南京大学
NANJING UNIVERSITY

研 究 生 毕 业 论 文

（申请硕士专业学位）

论 文 题 目 基于同源代码匹配的在线测试开发系统设计与实现

作 者 姓 名 袁阳阳

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 1 日

学　　号：**MF1832230**

论文答辩日期：**2020 年 5 月 23 日**

指 导 教 师：  **（签字）**

The Design and Implementation of an Online Test Development System Based on Homologous Code Matching

By

Yangyang Yuan

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：基于同源代码匹配的在线测试开发系统设计与实现

工程硕士（软件工程领域）专业 2018 级硕士生姓名：袁阳阳

指导教师（姓名、职称）：陈振宇 教授

摘 要

随着软件对生活的渗透，人们对其正确性和稳定性的要求日渐上升，软件测试的地位也愈发重要。但仍有许多开发人员缺乏基本测试技能，一些新手甚至不知道如何编写一个测试用例，部分初学者或许能够开发简单的测试，却很难在功能覆盖上达到所需标准。因此诞生了许多测试自动化生成工具来辅助测试开发，但现有工具往往存在配置复杂、耗时久、源码覆盖不直观、生成测试用例体量大难阅读等问题。简单易用、快速反馈、覆盖直观、用例可读的测试生成方法将更加有助于覆盖目标功能点，同时提高开发人员的测试水平。

本文依托慕测 WebIDE，设计并实现了一个基于同源代码匹配的在线测试开发系统。利用 LSP 协议新增多语言智能代码提示；结合 OpenClover 实现测试覆盖可视化；同时提出了一种测试自动化生成方案：首先，收集开源网站和考试平台历史数据构建测试代码语料库；然后，利用程序分析技术提取被测代码结构和文本信息；接着，基于字符串匹配、拼写校正、近义查找、程序相似分析等手段进行同源度量，最后，检索库中同源方法的测试用例并进行改造，最终为待测方法生成简洁可用、易于理解的测试代码。对于语料库未能覆盖的项目，系统集成了优化后的 Evosuite 工具，为用户提供针对目标方法的基本测试用例。

本系统主要包括智能提示、覆盖可视化、同源代码匹配和测试生成等服务模块。服务间通过 Restful、WebSocket 等方式进行通信。利用 Nginx 进行水平扩展，结合 NAS 存储实现服务器之间的数据共享和无状态性。为确保服务的可迁移性，使用 Docker 进行封装部署。通过 ElasticSearch 存储语料，提高匹配和检索效率。利用缓存和多线程并发技术，加快服务端响应速度。

本文对系统有效性进行了测试和实验评估。测试统计发现系统将单个项目的平均处理时间由 65s 降低至 20s，吞吐量大幅提高。应用实验确认了系统能够帮助用户逐步提高测试覆盖率；对比实验表明，相对于签名匹配算法，本系统的同源匹配算法能够针对更多的项目方法生成更多测试用例，且准确率达 90% 以上。综上，本系统能够快速、有效地帮助测试人员编写基本测试用例，在提高源码覆盖率的同时，方便测试人员阅读和学习。

关键词：智能提示，同源代码匹配，测试自动化生成，在线开发

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 an Online Test Development System Based on Homologous Code Matching

SPECIALIZATION: Software Engineering

POSTGRADUATE: Yangyang Yuan

MENTOR: Professor Zhenyu Chen

Abstract

With the penetration of software into life, people's requirements for correctness and stability are increasing, and software testing is becoming more and more important. But there are still many developers who lack basic testing skills. Some novices don't even know how to write a test case. Some beginners may be able to develop simple tests, but it is difficult for them to achieve the required standards in functional coverage. Therefore, many test automation generation tools were born to assist test development. However, the existing tools often have some problems such as complicated configuration, time-consuming, unintuitive source code coverage, and large test cases volume that is difficult to read. Test generation methods with high usablity, fast feedback, intuitive coverage, and readable use cases will be more helpful to cover the target function points, while improving the developers' test ability.

This article relies on MoocTest WebIDE to design and implement an online test development system based on homologous code matching. Use the LSP protocol to add multi-language intelligent code prompts. Combine OpenClover to achieve test coverage visualization. At the same time, a test automation generation scheme is proposed. First, collect historical data of open source websites and examination platforms to build a corpus of test codes; Then, use program analysis technology to extract the structure and text information of the code under test; After, based on string matching, spelling correction, near-sense search, program similarity analysis and other means to measure homology; Finally, the test cases of the homologous method in the library are searched and modified, and concise and usable test code for the method to be tested is generated. For projects that are not covered by the corpus, the system integrates the optimized Evosuite tool to provide users with basic test cases for the target method.

