜索总数，该数值为 int 型，最大值为 Integer.MAX_VALUE。formatQueryString 方法负责解析查询并根据 Lucene 转义，但不影响搜索操作符如 AND、OR 和 NOT。doPagingSearch 方法需要传入四个参数，分别是 Lucene 的 IndexReader 抽象类，提供访问索引的接口和 IndexSearcher，实现在单个索引阅读器上搜索，以及查询语句 query 和页码 page，实现按页搜

索主要逻辑，返回搜索代码结果。在发生索引的增删改之后，索引信息发生变化，IndexReader 需要重新读取才能保证正确性，如果发生变更，最后需要将旧的 IndexReader 资源关闭释放。

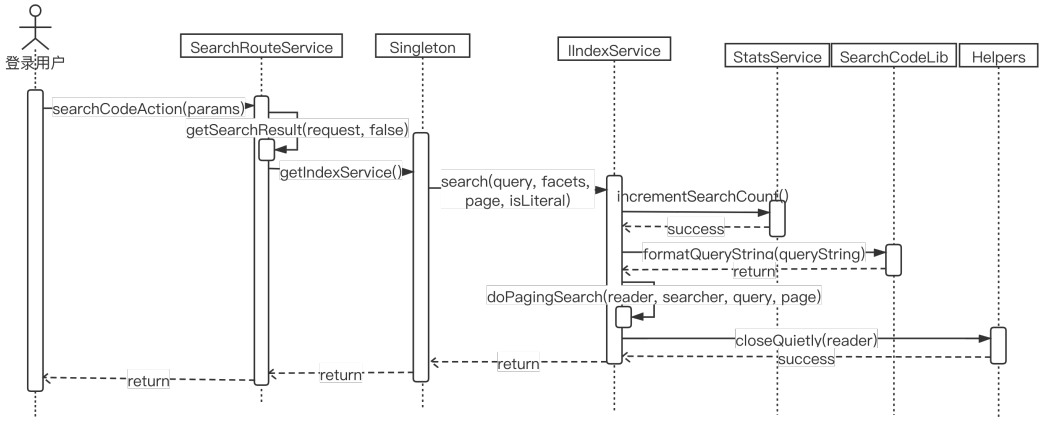

图 4.9: 代码搜索模块顺序图

代码搜索模块的主要类职责已在 3.6.1 节给出。代码搜索模块完整的类图如图 4.10所示，包含了类的属性和方法以及各类之间的调用关系。SearchRouteService 负责处理请求，通过 codeSearch 方法调用通用方法 getSearchResult()，其中含有布尔值参数 isLiteral 表示是否需要格式化查询字符串，若为 false 将会解析查询字符串并根据 Lucene 转义。Singleton 实现惰性单例模式，用于共享对象避免资源消耗，其中 getIndexService 方法可获得 IndexService 单例。IndexService 继承自 IIndexService，search 方法包含主要搜索逻辑，给定一个查询和页码，返回该搜索的匹配结果。CodeAnalyzer 继承 Lucene 的 Analyzer 分词器，并自定义限制字符串长度不超过 100 字符。Helpers 在此过程中提供最终关闭索引阅读器的能力。SearchResult 为搜索结果模型，包含了结果信息，如查询字段、页码、代码结果列表、代码拥有者聚类列表、代码语言聚类列表、代码仓库聚类列表等。其中代码结果为 CodeResult 模型类，包含代码列表、仓库名、来源地、文件名、链接地址、哈希值、编程语言、总代码行数等。

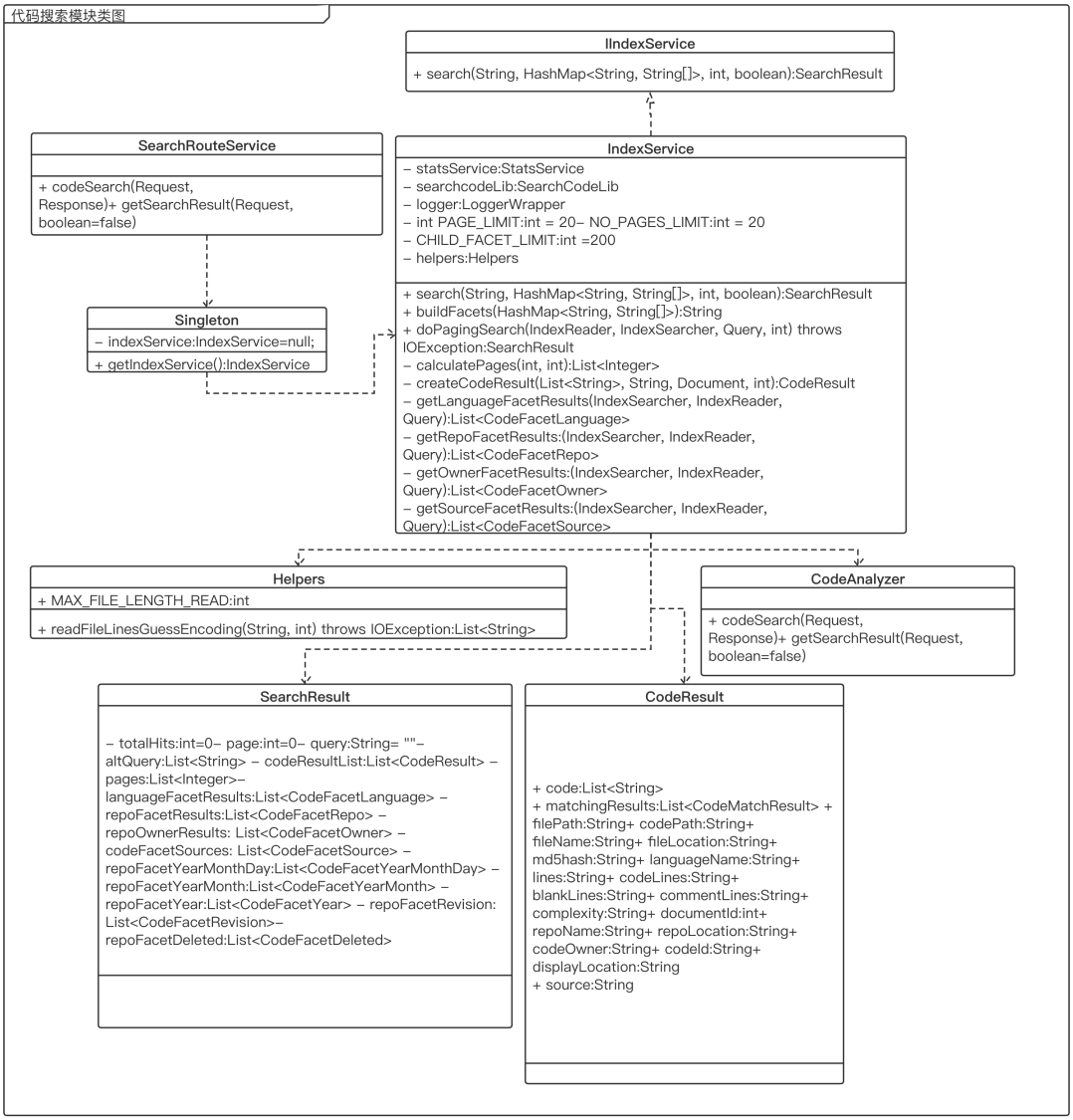

图 4.10: 代码搜索模块完整的类图设计

4.3.2　代码搜索取模块的实现

代码搜索过程通过为源代码建立索引，使得搜索代码的速度加快，并且可以按照仓库、语言和文件所有者进行筛选，以查找所需内容。支持布尔值和搜索运算符进行搜索，比如 AND、OR、NOT 等，支持多语言查询并且忽略最小化代码，支持使用通配符运算符进行搜索，比如 *，使用缓存来避免过多地访问数据库，避免连接超时或过多的情况发生。

图 4.11为搜索的主逻辑代码，SearchResult 为搜索结果模型用于存储搜索的结果信息。查询次数在每次查询时需要被增加一次，用于数据统计，避免搜索次

数过多，导致大量资源消耗。搜索使用 Lucene 搜索引擎，在获得用户的查询请求后，对查询语句进行分析，然后搜索创建的索引，最后返回结果。具体为，首先创建索引阅读器 IndexReader，并指定 Directory；其次，创建 IndexSearcher 对象，并指定 IndexReader 对象，实现在单个索引阅读器上搜索；接着，QueryParser 和 Query 将指定查询的域以及查询的字段，其中 CodeAnalyzer 为分词器，限制字符串长度不超过 100 字符；然后，按页执行查询过程；最后，关闭 IndexReader 对象并将查询结果返回。

```
@Override
    public SearchResult search(String qStr, HashMap<String, String[]> facets, int page, boolean isLiteral) {
        SearchResult searchResult = new SearchResult();
        this.statsService.incrementSearchCount();
        // 索引阅读器，提供访问索引的接口
        IndexReader indexR = null;
        if (!isLiteral) { qStr = this.searchcodeLib.formatQueryString(qStr); }
        qStr += this.buildFacets(facets);
        try {
            indexR = DirectoryReader.open(FSDirectory.open(this.INDEX_READ_LOCATION));
            // 在单个索引阅读器上搜索
            IndexSearcher indexS = new IndexSearcher(r);
            Analyzer codeAnaly = new CodeAnalyzer();
            // QueryParser 将搜索 Values.CONTENTS
            QueryParser qp = new QueryParser(Values.CONTENTS, codeAnaly);
            Query q = qp.parse(qStr);
            // 按页搜索
            searchResult = this.doPagingSearch(r, indexS, q, page);
        }
        catch (Exception ex) {…}
        finally {this.helpers.closeQuietly(indexR); }
        return searchResult;
    }
```

图 4.11: 搜索主逻辑代码

如图 4.12为 Python 的代码搜索界面和 Java 的代码搜索界面。系统主要通过获取用户正在关注的文件的，确定编程语言。在 WebIDE 界面右侧栏中提供了代码搜索的按钮，默认关闭，用户可通过按钮的点击打开或关闭代码搜索的面板。在用户进行代码编辑时，在代码字段上停留超过 3 秒，系统会获取当前文件编

程语言以及字段作为查询字段自动触发代码搜索，同时也支持用户在搜索框内输入代码片段或方法进行主动搜索。搜索结果将分页展示，提供上下页翻页按钮。每一个搜索结果包含源代码、行信息、源项目名称、项目地址链接、项目总行数、主要编程语言等，其中查询语句将通过黄色高亮在代码中标注显示，方便用户查看。希望能通过从现有源代码中获取价值来提高生产力并缩短开发时间。