The system mainly includes service modules such as intelligent prompt, coverage visualization, homologous code matching and test generation. The services communicate with each other through Restful API and WebSocket. Use Nginx for horizontal expansion, combined with NAS storage to achieve data sharing and statelessness between servers. To ensure the portability of services, Docker is used for packaging and deployment. With ElasticSearch used for corpus storage, improve matching and retrieval efficiency. Use caching and multi-threaded concurrent technology to speed up the response speed of the server end.

This article tests and evaluates the effectiveness of the system. Test statistics found that the system reduced the average processing time of a single project from 65s to 20s, and the throughput was greatly improved. Application experiments confirmed that the system can help users to gradually increase test coverage. Comparative experiments show that, compared to the signature matching algorithm, The homology matching algorithm can generate more test cases for more project methods, and the accuracy rate is more than 90%. In summary, the system can quickly and effectively help testers write basic test cases. While improving the coverage of source code, it is concise enough for testers to read and learn.

Keywords: Intelligent Prompt, Homologous Code Matching, Test Automation Generation, Online Programming

目录

表 目 录

插图

第一章　引言

1.1　项目背景及意义

随着互联网技术的快速发展与应用，软件已经成为人们生活中不可或缺的一部分。但正如其他产品一样，软件制品也会出现错误和缺陷，致命的缺陷可能导致严重的用户流失，甚至造成软件的消亡。因此需要软件测试来确保其正确性和可用性。由此，诸如单元测试[1]、集成测试、系统测试、验收测试等测试活动的地位愈发重要。单元测试作为检测程序正确性的最小单位，是软件测试的基础，其效果将直接影响后续测试，最终对软件的质量产生重大影响 [1]。

然而即使意识到了软件测试的重要性，测试技能的培养在业内人员眼中的重视程度也远远不及软件开发，因此也导致许多开发人员在测试技能方面的缺失。特别是仍在高校或者刚刚毕业的学生 [2]，对于一些新手而言，甚至不知道如何开始编写第一个测试用例。部分初学者也许可以编写基本的测试，却很难在覆盖率上达到所需标准。因此各种自动生成测试用例的工具应运而生，也在开发者测试等领域得到广泛应用。

现阶段，通过市场调研和研究，尽管很多已有自动化测试生成工具正在尝试提出解决方案，但它们本身还是存在诸多问题。首先，大部分工具作为单独的软件产品需要用户自行下载和配置，部分作为插件集成在本地 IDE，因此带来对本地环境、版本兼容等约束，不够灵活和便利，增加了用户的学习成本和使用复杂度；其次，这些工具往往只展示整体的覆盖率情况，测试人员需要能够直观地看到已编写的用例覆盖了哪些源代码行以便进行下一步迭代测试；另外，它们大多是对整个项目工程中的开发代码进行测试用例生成，耗时良久且生成的用例众多，难以阅读和学习，不利于测试人员技能的增长和进步。因此，能够方便灵活地针对目标代码生成简洁而准确的测试用例，同时直观地展示这些用例对源代码的覆盖情况十分必要。

本论文依托慕测 WebIDE，设计并实现了一个基于同源代码匹配的在线测试开发系统。作为线上平台，用户只需打开浏览器即可编写代码，无任何下载和配置成本。系统在原先 WebIDE 基础上利用 LSP 提供了多语言的智能代码提示，最大程度减轻用户记忆难度。同时将测试用例覆盖情况通过代码行着色直观展示给测试人员，渐进式结果反馈有助于开发人员清晰判定每个用例的作用。更关

[1] https://zh.wikipedia.org/wiki/单元测试

键的是，系统通过收集和分析历史数据构建测试代码语料库，利用拼写校正、语义相似度分析、程序相似匹配等手段进行同源匹配，最终为用户生成了简洁可用的方法级测试代码。对于语料库未覆盖的项目，也可通过集成改造后的 Evosuite 工具，为用户提供针对待测方法的基础测试用例。本系统力求为测试新手和初学者提供简洁可用的测试参考样例，提高源码覆盖率的同时，降低其学习成本，辅助测试技能增长。目前系统已投入实际应用，获得了较为正面的反馈。

1.2　国内外研究现状

1.2.1　在线编程系统研究现状

随着 Web 技术的发展，许多公司和团队也开始着手开发自己的 WebIDE 并投入市场使用，通过对这些项目和产品的学习和研究，有助于丰富和完善本系统，同时体现本系统的独特性。