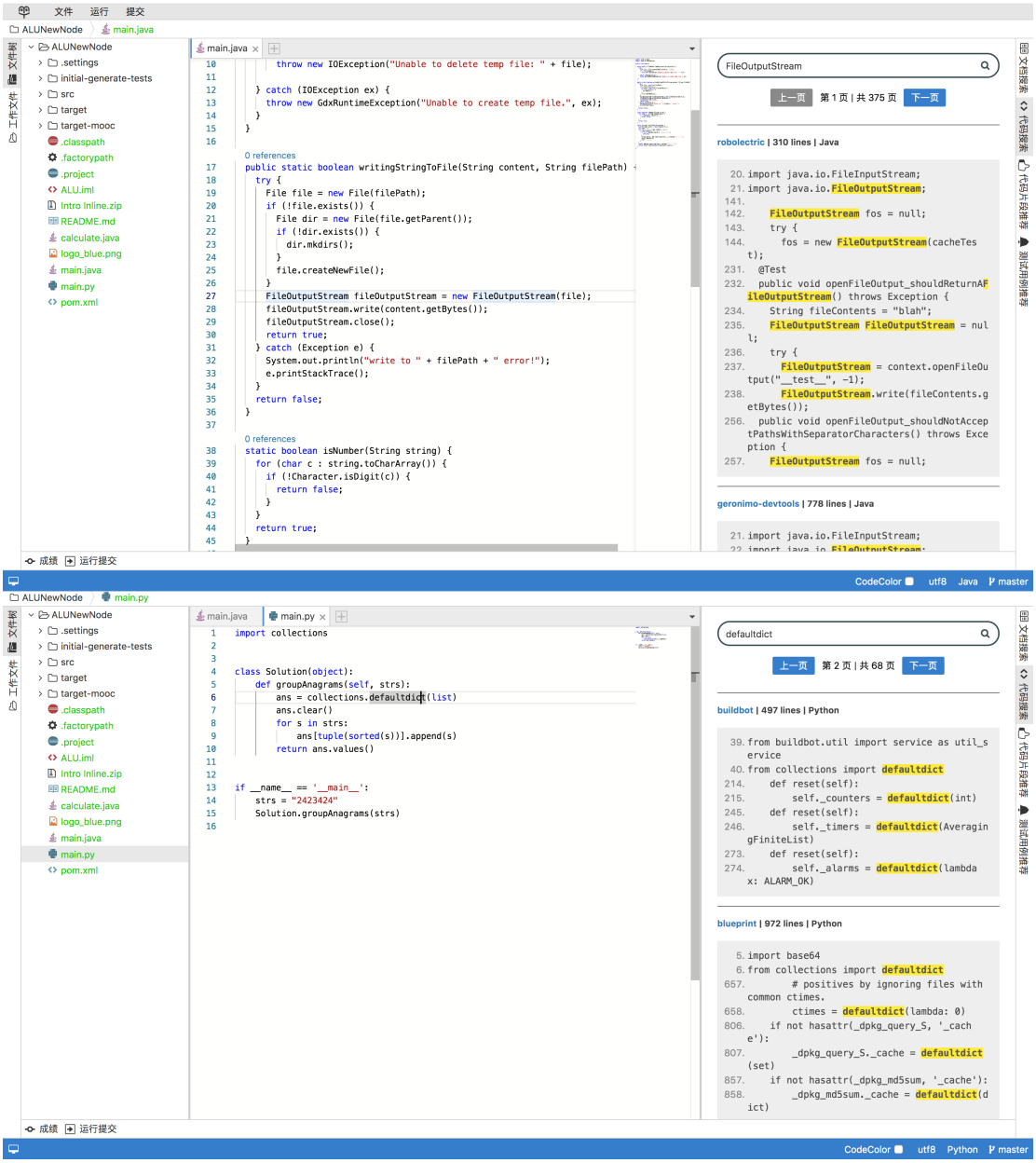

图 4.12: 编辑器代码搜索界面

4.4　特征提取模块设计

4.4.1　特征提取模块的详细设计

如图 4.13所示是特征提取模块的顺序图。特征提取模块首先由 CodeRecommendService 通过 getAntlrCodeResult 方法将获取到的用户源代码和光标位置信息。再利用 ANTLR 4 生成的词法解析器 JavaLexer，通过 getVocabulary 方法获得词汇表。利用 JavaParser 的 setErrorHandler 方法取消解析操作来响应语法错误，通过 setBuildParseTree 方法设置 _buildParseTrees 字段为 false，使得解析器在解析过程中不构造解析树，而是自己构建简单解析树。之后会返回以方法体为单位的，包含类名、方法名、方法体开始行、方法体结束行、特征序列、简单解析树的结构信息。结合光标位置信息即光标所在行信息与获得的所有方法体开始行和结束行作对比，确定需要进行推荐的源代码方法体。

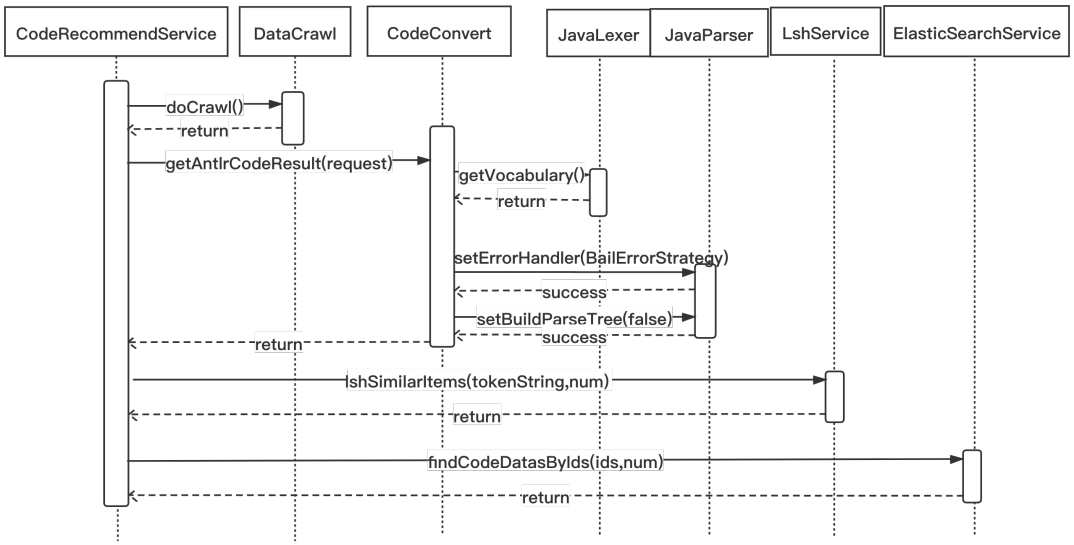

图 4.13: 特征提取模块顺序图

事先，通过特征提取和特征序列分析部分步骤，完成了包括局部敏感哈希模型、词向量模型和句向量模型在内的各语料模型的训练。在用户代码方法体被确认后，将得到的特征序列与语料库中的方法体的特征序列通过 lshSimilarItems 方法进行局部敏感哈希。选出前 N 位语料库中能够数据作为备选数据集，存储数据 index 信息便于搜索，这一步能避免与语料库中数据进行一对一对比，大大缩短推荐时间，提高效率。

最后，通过调用 ElasticSearchService 的 findCodeDataByIds 方法，在 Elastic

Search 中将这 N 个方法信息搜索出来为特征序列分析模块做准备，其中使用搜索引擎为加快查询速度。

特征提取模块的主要类职责已在 3.7.2 节给出。特征提取模块完整的类图如图 4.14所示，包含了类的属性和方法以及各类之间的调用关系。CodeRecommendService 提供构建语料库、开始代码搜索的逻辑。基于语料库的训练中在此阶段需要保存 LSH 模型，单独向外提供服务，避免每次需要重新加载模型消耗时间。其中 doCodeRecommendate 方法调用 CodeCovert 的 serializeFile 方法开始代码初步解析，解析过程依赖 JavaLexer 和 JavaParser 两个类。JavaLexer 为词法解析器，有 getVocabulary 方法用于获取词汇表，JavaParser 为语法解析器，用来取消解析操作响应语法错误及取消构建解析树的能力。调用 LshService 的 lshSimilarItems 方法将用户代码加入语料库训练好的 LSH 模型 MinHashLSHForest 中进行哈希，获得其中相似度最高的前 N 位代码片段 ID 作为候选集，之后将在这基础上进行分析比较，其中 MinHashLSHForest 支持指定 topN 查询内容，并且占用更少的空间。

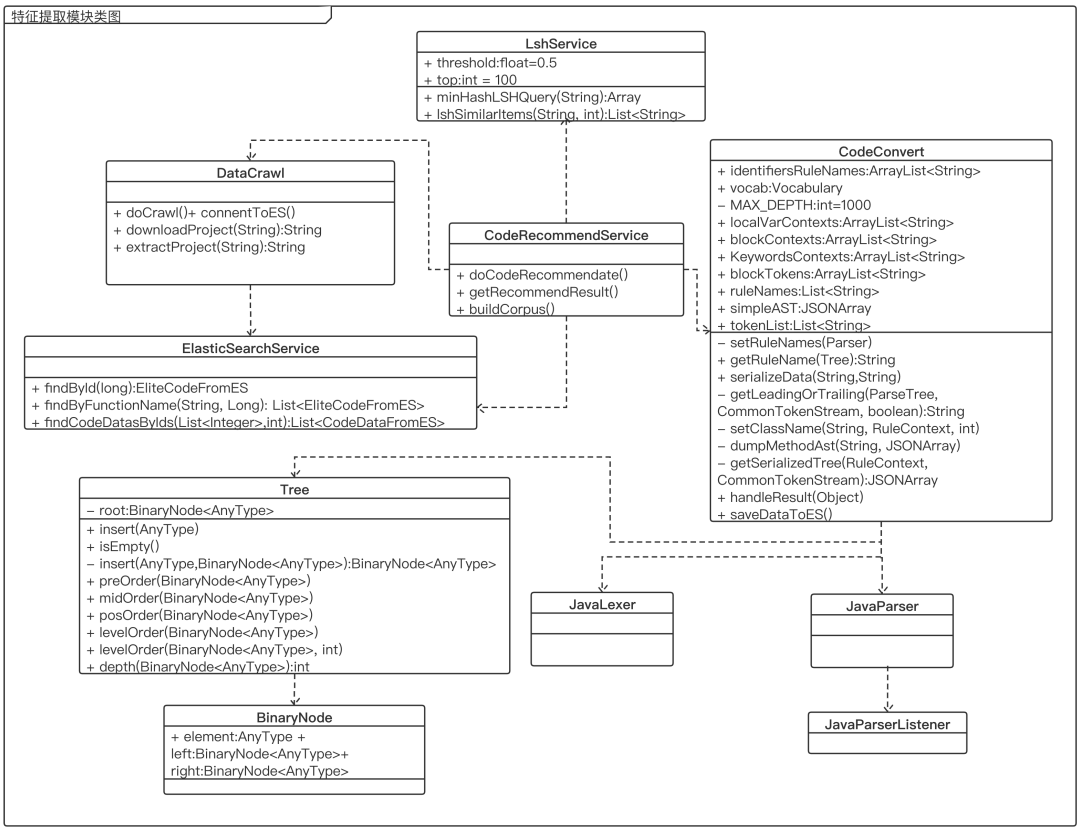

图 4.14: 特征提取模块完整的类图设计

4.4.2　特征提取模块的实现

如图 4.15为源代码特征提取的逻辑代码。对于代码推荐的请求，会获取用户文件路径和光标位置信息以及 LSH 取前 N 位数，首先，从用户文件路径处提取用户源代码，进行源码解析，通过调用 getAntlrCodeResult 获得当前用户关注方法体代码的解析数据，这一步需要使用 Antlr 4 对用源代码进行词法解析、语法解析，并按照规则判断进行代码信息的提取和结构的构建。Antlr 4 的使用需要源代码中不存在语法错误，因此对于存在语法错误，或其他情况无法正常跑完解析的源代码，进行错误处理。接着，使用预先在大量代码片段上训练好的 LSH 模型 MinHashLSHForest 对用户代码特征序列进行哈希，获得前 N 位相似度高的方法体数据作为候选集，避免了和语料中所有数据进行一对一比较。然后，通过候选集 ID 信息，在 ES 搜索引擎中将数据搜索出备用，为特征序列分析做准备。

```
    @Override
    public RecommendCodeRespVO codeRecommend(RecommendRequest recommendRequest) {
        //step.1  提取用户代码  抽象语法树
        AntlrCodeResult currentMethodCode = getAntlrCodeResult(recommendRequest);
        if(currentMethodCode == null) { return null;//  光标位置无法解析到方法}
        String tokenString = String.join(" ", currentMethodCode.getTokenList());
        //step2  计算 lsh 最相近的代码片段  获得 id
        List<String> esIds = embeddingService.lshSimilarItems(
                tokenString,
                recommendRequest.getNum());
        if(esIds == null) { return null; //没有可推荐代码片段}
        //step3 es 获取候选语料库数据
        List<CodeDataFromES> codeDataFromESList = elasticSearchService.findCodeDatasByIds(
                        esIds.stream().map(Integer::parseInt).collect(Collectors.toList()),
                        recommendRequest.getNum());
        log.info("查询结果数:{}",codeDataFromESList.size());
        if (codeDataFromESList.size() == 0) { return null; //没有可推荐代码片段}
    …
}
```

图 4.15: 源代码特征提取逻辑代码

提取用户代码进行解析步骤的核心代码如图 4.16所示。

```
public void analysisFile(String f, String startSymbol) {
        try {
                stackDepth = 0;
                Lexer lexer = new JavaLexer(new ANTLRFileStream(f));
                //循环调用 lexer 获得下一个 token，返回按次序排列的 token
                CommonTokenStream tokens = new CommonTokenStream(lexer);
                vocab = lexer.getVocabulary();
                Parser parser = new JavaParser(tokens);
                parser.setErrorHandler(new BailErrorStrategy());
                //设置所有规则名列表
                String[] ruleNames = parser != null ? parser.getRuleNames() : null;
                this.ruleNames = ruleNames != null ? Arrays.asList(ruleNames) : null;
                Method method = parser.getClass().getMethod(startSymbol);
                //跟踪与规则上下文关联的所有令牌和规则调用
                ParserRuleContext t = (ParserRuleContext) method.invoke(parser);
                parser.setBuildParseTree(false);
                //分析 token
                JSONArray tree = getSimpleTree(t, tokens);
                if (tree.length() == 2) {tree = tree.getJSONArray(1);}

        } catch (Exception e) { e.printStackTrace();}
}

private JSONArray getSimpleTree(RuleContext t, CommonTokenStream tokens) {
        ...
        int ruleIndex = ((RuleNode) t).getRuleContext().getRuleIndex();
        String thisRuleName = ruleNames.get(ruleIndex)
        if (thisRuleName.equals("methodDeclaration")) {...}//获取方法体结构
        ...//遍历 t，递归 getSimpleTree
        return ;
  }
```

图 4.16: 源代码解析核心代码

Lexer 为词法解析器，将用户代码作为输入，进行分词。CommonTokenStream 循环调用 Lexer 获取下一个 token，并返回按次序拍好的 tokens。Parser 为语法解析器，对分词后的句子按照语法规则进行顺序的处理。ParserRuleContext 用于解析的规则调用记录，包含没有存储在 RuleContext 中的当前规则的所有信息。它处理解析树子列表、任何 ATN 状态跟踪以及可用于规则指示的默认值：开始、

停止、规则索引、当前 ALT 号、当前 ATN 状态。为每个规则和语法创建的子类跟踪特定于该规则的参数、返回值、本地值和标签。因此，使用在本系统中使用其规则索引获得规则名，当规则名为 methodDeclaration 时，即记录当前各信息，以此获得类中所有方法体结构的信息。

4.5　特征序列分析模块设计

4.5.1　特征序列分析模块的详细设计

特征序列分析模块的顺序图如图 4.17所示。

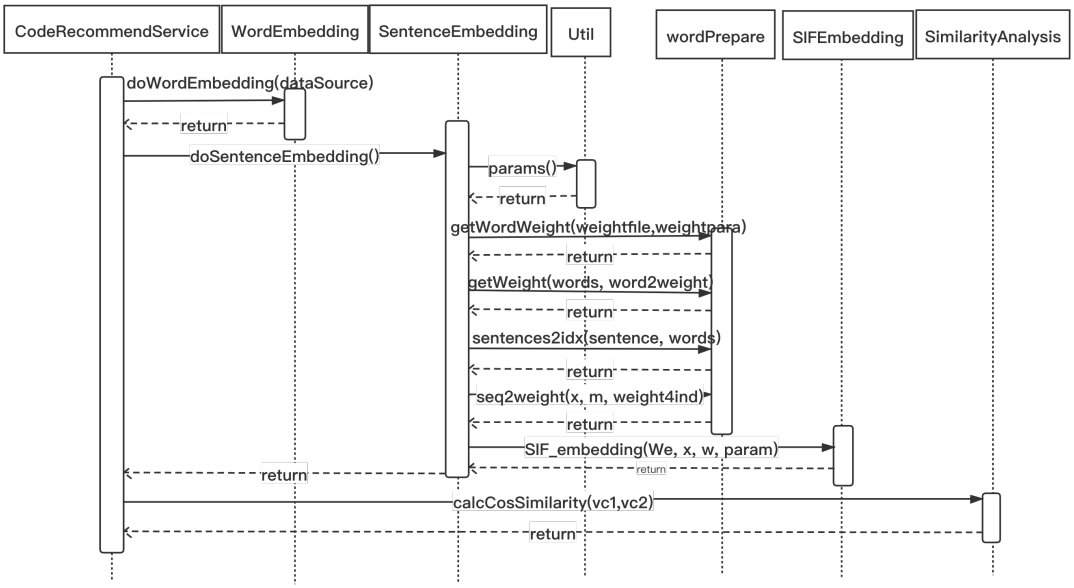

图 4.17: 特征序列分析模块顺序图

doWordsEmbedding 用于训练词向量，训练好的词向量模型将被保存在服务器硬盘上，作为单独服务对外提供。在用户代码分析过程中，只需使用训练好的词向量模型使用 doSentenceEmbedding 训练句向量。句向量的生成使用 params 方法获得需要移除的主成分个数。wordPrepare 的 getWordWeight 方法传入使用 gensim 训练出的 FastText 模型中本身就有的统计参数，实现了平滑倒词频。getWeight 方法用于记录每一个词的权重，sentences2idx 方法给定一个句子列表，输出可以输入到算法中的单词索引数组和指示该位置是否有字的二进制掩码，seq2weight 方法用于获得句子的权重。SIF_embedding 方法用加权平均计算两对句子之间的分数，去除第一主成分上的投影。calcCosSimilarity 方法需要传入用户句向量和

特征提取模块获得的候选集向量组成的向量矩阵进行余弦相似度计算，再将结果数值去重、排序，将大于阈值的前五位相似度的数据通过简单解析树还愿源代码片段返回。

特征序列分析模块的主要类职责已在 3.8.2 节给出。特征序列分析模块完整的类图如图 4.18所示，包含了类的属性和方法以及各类之间的调用关系。CodeRecommendService 包含主要分析逻辑，除了在特征提取模中依赖的类外，还需调用 WordEmbedding、SentenceEmbedding、SimilarityAnalysis 类中那个的方法完成整个推荐过程。WordEmbedding 类使用 gensim 训练出的 FastText 模型获得词向量。在训练语料库时，需要将所有特征序列作为输入进行词嵌入，获得 FastText 模型并保存，单独对外提供服务，避免在每次更新词向量模型或进行句嵌入时加载模型产生耗时，提高性能。SentenceEmbedding 类将用户代码特征系列作为输入，以 FastText 模型为基础，进行 SIF 加权计算句向量。

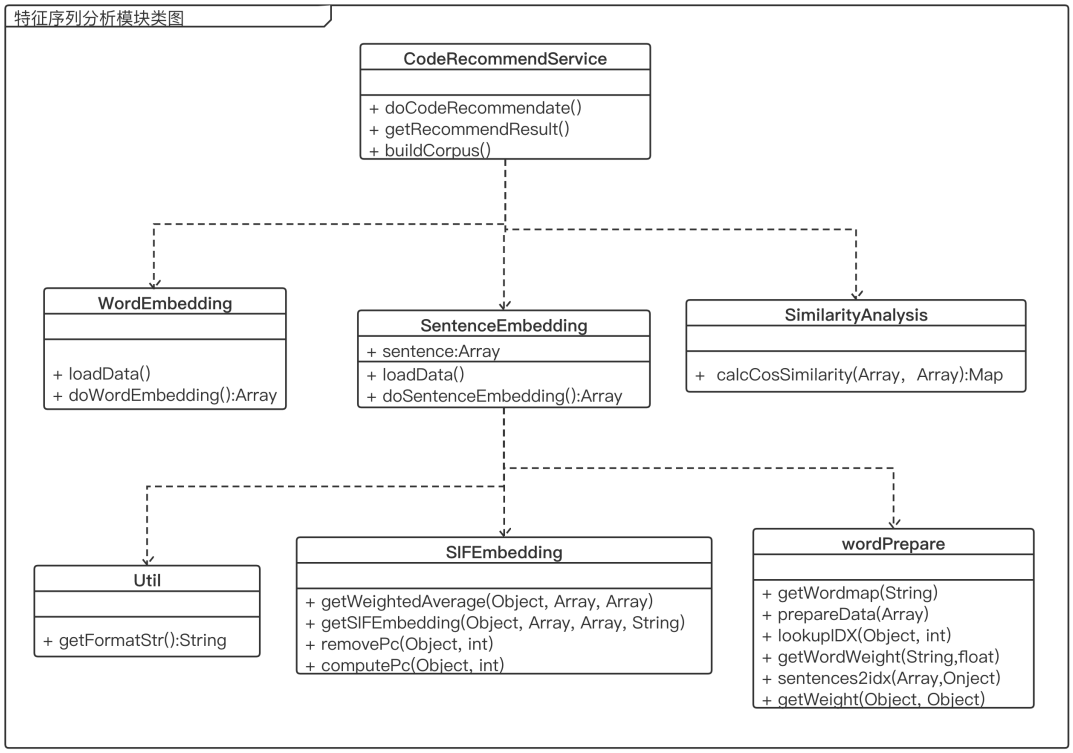

图 4.18: 特征序列分析模块完整的类图设计

其中，依赖 Util 中 getFormatStr 方法获得主成分个数。SIFEmbedding 中 getWeightedAverage 方法计算加权平均向量，computePc 方法通过 SVD 奇异值

分解训练移除项，进行数据降维。wordPrepare 中最重要的 getWordWeight 方法中实现了平滑倒词频。特征序列分析模块最后一部分是将用户代码计算出的句向量与 LSH 获得的候选集的代码片段的句向量组成的矩阵进行余弦相似度计算，并将计算结果进行去重和排序，选取其中大于阈值的前五位代码片段通过简单解析树结构进行还原代码，用于显示。

4.5.2　特征序列分析模块的实现

特征序列分析模块包含了对特征序列的主要分析过程，包括词向量分析、句向量分析以及相似性分析。如图 4.19为特征序列分析模块逻辑代码。

```
    @Override
    public RecommendCodeRespVO codeRecommend(RecommendRequest recommendRequest) {
        // …计算用户代码的句向量
        List<Double> embeddingList =    embeddingService.embedding(tokenString);
        log.info("[{}]的句向量：{}",tokenString,embeddingList);
        List<CodeDataScore> codeDataScoreList = codeDataFromESList.stream()
                .map(codeRecommendWrapper::wrapperCodeDataScore)
                .collect(Collectors.toList());
        //计算 cossimilar 返回最相似的几个
        CosSimilarResponse    cosSimilarResponse    =    embeddingService.cosinSimilar(codeDataScoreList,
embeddingList);
        //step6 score 去重，排序，取 top5
        List<CodeDataScore> codeDataScores = cosSimilarResponse.getCodeDataScoreList()
                .stream()
                .collect(Collectors.collectingAndThen(Collectors.toCollection(
                // 利用 TreeSet 的排序去重构造函数来达到去重元素的目的
                () -> new TreeSet<>(Comparator.comparing(CodeDataScore::getScore))), ArrayList::new))
                .stream()
                .sorted(Comparator.comparingDouble(CodeDataScore::getScore).reversed())
                .limit(5)
                .collect(Collectors.toList());
        log.info("排序后的成绩为：{}",codeDataScores.toString());
        // 返回 VO
        RecommendCodeRespVO    ·    recommendCodeRespVO    =
codeRecommendWrapper.wrapper(codeDataScores, codeDataFromESList, currentMethodCode);
        return recommendCodeRespVO;
    }
```