目前应用较为广泛的 WebIDE 包括腾讯与 Coding 团队合作的 Cloud Studio[2]、Eclipse 团队推出的 Eclipse Che[3]和 THEIA[4]、阿里云的 WebIDE[5]等。这些产品均具有与本地集成开发环境较为类似的功能，例如文件系统、语法高亮、代码补全、编译部署等等。其目标是实现云端开发，将软件工程师与本地环境解绑。此外，Cloud Studio 和 Collabode [3] 提供了协作编程 [4] 能力。但由于它们对资源的要求往往较高，而普通开发人员只能够免费使用很少的资源，且相对本地 IDE 而言暂时还无法做到完全的功能覆盖，同时受限于系统稳定性等因素导致难以投入大规模使用当中。还有一些 WebIDE 尝试在普通编程功能的基础上加入代码的推荐和教学工作，希望以此培养入门级编程新手的开发能力。例如 CODELAB [5]，在 WebIDE 的基础上侧重于简短的代码示例，以帮助学生理解相关编程概念和方法。

尽管存在许多本地 IDE 测试插件，但将测试相关功能与在线编程环境结合的实例并不常见。Parlante 的 CodingBat[6]采用了一种测试驱动[7]方法来指导学生编程，但事实上学生并不在该系统内编写测试。网站通过在题目描述中提供测试数据帮助学生理解编程目的，且 CodingBat 仅限于一组非常小型的集中练习，不具有普适性。Thomas 等人提出一种 WebIDE LAB [6]，通过使用小的迭代示例并在系统引入锁定步骤来使学生专注于测试驱动学习，利用强制手段要求学生

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

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

[4]https://theia-ide.org/

[5]https://ide.fc.aliyun.com/

[6]https://codingbat.com/java

[7]https://zh.wikipedia.org/zh-hans/测试驱动开发

编写用例。这在一定程度上有助于培养新手单元测试的能力。但系统并不具备测试编写教学功能，对于测试初学者而言仍然具有一定门槛。

1.2.2　测试自动化生成研究现状

测试自动化生成和推荐是近年来的热门课题，由此衍生了诸多工具和算法。

Google 的 CodePro Analytix[8]曾是较为流行的基于 Eclipse 的单元测试生成工具。它声称能够从多个维度考量源代码，实现代码评审、设计模式检查、测试自动化生成等多项功能。但这一工具并未开源，且已数年未有更新和维护，导致其使用者寥寥无几。

Fraser 等人提出的 EvoSuite [7] 是目前使用相对广泛的一种 Junit 单元测试用例生成工具，可以作为独立的工具或 IDE 插件使用。它能够直接生成可执行的测试用例，支持多种覆盖准则，对于体量较小的项目具有相当不错的效果。且作为一款开源工具[9]，项目维护和答疑仍然在不断更新中。但由于其总是对整个项目生成测试用例，导致测试结果冗余和庞大；另外尽管生成的测试结构简洁，但测试数据往往较为随机，在可读性和易理解性方面有待进一步完善。

利用现有测试用例作为生成示例的想法也曾得到过多番实践。较为基本的例如 Hummel 等人引入的 Code Conjurer[8]，它不区分用户期望的是测试用例还是编程代码，只是通过用户输入查询来从库中搜索相关代码示例。更加有效些的如 Test Recommender [9], 通过向用户推荐同项目的其他测试用例来指导其完成新的测试用例编写。尽管并没有做到直接生成测试代码，但利用历史测试的思想仍然能为测试生成工具的开发提供许多思考和启发。Janjic 等人提出利用测试搜索引擎 sentre 实现 Eclipse 插件 TestTenderer [10]，通过对待测方法进行签名匹配从存储库中查找可用测试用例反馈给用户，但在效率和准确性上仍有待进一步验证。

还有一些工具并不直接生成或推荐测试用例和代码，但能提供有价值的测试数据。最典型的方法是 Pairwise Testing[10]，即在多个单维度数据基础上进行两两组合或高阶组合，最终生成更加全面的测试数据供用户参考。践行这一方法的工具非常多[11]，常见的包括微软的 PICT[12]，YU 等人提出的 ACTS [11]，McDowell 的 AllPairs[13]等等。AUSTIN [12] 是一款面向 C 语言的开源测试数据生成工具，由

[8]https://baike.baidu.com/item/CodePro Analytix

[9]https://github.com/EvoSuite/evosuite

[10]http://www.pairwise.org/

[11]http://www.pairwise.org/tools.asp

[12]https://github.com/microsoft/pict

[13]http://www.mcdowella.demon.co.uk/allPairs.html

Lakhotia 等人设计实现，支持包括随机和爬山等在内的多种搜索算法 [13]，能够生成大部分除指针、字符串与结构体以外类型的数据，但这些测试数据能否揭示源码中的错误仍需要由开发人员自行判定。