图 4.19: 特征序列分析模块逻辑代码

在构建语料库时，会预先训练词向量模型，用于计算句向量。因此，当代码推荐请求被触发时，对于用户代码会在预先训练的 FastText 词向量模型基础上进行 SIF 句向量计算，获得用户代码片段的句向量。在特征提取模块中，基于 LSH 的 MinHashLSHForest 获得候选集，将候选集中的所有代码片段的句向量构成的向量矩阵与用户代码片段句向量进行余弦相似性分析。最后将余弦相似度距离结果进行去重，去掉相同结构的候选代码片段，并按照相似度从高到低排序，取出前五名构建推荐代码对象返回。

如图 4.20为词嵌入核心代码。代码中使用 Gensim 的 FastText 模型。该模块允许从语料库中进行单词嵌入的训练，并且由于 FastText 自带的对于词典外的词条进行向量补齐，能够获得词汇外的单词向量，即使训练数据集中不存在的词，仍然能够确定这个词与某些词密切相关。并且 Gensim 还支持模型的持续训练，可以在已有训练模型的基础上进行添加和更新。代码中使用语料文件绝对路径地址加载文件来源。FastText 参数设置 size 即向量维度为 200，window 即上下文窗口大小为 5，min_count 即最小有效单词数为 1，word_ngrams 为 1 代表使用子词（n-grams）信息丰富词向量，如果为 0，则相当于 Word2Vec。build_vocab 扫描语料库构建词汇表，模型需要 corpus_total_words 才能正确管理训练率，以此获得语料中那个的单词数接着，使用 train 方法，进行语料库词向量模型的训练，并将训练得到的 FastText 模型保存，以备后用。

```
# 词向量训练
def word_embedding():
  corpus_file = datapath('/code_recommend/data/corpus_313/unlabeled_data.txt')  # absolute path to corpus
    model = FastText(size=200, window=5, min_count=1,word_ngrams=1,max_vocab_size=None)
    model.build_vocab(corpus_file=corpus_file) # scan over corpus to build the vocabulary
    total_words = model.corpus_total_words  # number of words in the corpus
    model.train(corpus_file=corpus_file, total_words=total_words, epochs=25)
    # save model 文件
    fname = get_tmpfile("/code_recommend/data/corpus_313/model/fasttext.model")
    model.save(fname)
```

图 4.20: 词嵌入核心代码

训练句向量使用 SIF 加权模型，需要依赖词向量模型，因为词向量模型每次加载的时间长，因为在服务启动时加载词向量模型一次，之后重复使用。核心

代码如图 4.21所示，SIF 加权模型对每个词向量会给出一定的权重，这个权重大小基于词频，被称为平滑倒词频。使用 Gensim 训练出的 FastText 模型，其本身就有词频及总次数等统计数据，用于句向量计算过程。getWordWeight 用户获得单词与权重的对应信息，getWeight 用于获得 id 相关的单词的权重信息。通过传入句子数组，进行句向量计算，其中 SIF_embedding 通过 SVD 奇异值分解训练出移除项用于降维。

```
# 获得句向量
fname = get_tmpfile("/code_recommend/data/corpus_313/model/fasttext.model")
model_200 = FastText.load(fname) #加载词向量模型
words = {}
for index, word in enumerate(model_200.wv.index2entity):
    words[word] = index
We = model_200.wv.vectors
weightpara = 1e-3   # SIF 加权方案中的参数，通常在[3e-5，3e-3]范围内
rmpc = 1   # 需要在 SIF 中移除的主成分数量
word2weight = word_prepare.getWordWeight(model_200.wv.vocab,weightpara)   # 单词的权重
weight2id = word_prepare.getWeight(words, word2weight)   # 第 i 位为第 i 个单词的权重

def sentence_embedding(sentence):
    # 加载 sentence
    x, m = word_prepare.sentences2idx(sentence,words)   # x 是字索引的数组，m 是指示该位置是否有字的二进制掩码
    w = word_prepare.seq2weight(x, m, weight2id)   # 获得单次权重
    # 设置参数
    param = utils.params()
    param.rmpc = rmpc
    # 获得 SIF 向量
    embedding = SIF_embedding.SIF_embedding(We, x, w, param)   # embedding[i,:] 是句子 i 的嵌入值
    return embedding
```

图 4.21: 句嵌入核心代码

代码推荐功能为了不影响用户编程行为，可由用户主动开启或关闭，在 WebIDE 界面右侧提供“代码片段推荐”按钮，用户可点击打开或关闭“代码片段推荐”面板。当面板被打开时，系统会监听用户光标位置停留情况，当停留超过 5 秒时，会触发代码推荐请求，在面板中显示代码推荐结果，包括获取到的用户代码的类名和关注的方法名，以及前五名代码片段，每一个代码片段包含代码、

文件名、项目链接、代码开始行和结束行、项目主要编程语言、以及排名和评分，评分依据相似度值在 [0,100] 区间内映射得到。界面如图 4.22所示。当用户光标所在行位置发生改变，并停留超过 5 秒时，将再次触发请求。

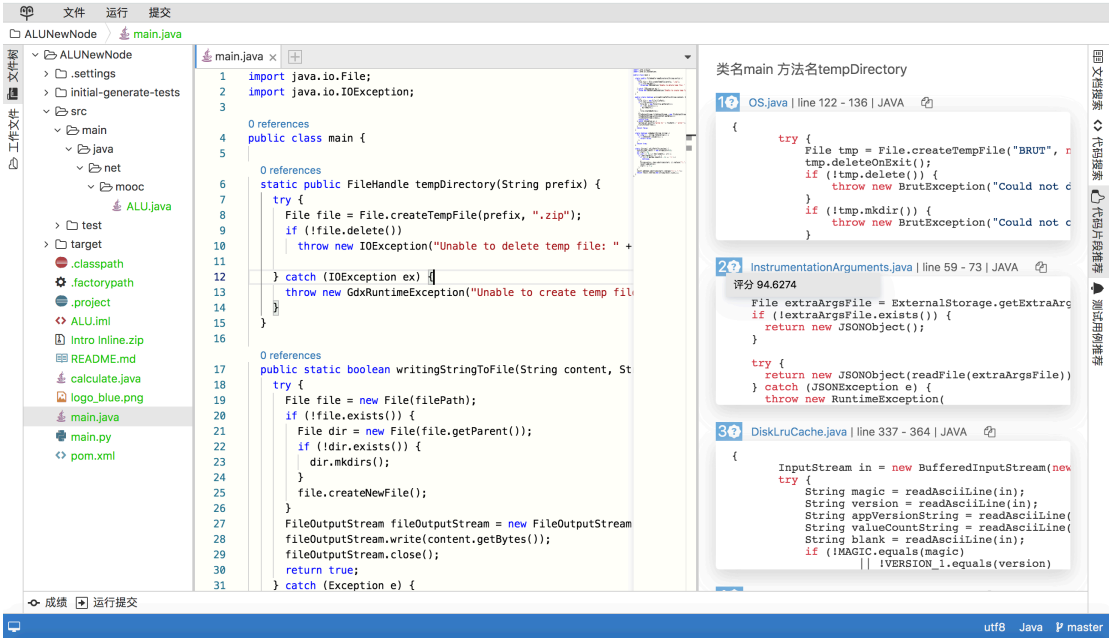

图 4.22: 编辑器代码推荐界面

4.6　本章小结

本章主要对智能代码推荐系统各模块的详细设计与实现依次进行描述。首先，对编辑器模块的顺序图和类图进行介绍和描述，并对编辑器主要功能界面和数据结构进行展示和说明；其次，对语言服务模块的顺序图和类图进行介绍和描述，并展示和说明主要实现代码和语言特性界面效果；接着，对代码搜索模块的顺序图和类图进行介绍和描述，并对主要搜索逻辑代码和主要编程语言的搜索方式和结果界面进行展示和说明；然后，对特征提取模块的顺序图和类图进行介绍和描述，并对其逻辑代码和源代码解析核心代码进行说明；最后，对特征序列分析模块的顺序图和类图进行介绍和描述，并展示和说明模块主要逻辑代码、词嵌入及句嵌入的核心代码及进行代码推荐的界面效果。对于系统的测试与分析，将在下一章中进行详细介绍和说明。

第五章　智能代码推荐系统的系统测试与案例分析

5.1　系统测试

5.1.1　测试环境

智能代码推荐系统的测试环境如表 5.1所示，主要包括运行 Chrome 浏览器的用户计算器，这里使用 Mac OS 10.13 操作系统。由于系统设计为前后端分离，因此前端系统和后端系统都单独部署。测试环境还包含部署运行基于 React 框架的前端服务所在服务器、基于 Spring Boot 框架的后端服务、推荐服务和 Elastic Search 服务所在服务器以及提供推荐所需模型的 Flask 服务所在服务器。其中 WebIDE 后端服务、推荐服务和 Flask 服务都使用 Docker 容器化。所有服务都被独立部署在阿里云 ESC 服务器上，操作系统均为 Ubuntu 16.04，主要根据各服务职责和服务器空间存量进行分配。

表 5.1: 测试环境说明

设备名称	应用	版本
用户计算机 (Mac OS 10.13)	Chrome 浏览器	Chrome 80.0（64 位）
ESC 服务器 (Ubuntu 16.04 64 位 4 核 CPU 8G 内存 500G 硬盘）	React 前端服务	React 16.8.6
ESC 服务器（Ubuntu 16.04 64 位 4 核 CPU 16G 内存 70G 硬盘）	Spring Boot 后端服务	Spring Boot 1.4.0
	Docker	Docker 17.09
	Spring Boot 推荐服务	Spring Boot 2.2.0
	Docker	Docker 17.09
	Elastic Search 搜索引擎	Elastic Search 7.5.1
ESC 服务器 (Ubuntu 16.04 64 位 8 核 CPU 16G 内存 250G 硬盘）	Flask 计算服务 Docker	Flask 1.1.1
		Docker 17.09

5.1.2　功能测试

本节将根据功能性需求分析结果和用例评估结果，对系统各项功能点进行测试，以确保系统的功能满足需求规范以及预期结果，保证系统质量。该过程主

要为黑盒测试，用户无需关心任何程序结果或特性，只需要根据需求规格设计测试用例，检查程序的功能是否能按照设计规范正确的执行。

本系统针对各功能需求点的测试用例设计在本节将进行详细描述并与预期结果进行比较。

表 5.2为系统设置的详细测试用例设计。需要验证系统设置的易用性、系统语言中文和英文是否切换方便并且结果是否正常在整个页面上对文字应用，系统主题黑色与默认选项切换是否方便并且页面包括编辑器主题色是否正常应用。

表 5.2: 系统设置测试用例

测试编号	TC1
对应用例编号	UC1
测试名称	设置系统配置项
测试目标	提供个性化选项，用户能对系统主题、语言进行更改
前置条件	用户已被认证并具有该权限
测试流程	1. 点击页面左上角“设置”按钮 2. 点击“通用”标签 3. 点击“语言”下拉框，点击“英文”或“中文”选项 4. 点击“确认”按钮 5. 点击“样式”标签 6. 点击“UI 主题”下拉框，点击“黑色”或“默认”选项
预期结果	1. 页面显示设置控制面板 2. 页面显示“通用”面板 3. 下拉框保存“英文”或“中文”选项 4. 系统内所有文字语言对应更改为英文或中文 5. 显示“样式”面板 6. 系统包括编辑器在内所有颜色风格更改为黑色或默认（白色）
测试结果	与预期结果相符

表 5.3为编辑器文件管理的详细测试用例设计。需要验证在编辑器内对文件的查看、编辑、关闭、保存等操作是否正常执行并达到预期效果。主要关注文件在被选中时，能否正确在编辑器内显示文件内容，并正确识别文件扩展名。编辑器是否能正常修改文件内容，编辑行为 1 秒后或使用组合键是否能正常保存文

件内容。标签栏中包含文件名的标签是否按照打开顺序正确排列展示。标签被关闭后，编辑器内容是否消失或被下一个标签内容替换。在文件编辑和保存之后的标签状态是否对应等。

表 5.3: 编辑器文件管理测试用例

测试编号	TC2
对应用例编号	UC2
测试名称	管理编辑器文件
测试目标	用户能在编辑器内查看、修改、保存文件内容并关闭文件
前置条件	用户已被认证并具有该权限
测试流程	1. 双击文件树中的一个文件 2. 通过键盘在编辑器内对文件内容进行编辑 3. 在编辑器内对文件进行编辑，使用键盘组合键 Mac（CMD+S）或 Windows（CTRL+S）保存文件 4. 点击标签处对某一标签的关闭按钮图标 5. 在标签区域对某一标签右键 6. 点击“关闭标签”按钮
预期结果	1. 文件内容在编辑器内显示，并形成一个包含文件名标签在标签区最后显示 2. 文件内容被更改，标签位置出现绿点标记编辑状态，并且 1 秒后自动保存，绿点消失 3. 文件内容被更改，标签位置出现绿点标记编辑状态，在 1 秒内使用组合键绿点消失 4. 编辑器内该文件内容被替换，不可见，且标签栏中无该标签 5. 出现包含“关闭标签”在内的按钮悬浮框 6. 按钮悬浮框消失，编辑器内该文件内容被替换，不可见，且标签栏中无该标签
测试结果	与预期结果相符

表 5.4为编辑器语言服务的详细测试用例设计。需要验证当代码文件内容在编辑器中显示时，是否有该编程语言的语法高亮以及每个方法前一行是否显示该方法在项目根目录下被引用次数；

表 5.4: 编辑器语言服务测试用例

测试编号	TC3
对应用例编号	UC3
测试名称	触发编辑器语言服务
测试目标	用户在编辑器内编写代码会出现语言特性服务，包括语法高亮、悬浮提示、自动补全、错误检测、跳转定义、引用次数查看
前置条件	用户已被认证并具有该权限，语言服务连接已建立
测试流程	1. 双击文件树中一个代码文件 2. 操作键盘对文件内容输入 3. 点击悬浮框内每一行最后的“i”信息图标 4. 点击或回车以选择补全信息悬浮框中的某一行 5. 鼠标停留在某一字段上，再移开 6. 在进行代码编辑时，输入错误语法 7. 将鼠标放置在波浪号位置，再移开 8. 同时对某一个方法或类等字段按下键盘 Mac（CMD）或 Windows（CTRL）及鼠标点击
预期结果	1. 代码文件内容在编辑器内显示，且所有字段已根据语法进行高亮，并且每个方法体前一行显示该方法在整个文件根路径内被引用的次数 2. 出现与已输入字段相关的补全信息悬浮框，以行为单位 3. 紧贴该行右侧出现新悬浮框，包含该行文本解释信息 4. 悬浮框中对应行信息被自动加入编辑器内对应位置，悬浮框消失 5. 出现悬浮框包含该字段解释信息，鼠标移开后，悬浮框消失 6. 编辑器语法错误区域显示红色波浪号提示 7. 出现悬浮框包含错误解释信息，鼠标移开后，悬浮框消失 8. 光标跳转到该方法或类等信息的定义位置
测试结果	与预期结果相符

当在编辑器中进行代码编辑，增加代码字段时，是否会有该代码字段的自动补全。当鼠标放置在代码字段上时，是否会出现描述该字段信息的悬浮提示。当编写的代码出现语法错误时，是否会出现包含错误描述的错误提示。当对某种方法或类的使用组合使用鼠标和键盘时，光标是否会跳转到该字段定义处。

表 5.5为搜索代码的详细测试用例设计。需要验证在代码搜索开关开启时，页面右边栏是否有代码搜索的标签。在点击代码搜索标签后，页面右侧是否会弹出新的面板。在面板的输入框中输入代码片段或方法名后点击搜索按钮，面板中是否会有结果信息显示。当输入框为空时，鼠标停留在代码字段上超过三秒，是否会将该字段填充在输入框中并有结果信息返回。当输入框内容被更改时，面板中的结果信息是否被清空。

表 5.5: 搜索代码测试用例

测试编号	TC4
对应用例编号	UC4
测试名称	搜索代码片段或方法
测试目标	用户可以在站内通过代码片段或方法的搜索，获得相关的现有源代码信息
前置条件	用户已被认证并具有该权限，且代码搜索功开关已被开启
测试流程	1. 点击页面右边栏中“代码搜索”标签 2. 在输入框内输入代码片段或方法，并点击“搜索”图标按钮 3. 在输入框为空时，鼠标在代码字段处停留超过 3 秒 4. 更改输入框内容
预期结果	1. 页面右侧弹出“代码搜索”面板 2. 在面板内显示相关代码搜索结果 3. 该字段自动填入输入框并触发搜索，结果会在面板内显示 4. 面板内之前的搜索结果被清空
测试结果	与预期结果相符

表 5.6为查看搜索代码结果的详细测试用例设计。该测试用例需要在测试用例 TC4 之后进行，需要验证代码搜索结果是否正常显示在代码搜索的面板中。搜索结果信息是否包含项目名称、项目地址、主要编程语言、总代码行数、相关

代码行及其所在行数信息，并且查询字段被高亮显示。点击项目名称时，项目地址是否在新网页标签页中打开。点击“上一页”按钮和“下一页”按钮时，结果信息是否更新并正常显示。

表 5.6: 查看搜索代码结果测试用例

测试编号	TC5
对应用例编号	UC5
测试名称	查看搜索代码结果
测试目标	通过翻页在面板中查看搜索结果，包含项目名称、项目地址、主要编程语言、总代码行数、相关代码行及其所在行数
前置条件	用户已被认证并具有该权限，搜索结果已正确返回
测试流程	1. 点击页面右侧“代码搜索”标签 2. 查看项目名称、主要编程语言、总代码行数、相关代码行及其所在行数信息是否正确显示，以查询字段是否高亮显示 3. 点击项目名称 4. 点击“上一页”按钮 5. 点击“下一页”按钮
预期结果	1. 页面右侧弹出“代码搜索”面板 2. 正确显示 3. 在新网页标签页中加载项目地址，打开源项目页面 4. 结果列表被更新为新的列表 5. 结果列表被更新为新的列表
测试结果	与预期结果相符

表 5.7为代码推荐的详细测试用例设计。需要验证在代码推荐功能开关开启的时候，页面右侧栏代码推荐标签是否正常显示。当光标在编辑器内停留超过五秒时，代码推荐面板中是否有至多五个代码片段结果显示。单个代码片段结果是否包含代码片段及其所属类名、方法开始行、方法结束行、主要编程语言信息。点击单个代码片段所属类名字段，是否在新网页标签页中显示项目地址。点击单个结果中复制标签，剪切板中是否保存了该代码片段。将鼠标放置在排序处停留，是否会悬浮显示该片段评分。

表 5.7: 代码推荐测试用例

测试编号	TC6
对应用例编号	UC6
测试名称	推荐代码片段
测试目标	用户可以在站内通过代码片段或方法的推荐，获得相关的现有源代码信息
前置条件	用户已被认证并具有该权限，且代码推荐功能开关已被开启
测试流程	1. 点击页面右边栏“代码推荐”标签按钮 2. 光标在编辑器内停留超过五秒 3. 查看“代码推荐”面板中结果信息，确认是否正确显示至多五个代码片段结果，并按照顺序显示 4. 查看单个代码片段结果信息是否包含代码片段及其所属类名、方法开始行、方法结束行、主要编程语言等信息 5. 点击单个代码片段结果信息内的代码片段所属类名 6. 点击单个代码片段结果中复制标签按钮 7. 将鼠标移动至单个代码片段的左侧序号出停留一秒
预期结果	1. 页面右侧弹出“代码推荐”面板 2. 代码推荐请求被触发，相似源代码信息在面板中显示 3. 显示正确 4. 显示正确 5. 在新网页标签页中加载项目地址，打开源项目页面 6. 提示“代码复制成功”，将代码复制到剪切板中，可以用于粘贴 7. 出现悬浮框，包含该代码片段的评分
测试结果	与预期结果相符

5.1.3　性能测试

本部分将对系统目前代码推荐语料库相关情况进行介绍，并从请求响应时长、系统负载情况等对系统性能进行数据分析。

在代码分析过程中，对特征序列进行词嵌入可以将一个所有词数量维度的高维空间嵌入到一个低维的新空间中，帮助聚集具有相似特征的对象。图 5.1为

基于上述 445787 条数据构建的 FastText 词向量模型数据，可以看出对于所有代码片段的分析，共有 447826 个单词被纳入模型进行了分析，模型的整体性较好。

图 5.1: FastText 词向量模型

图 5.2: 代码片段语料库 ES 索引

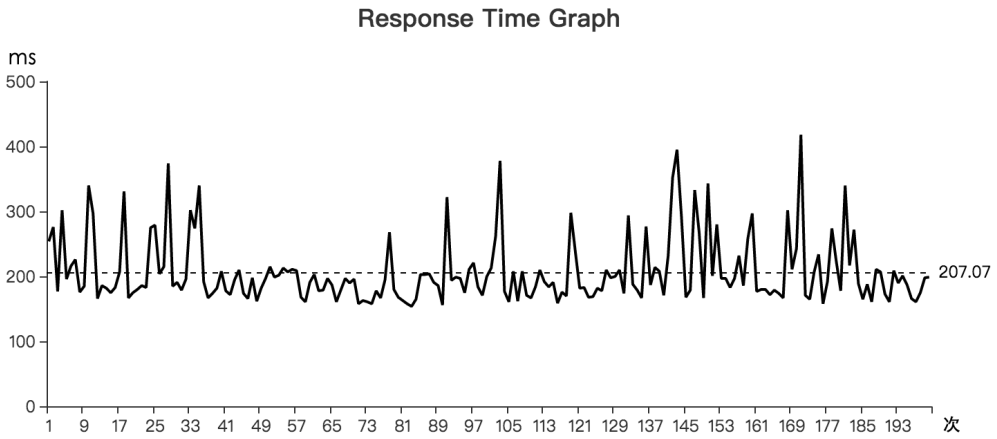

图 5.3: 代码推荐请求响应时间

代码推荐能力需要依赖大量代码片段数据作为语料，目前构建的代码片段语料库存储于 Elastic Search，以 Java 编程语言为例，其构建情况如图 5.2所示。从 Github 上挑选 707 个标星数大于 100 且主要编程语言为 Java 的开源项目，通过处理提取共 445787 条有效代码片段数据存储，占 1.3GB 大小空间，主分片和副本分片各为 1 个。丰富的数据量为代码片段的筛选和推荐提供了重要基础。