1.2.3　程序相似度分析研究现状

程序相似性检测的研究工作已有多年历史。主要应用于抄袭检测和相似分析等领域 [14–16]。包括属性计数法、基于文本的相似检测和基于代码语法结构的相似检测等方法。

属性计数法是一种早期的代码相似检测技术，通过提取源代码的各种属性度量元进行相似度评估。Ottenstein [17] 曾利用 Halstead 软件参数 [18] 评判 Fortran 程序之间的相似性，Berghel [19]、Grielz [20] 以及 Faidhi [21] 等人则尝试了使用更多的度量元来检测代码相似度 [15]。属性计数法实现较为简单，但是缺点也非常明显，作为一种全局度量模式，准确性有待提高。此外 Verco [22] 提出仅仅依靠属性度量检测相似性，即使继续增加向量维数也难以取得更好的效果。因此，很少有工具只采取属性计数来进行代码分析，而是融合了更多的程序结构信息。

基于文本的相似检测方法通过将源代码转换为字符串，然后利用字符串指纹 [23] 等方式进行查找匹配 [24, 25]。单纯的字符串匹配准确率是很低的，甚至无法检测出那些仅仅是修改了变量名或者调换代码段顺序的程序。对此，优化手段之一是在字符串转换过程中首先进行噪音消除 [26]，例如注释、空行、独立的大括号等。而由于代码本身是一种具有特定规则的文本，所以从文本角度而言，代码天生是具有相似性的 [27]。因此另一种优化方式是将源代码转换成语法上有意义的 token 序列 [28, 29]，再利用诸如 LCS [30] 或 K-gram [31] 算法比对两个 token 序列来反映源代码的相似程度。采用这种技术的比对工具有 CP-Miner [32]、CCFinder [33]、CCFinderSW [34] 等。这种方式尽管在检测效果上具有一定的局限性，但实现简单，对空间和计算要求较低，对于规模较大的源代码十分适用。

基于代码语法结构的相似检测方法首先从源程序提取诸如抽象语法树（AST）[35]、程序依赖图（PDG）这样的结构化信息，然后通过相似子树、子图同构等方式进行相似检测。文献 [36] 提出了一种基于语法树指纹识别的可扩展架构，可有效索引数据库中的 AST 表示形式，以期快速准确检测相似集群。文献 [37] 提出了一种先从源代码生成抽象语法树，再遍历语法树获得代码序列，最后对代码序列进行相似比较的方法。相对于 token 和语法树的方式，PDG 在检测非连续的代码克隆方面有显著效果，但在检测连续代码克隆时性能较差且十分耗时。文献 [38] 使用增量的方式来提高 PDG 的效率。文献 [39] 和 Scorpio [40] 则利用启发式方法对其检测能力进一步增强。

1.3　本文的主要工作

为解决初学者的测试开发与学习窘境，本文依慕测 WebIDE，设计并实现了一个基于同源代码匹配的在线测试开发系统。该系统提供代码智能提示、测试覆盖可视化和测试用例自动化生成功能，力求帮助初学者降低学习成本，快速掌握如何编写简洁准确的测试用例。

本文实现了智能提示服务，其主要功能是结合 MonacoEditor 实现代码提示，主要包括代码补全、悬浮提示、跳转定义、引用查找、代码诊断等功能。针对每种编程语言搭建相应的 LSPServer，将 WebSocket 作为通信转发管道，通过中间服务适配层和 LSP 协议实现 MonacoEditor 与不同 LSPServer 之间的统一交互。

本文实现了覆盖可视化服务，其主要职责是将测试用例对源代码的覆盖情况直观地展示给开发人员。通过 OpenClover 提取测试用例的覆盖信息，对数据做格式转换后传递给前端组件，前端通过不同的颜色渲染标识不同的覆盖程度，清晰地显示出未覆盖、部分覆盖和全覆盖三种情况。

本文实现了同源代码匹配服务，首先通过开源网站和考试平台的历史数据构建测试语料库，而后使用程序分析技术提取待测方法信息，接着利用拼写校正、语义相似分析、代码匹配等手段在语料库中进行同源方法检测，同源方法的已有测试用例将作为待测方法的测试用例生成原料。