代码推荐过程经过 LSH Forest 模型筛选、FastText 词向量模型训练、以及搜索引擎中大量数据的相似性对比，计算分析时间也在不断增加。为了探究目前数据量情况下代码推荐响应时间是否满足及时响应需求，基于同一推荐请求，触发 200 次，结果数据如图 5.3所示。请求平均响应时间为 207 毫秒，响应时间中位数为 191 毫秒，并且没有错误请求发生，这个时间较好地满足及时响应需求，视为合理状况。

5.2　案例分析

5.2.1　案例说明

慕测平台是一个为高校教师和学生提供的编程类练习和考试的服务平台，该平台同对编程练习进行自动化评分和可视化展示。学生或老师用户在平台进行登录后，进入如图 5.4所示题目描述页面。页面中显示了练习的详细信息，包括练习名称、练习起止时间、练习平台、练习描述信息、练习试卷名称、出卷人、难度值、以及包含的所有题目和题目信息。题目信息会以列表形式显示题目名

称、做题状态、题目得分规则、该题在本场练习中所占分支比例、以及在线做题按钮和题目分析报告按钮。

图 5.4: 练习或考试题目界面

图 5.5: WebIDE 在线题目界面

其中，在线做题按钮即为 WebIDE 做题风格链接。本场 TDD-1 练习中主要

包含 CalculatorTDD 和 HouseManager 两道题目，都为编程练习题，各占 50% 分数，用户通过点击“在线做题”按钮可以进入 WebIDE 进行在线编程做题，智能代码推荐系统以 WebIDE 形式向用户提供服务。

WebIDE 界面如图 5.5所示。主要包含左侧试题文件树结构，上侧文件管理按钮、系统设置按钮、“运行”题目按钮、“提交”题目按钮，中间主要编辑区域，下侧题目成绩面板、运行提交日志输出面板，右侧代码搜索面板、代码片段推荐面板。其中，文件内容将在中间主要编辑区域展示，试题 CalculatorTDD 文件包内的 readme 文件包含了题目描述内容、主要功能、以及用户需要完成的编程任务。该题用户需要实现计算器类中各方法通过测试用例获得分数。

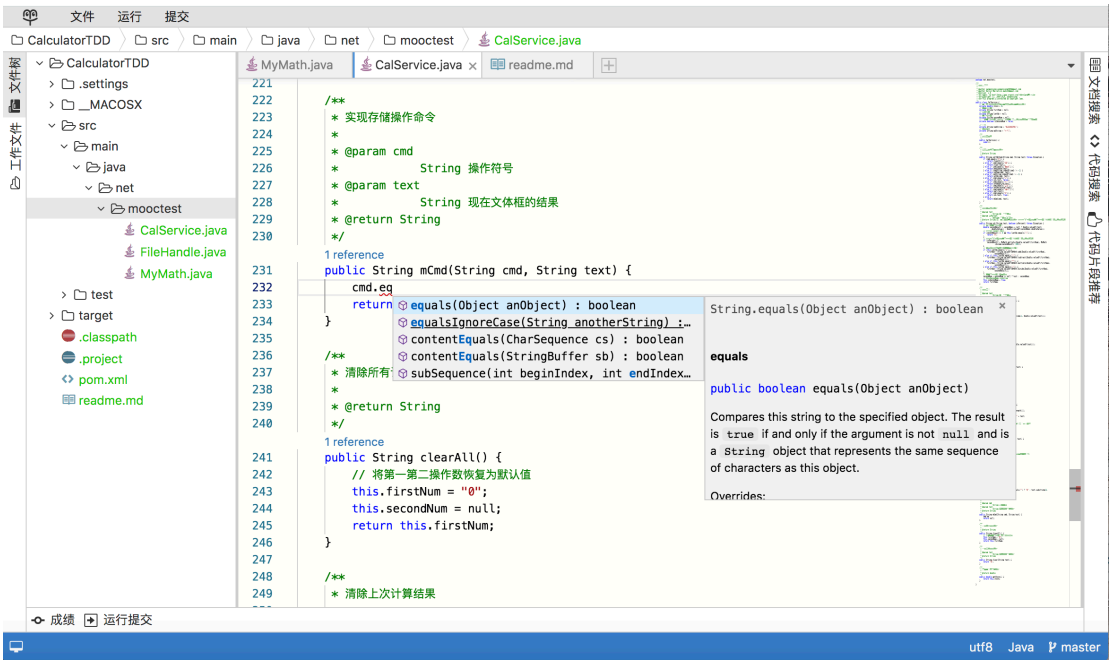

图 5.6: 题目编辑特性提示界面

题目 CalculatorTDD 中 CalService 类主要是用来处理计算器的业务逻辑，用户在编辑区域打开 CalService.java 文件进行编辑，详细如图 5.6所示。当用户尝试完成 mCmd 方法实现存储操作命令时，在文件中输入 eq。系统通过获取文件后缀确定此代码主要编程语言为 Java，与 Java 语言服务器建立连接，提供了代码自动补全功能，主要提示 JavaDoc 信息以及该文件内变量信息和说明。帮助用户快速开展代码编辑，达到和本地集成开发环境一样便捷的使用和体验。

用户点击代码片段推荐按钮，即可打开右侧功能面板。当用户编辑代码时，系统会实时监控用户操作和代码，当光标停留超过五秒时就会触发自动代码推

荐。如图 5.7，由于系统是针对方法级别的代码片段推荐，因此需要确定用户正在关注的方法体，用户光标停留在文件第十行，并且超过五秒，通过系统对文件中代码和光标位置分析，确认该光标在方法 tempDirectory 方法中。右侧代码推荐面板上侧显示了代码推荐的对象，即为 Main 类中的 tempDirectory 方法。下方显示了以相似度评分前五的代码片段，每个代码片段都包含了代码、排名、评分（相似度映射到 0 到 100 区间）、所属源文件、所属项目、方法体起止行以及主要编程语言。在本案例中，用户正在对 File 文件对象进行操作，推荐的代码片段也为相似的文件操作代码，利用现有的代码片段，帮助用户纠正或补全原代码和学习。

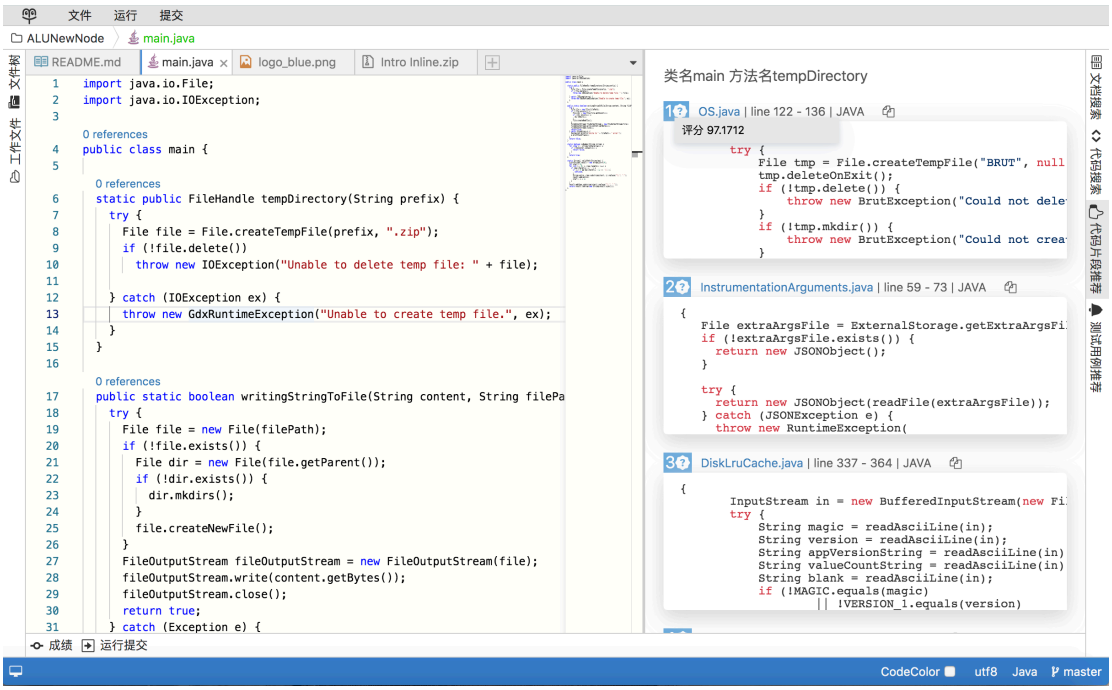

图 5.7: 题目代码片段推荐界面

用户在代码编辑时，如果对某一代码片段或方法的使用产生疑问，可以通过代码搜索功能获得该代码片段或方法使用示例，帮助学习其使用场景。如图 5.8右侧，通过点击“代码搜索”按钮展开代码搜索面板，用户可以主动在面板中上方输入框中输入代码片段或方法点击搜索图标触发搜索。系统也会实时监听用户鼠标操作，比如当用户鼠标在文件 141 行 sqrt 方法上停留超过三秒，即会自动触发代码搜索。搜索结果会在面板下方分页显示，每页包含 20 条代码片段信息，用户可以通过上下页按钮进行翻页。每个代码片段信息包含代码片段、所属项目名称、所属项目地址、总相关行数、主要编程语言，其中查询字段在代码片段

中通过黄色高亮显示更为醒目。搜索结果中显示了 sqrt 方法的使用方法，减少了用户搜索和筛选时间，除此以外用户还可以通过点击项目名称进入源项目查看更详细代码和信息。

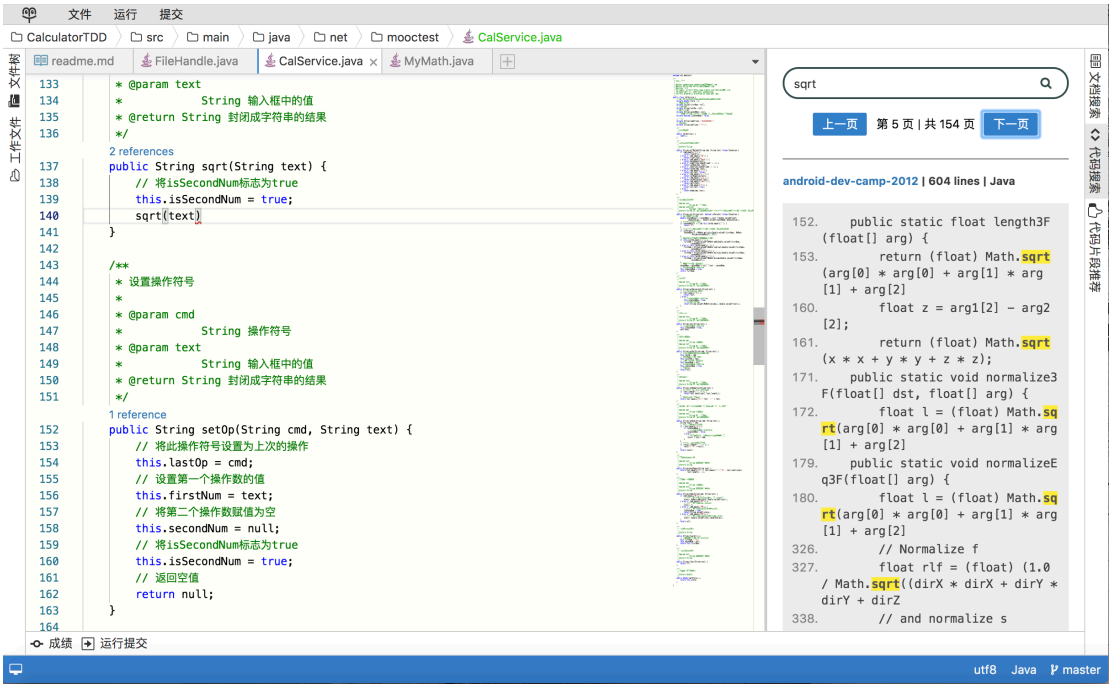

图 5.8: 题目代码搜索界面

用户通过代码编辑完成题目，通过系统中“运行”或“提交”按钮进行测试，获得编程成绩。如图 5.9所示，用户点击了“提交”按钮，将项目上传系统保存，并运行获得成绩反馈。在界面下方面板中显示了用户代码提交成功，且目前代码通过测试用例获得 90 分，这表示该用户利用智能代码推荐系统很好地完成了编程练习。

通过智能代码推荐系统，用户在进行编程练习时可以更加专注于题目，减少站外操作和代码搜索和筛选时间。通过利用现有高质量代码和方法代码示例帮助用户进行编程学习，提高编程效率，提升编程体验。并且代码片段推荐、代码搜索和代码特性推荐服务都可以通过系统进行自定义开关使用，老师可以决定在练习或考试时，使用或不使用某一服务，不为老师或学生增加额外负担。

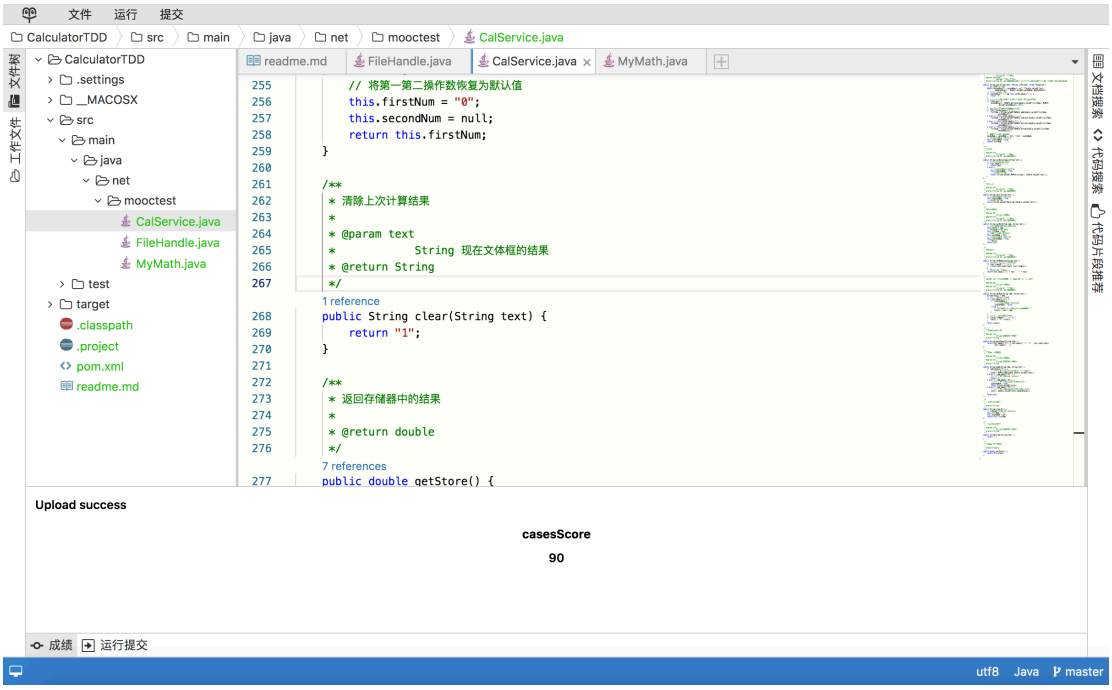

图 5.9: 题目运行结果成绩界面

5.2.2　实验调查

为调查智能代码推荐系统对程序员编程的作用和价值以及程序员对于系统的支持度与满意度，设计了一个简单实验来进行追踪调查。实验安排 14 个程序员每人完成 4 个复杂度不一的编程任务。其中，对于两个随机选择的任务，提供智能代码推荐系统供他们使用。对于另外两个任务，不提供任何智能推荐服务。在每个程序员完成所有任务后，对他们进行简短的问卷调查。每个任务都提供了要完成的功能描述，以及一些不完整的代码，参与者被要求编写代码来实现所需的功能。由于程序员水平不同且对每道题的熟悉程度也有不同，因此不考虑完成任务所消耗的时间。实验仅专注使用智能代码推荐系统进行开发的程序员得到的相关反馈。

问卷调查中包括以下问题及结果。第一问：代码片段推荐功能是否能精确推荐出相关代码？有 6 名程序员认为总能被推荐相关代码，8 名程序员认为偶尔能被推荐出相关代码。第二问：代码推荐功能推荐速度如何？有 9 名程序员任务能很及时地推荐代码，有 5 名程序员认为能够较及时地得到推荐。

将智能代码推荐系统的主要功能分为三部分，分别为代码特性推荐功能、代码搜索功能、代码片段推荐功能。对这三个功能，提出以下两点问题：

第一问：智能代码推荐服务在完成编程任务时是否有用？结果如图 5.10所

示。对于代码特性推荐功能，13 名程序员都认为总是非常有用，1 名程序员认为偶尔有用；对于代码搜索功能，8 名程序员认为总是非常有用，5 名程序员认为偶尔有用；对于代码片段推荐功能，6 名程序员认为总是非常有用，8 名程序员认为偶尔有用。

第二问：是否希望在不提供代码推荐服务的编程任务中使用到它？结果如图 5.11所示。对于代码特性推荐功能，14 名程序员都非常希望；对于代码搜索功能，9 名程序员认为总是非常希望，4 名程序员有时希望能提供；对于代码片段推荐功能，8 名程序员认为总是非常希望，6 名程序员有时希望能提供；

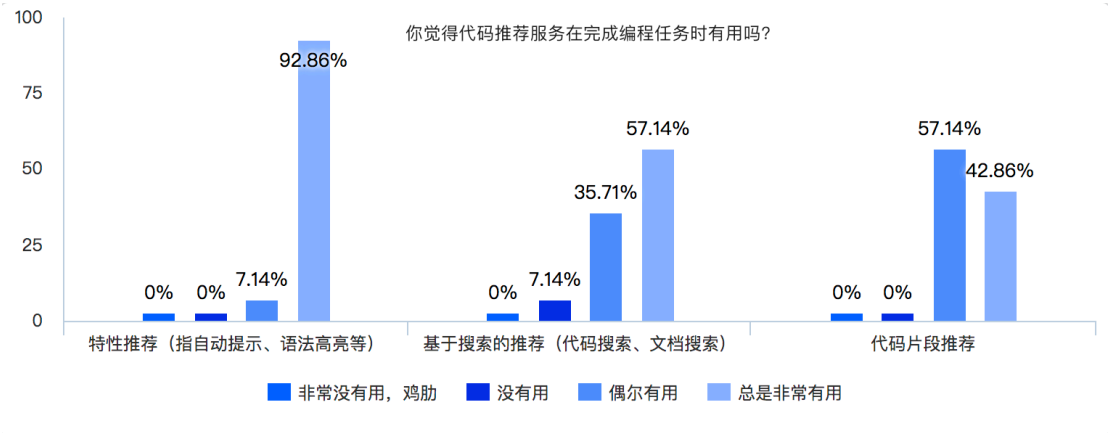

图 5.10: 系统作用度调查柱状图

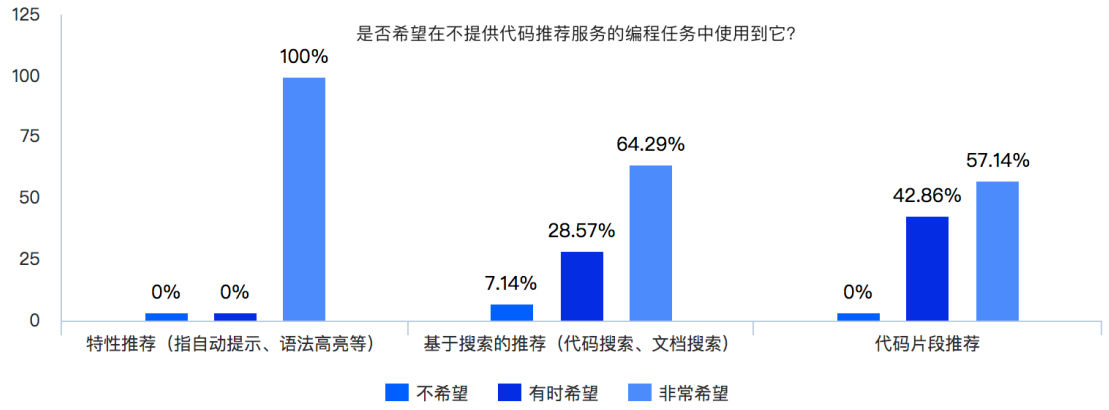

图 5.11: 系统支持度调查柱状图

综合以上调查结果可以看出，智能代码推荐系统所提供的服务，对于平均

97.62% 的用户能起到了一定的帮助作用。有平均 73.81% 的用户非常希望在编程中使用智能代码推荐服务，这表现了系统一定的用户支持度和满意度。因此得出结论，系统显著提升了平均 82.72% 的用户的编程体验。

通过案例和实验分析得出，智能代码推荐系统在用户开发时，为用户提供了简单智能的在线编辑环境和及时有效的代码推荐服务，减少了用户站外搜索和筛选信息的时间消耗，提高了用户编码效率，提升了用户编程体验。

5.3　本章小结

本章对智能代码推荐系统进行了系统测试和案例分析。首先，根据系统用例和功能性需求分析结果对系统进行了功能测试。接着，对系统非功能性需求进行测试，主要针对系统功能的性能、健壮性进行测试，包括对推荐响应反馈时间的测试评估。最后，针对一场练习，对某一试题进行编程演示，通过此案例对系统提供的功能进行了解和分析。并结合实验，调查研究系统对用户编程体验的影响。经分析，系统满足了各项功能需求、并且达到了非功能测试指标，具有一定的健壮性和可靠性，服务表现较好，并完善和提升了用户体验。

第六章　总结与展望

6.1　总结

在互联网扩速发展的背景下，在浏览器中编写代码已被越来越多的人所接受。特别是在编程教学场景下，对于需要编程练习的开发者或者初学者而言，WebIDE 克服了时间，空间，环境等因素的影响，为用户提供了方便快捷的在线编程环境，并解决本地编辑器安装复杂、配置困难、插件安装和更新等问题，提高用户体验。

在开发过程中，无论是有一定经验的开发者还是编程初学者，经常存在需要参考学习或复用网络上已有代码段的情况。然而网页上的信息量庞大且分散、冗杂，用户必须通过大量的搜索和验证才能找到有价值的代码片段，这大大影响了用户的整体编程体验，并导致开发效率低下等问题。本文针对以上问题，设计开发了基于结构嵌入分析的智能代码推荐系统，以解决用户目前 WebIDE 缺少智能编程辅助、用户编程效率低下、兴趣低落等问题。

本系统使用 React 作为前端开发框架，辅助 Mobx 和 Redux 完成状态管理，编辑器以 Monaco Editor 为基础，开发出在 React 框架中易于使用的通用组件。前端使用 Webpack 进行模块管理和打包。后端使用轻量框架 Spring Boot，并实现服务 Docker 化，保证隔离性和独立性。代码推荐使用的主要模型训练使用 Flask 框架开发，提供单独服务保证 LSH 模型和 FastText 模型的一次加载和读取。LSP 服务使用 WebSocket 保持长连接，实现客户端和服务器端的全双工通信能力，除此以外其他服务以 RESTful 接口对外提供。系统实现前后端实现分离，使用 Nginx 完成负载均衡和反向代理，方便前后端分开开发和部署。系统使用 Jenkins 实现持续集成和自动化部署。系统中的简单数据使用 Mysql 进行持久化存储，大数据语料在 Elastic Search 搜索引擎中进行存储和管理，便于搜索，提高代码推荐效率。

目前系统已上线投入使用，通过简单易用的界面和交互设计帮助用户快速专注于编程开发学习。系统通过遵循语言服务协议，实现了对包括 Java、Python 在内的多种编程语言的编辑器内实时自动补全、语法高亮、悬浮提示、错误检测、定义跳转、引用查看等语言特性功能。代码搜索和代码推荐功能相辅相成，为用户提供查询代码的使用信息和推荐相关代码片段信息，减少用户站外代码搜索及信息检索时间，为用户提高生产率并缩短开发时间。其中利用 LSH 局部敏感哈希技术和 Elastic Search 搜索引擎都能缩短代码分析时间，加快推荐速度，

这在一定程度上解决了目前代码推荐服务速度较慢，在工业应用场景上无法达到实时推荐的问题。该系统已帮助平台内学生用户提高编程效率和激发编程学习兴趣，受到学生的普遍欢迎。

6.2　进一步展望

本系统主要以 WebIDE 的形式展示，在用户进行代码编辑过程中能主动或被动地获取代码相关信息提示，希望能提高用户的编程效率。但是对于推荐系统还有很多问题需要纳入考虑，例如，开发者进行编程的需求往往很复杂具有多样性，并且涉及的技术领域也各有不同；开发者编程的需求有时并不明确，且较为开放多变，如何了解用户意图；推荐的代码的价值如何体现，如何让用户理解推荐选项；当用户代码为空时，如何向用户推荐等。

因此可以从以下几个方面改进：

1）推荐规模。目前系统提供的是对于方法级别代码片段的推荐功能，但有时一个方法体内的代码量很大，用户很难定位对其有价值的语句，因此如果能将进行在更细粒度的代码单元上推荐，或以此合成部分代码语句，此类级别的代码生成和推荐可能更有价值。

2）推荐内容。编程代码信息主要可分为通用 API、算法或应用逻辑等。现有的推荐方法通常只能集中于一种类型，无法兼顾通用 API 调用、算法使用、应用、业务逻辑各个方面，因此推荐内容可能较为单一，无法完全满足用户需求。除此以外，推荐的代码、数据质量仍需不断改进。对于语料库的挑选和构建仍需把关。

3）交互形式。目前系统支持用户主动的代码片段或方法名的搜索以及监听用户进行，当用户停顿超过五秒时，系统自动触发推荐。但此类的方只能完成一次的代码推荐行为，对于持续的代码编辑操作，没有很好地进行学习和理解。因此可以考虑对于用户编程意图进行持续理解、提供路径给用户提出问题，并做出解释、对于推荐的代码片段接收用户反馈并持续调整。

综上，目前系统中仍然存在着问题，有很多可以改进的地方。在推荐规模、推荐内容、以及推荐的交互形式方面都有很多改良方式，同时，由于对大量用户的使用，对系统性能和稳健性提出了很高的要求，对于系统的完善将持续进行。

参考文献

[1] 刘斌斌, 董威, 王戟, 智能化的程序搜索与构造方法综述, 软件学报 29 (8) (2018) 2180–2197.

[2] X. Gu, H. Zhang, S. Kim, Deep code search, in: Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, 2018, pp. 933–944.

[3] X. Gu, H. Zhang, D. Zhang, S. Kim, Deep API learning, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, 2016, pp. 631–642.

[4] M. Raghothaman, Y. Wei, Y. Hamadi, SWIM: synthesizing what i mean: code search and idiomatic snippet synthesis, in: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, 2016, pp. 357–367.

[5] J. Stylos, B. A. Myers, Mica: A web-search tool for finding API components and examples, in: 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), 4-8 September 2006, Brighton, UK, 2006, pp. 195–202.

[6] L. Wang, L. Fang, L. Wang, G. Li, B. Xie, F. Yang, Apiexample: An effective web search based usage example recommendation system for java apis, in: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011, 2011, pp. 592–595.

[7] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, D. Dig, API code recommendation using statistical learning from fine-grained changes, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, 2016, pp. 511–522.

[8] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. M. Al-Kofahi, T. N. Nguyen, Graph-based pattern-oriented, context-sensitive source code completion, in: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, 2012, pp. 69–79.

[9] A. T. Nguyen, T. N. Nguyen, Graph-based statistical language model for code, in: 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1, 2015, pp. 858–868.

[10] M. Allamanis, C. A. Sutton, Mining source code repositories at massive scale using language modeling, in: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013, 2013, pp. 207–216.

[11] M. Bruch, M. Monperrus, M. Mezini, Learning from examples to improve code completion systems, in: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009, 2009, pp. 213–222.

[12] R. Robbes, M. Lanza, How program history can improve code completion, in: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy, 2008, pp. 317–326.

[13] 李正, 吴敬征, 李明树, Api 使用的关键问题研究, 软件学报 029 (6) (2018) 1716–1738.

[14] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, S. Chandra, Retrieval on source code: a neural code search, in: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018, 2018, pp. 31–41.

[15] A. Hindle, E. T. Barr, Z. Su, M. Gabel, P. T. Devanbu, On the naturalness of software, in: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, 2012, pp. 837–847.

[16] C. J. Maddison, D. Tarlow, Structured generative models of natural source code, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, 2014, pp. 649–657.

[17] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, T. N. Nguyen, A statistical semantic language model for source code, in: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013, 2013, pp. 532–542.

[18] V. Raychev, M. T. Vechev, E. Yahav, Code completion with statistical language models, in: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014, 2014, pp. 419–428.

[19] Z. Tu, Z. Su, P. T. Devanbu, On the localness of software, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014, 2014, pp. 269–280.

[20] K. Krugler, Krugle code search architecture, in: Finding Source Code on the Web for Remix and Reuse, 2013, pp. 103–120.

[21] J. R. Cordy, C. K. Roy, The nicad clone detector, in: The 19th IEEE International Conference on Program Comprehension, ICPC 2011, Kingston, ON, Canada, June 22-24, 2011, 2011, pp. 219–220.

[22] L. Jiang, G. Misherghi, Z. Su, S. Glondu, DECKARD: scalable and accurate tree-based detection of code clones, in: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007, 2007, pp. 96–105.

[23] T. Kamiya, S. Kusumoto, K. Inoue, Ccfinder: A multilinguistic token-based code clone detection system for large scale source code, IEEE Trans. Software Eng. 28 (7) (2002) 654–670.

[24] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, C. V. Lopes, Sourcerercc: scaling code clone detection to big-code, in: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, 2016, pp. 1157–1168.

[25] 张幸儿, 规范抽象语法与抽象语法树的直接生成, 计算机学报 013 (012) (1990) 926–933.

[26] 刘呈龙, 贾胜颖, 张丽萍, 刘东升, 基于 ast 的代码抄袭检测方法研究, 计算机工程与设计 033 (04) (2012) 408–412.

[27] 肖丽, 校景中, 基于 antlr 的领域语言构造, in: 2011 国际信息技术与应用论坛论文集（《计算机科学》2011.7）, 2011, pp. 94–95.

[28] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, CoRR abs/1607.01759.

[29] T. Mikolov, A. Deoras, D. Povey, L. Burget, J. Cernocký, Strategies for training large scale neural network language models, in: 2011 IEEE Workshop on Automatic Speech Recognition & Understanding, ASRU 2011, Waikoloa, HI, USA, December 11-15, 2011, 2011, pp. 196–201.

[30] J. Goodman, Classes for fast maximum entropy training, in: IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2001, 7-11 May, 2001, Salt Palace Convention Center, Salt Lake City, Utah, USA, Proceedings, 2001, pp. 561–564.

[31] S. I. Wang, C. D. Manning, Baselines and bigrams: Simple, good sentiment and topic classification, in: The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 2: Short Papers, 2012, pp. 90–94.

[32] X. Zhang, Y. LeCun, Which encoding is the best for text classification in chinese, english, japanese and korean?, CoRR abs/1708.02657.

[33] A. Conneau, H. Schwenk, L. Barrault, Y. LeCun, Very deep convolutional networks for natural language processing, CoRR abs/1606.01781.

[34] X. Zhang, Y. LeCun, Text understanding from scratch, CoRR abs/1502.01710.

[35] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, 2014, pp. 1746–1751.

[36] S. Arora, Y. Liang, T. Ma, A simple but tough-to-beat baseline for sentence embeddings, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[37] J. Wieting, M. Bansal, K. Gimpel, K. Livescu, Towards universal paraphrastic sentence embeddings, in: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[38] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization, CoRR abs/1409.2329.

[39] R. C. Staudemeyer, E. R. Morris, Understanding LSTM - a tutorial into long short-term memory recurrent neural networks, CoRR abs/1909.09586.

[40] K. S. Jones, A statistical interpretation of term specificity and its application in retrieval, Journal of Documentation 60 (5) (2004) 493–502.

[41] S. Robertson, Understanding inverse document frequency: on theoretical arguments for IDF, Journal of Documentation 60 (5) (2004) 503–520.

[42] 蔡衡, 李舟军, 孙健, 李洋, 基于 lsh 的中文文本快速检索, 计算机科学 36 (8) (2009) 201–204.

[43] 於慧, 谢萍, 李士进, 冯钧, 基于多特征 lsh 索引的快速遥感图像检索, 山西大学学报 (自然科学版) (3) (2013) 37–43.

[44] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning 63 (1) (2006) 3–42.

[45] M. Bawa, T. Condie, P. Ganesan, LSH forest: self-tuning indexes for similarity search, in: Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005, 2005, pp. 651–660.

简历与科研成果

基本情况 韩奇，女，汉族，1995 年 12 月出生，江苏省兴化市人。

教育背景

2018.9 ~ 2020.6 南京大学软件学院 硕士

2014.9 ~ 2018.6 南京大学软件学院 本科

本论文相关成果

1. 陈振宇，袁阳阳，**韩奇**，徐朱峰，张馨中，程翔，房春荣，“一种面向多语言的高并发在线开发支撑方法”，申请号：2018104701920，已受理。
2. 房春荣，程翔，徐朱峰，袁阳阳，张馨中，**韩奇**，陈振宇，“一种基于静态分析的 Java 测试覆盖分析方法”，申请号：2018104701954，已受理。

致　　谢

首先感谢陈振宇老师，在研究生期间对我的学业和工作的指导和帮助。在毕业设计阶段，从选题到最终实验，陈老师不断敦促、正确引导，提供了很多建设性的意见，对论文的写作和项目的设计开发实验都提供了很大的帮助。

感谢房春荣老师和黄勇先生在项目开发过程中提供的专业意见，对项目的设计起到了重要的参考。

感谢门铎同学和袁阳阳同学作为实验组成员对于整体项目的付出与协助，相互支持鼓励，共同前进。

感谢父母在我的学习期间，给予的精神上的支持与关心。

感谢研究生期间南京大学软件学院对我的教育与培养，让我在研究生阶段不断充实自己，获得成长。

版权与原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期: 2020 年 5 月 23 日