本文实现了测试自动化生成服务。由用户指定待覆盖的源代码行，通过语料库检索改造或者工具产出测试代码两种方式自动生成简洁清晰、指向性明确的测试用例。语料库生成利用同源代码匹配服务定位相似方法，检索相似方法对应的测试用例，将这些测试用例根据当前的测试上下文进行改造与组合，包括变量声明、方法调用名替换，前置后置条件迁移等。对于语料库未能覆盖的项目，系统利用改造和参数优化后的 Evosuite 工具生成基础测试用例，以供用户参考。

本文提供了符合传统 IDE 习惯的交互方式，解除用户与本地开发环境的捆绑，提供了指向性明确、可执行率高的测试代码供用户学习和使用。

1.4　本文的组织结构

本文的组织结构如下：

第一章 引言。本章介绍了软件测试开发的时代背景和社会意义，阐明了当前初学者编写测试用例所面临的窘境和困难，对现有的 WebIDE、测试生成工具以及程序相似性度量方法的优缺点进行说明，最后简要论述了本文的主要工作内容。

第二章 技术综述。本章对项目使用的 OpenClover、程序结构分析技术、词汇相似度计算算法、代码相似度度量方法、Evosuite 工具、ElasticSearch 存储、LSP 协议、Docker 容器等技术展开简要介绍。

第三章 系统需求分析与设计。本章阐述了系统的整体思路，对系统的功能需求和非功能需求进行了说明。介绍了系统的整体框架和实现思路，详细阐释了系统四大模块——即智能提示模块、覆盖可视化模块、同源代码匹配模块和测试自动化生成模块的架构设计、核心类构成、数据存储格式定义与相关参数配置。

第四章 系统的具体实现。在前一章的分析和整体设计基础上，本章对智能提示模块、覆盖可视化模块、同源代码匹配模块、测试自动化生成模块的具体实现进行详细说明，通过顺序图阐述调用流程，通过关键代码描述实现细节，最后通过系统的主要界面展示实现效果。

第五章 系统测试和实验分析。本章根据第三章的功能点说明和技术实现方案，对系统进行功能和性能测试，并设计实验验证系统的应用价值与意义。包括测试用例设计说明、测试执行结果评估、实验设计、实验执行与实验结果分析。

第六章 总结与展望。本章对系统和论文的相关工作进行总结，说明了系统解决的问题和优势所在。同时对未来工作进行展望，指出了系统现存的不足之处与进一步改进思路。

第二章　相关概念和技术

本章对系统中采用的技术进行简要说明。系统采用 OpenClover 来度量测试代码对项目源码的覆盖情况以便页面进行着色渲染；程序结构分析过程将源代码转换为抽象语法树 AST，通过 JavaParser[1]和 CallGraph 提取程序的文本、结构以及调用信息；被测代码匹配环节采用 Levenshtain 距离进行拼写校正，使用 Word2Vec 查找方法名近义词，利用余弦距离度量相似方法体；将数据存入 ElasticSearch 提高检索效率；将 Evosuite 作为补充手段弥补测试代码库的不足；基于 LSP 构建智能提示服务；服务部署统一封装到 Docker 容器以便管理。

2.1　OpenClover 技术

2.1.1　OpenClover 简介

OpenClover[2]是一个开源的 Java 源代码覆盖工具。代码的覆盖率作为测试执行的一项重要定量数据，充分显示了代码的哪些部分通过了测试，哪些部分仍未测试到。目前存在三种类型的覆盖检测工具，分别是源代码检测工具、字节码检测工具和运行时信息收集工具。OpenClover 使用源代码检测这一准确率更高的方式。覆盖的度量标准通常包括语句覆盖、分支覆盖和方法覆盖 [41]。OpenClover 将这三者进行组合来计算总覆盖率（TPC）：

$$TPC = (BT + BF + SC + MC)/(2 \times B + S + M) \times 100\% \tag{2.1}$$

其中，BT 代表分支为 True 的数量，BF 代表分支为 False 的数量，SC 代表被覆盖的语句数量，MC 代表被调用的方法数量，B 为分支总数，S 为语句总数，M 为方法声明总数。

使用 OpenClover 获取测试对源码的覆盖情况需要经过以下步骤。首先清除已编译的字节码文件，确保所有代码都被重新编译；接着初始化 OpenClover 插件并对源码进行插桩；然后将源代码编译为字节码，运行测试用例记录覆盖率信息；最后对覆盖信息进行整合并生成报告。

[1]https://javaparser.org/

[2]https://openclover.org/

2.1.2　OpenClover 优势

目前市面上也存在许多代码覆盖工具，例如 Jacoco、Jcov、EMMA 等，相比之下，OpenClover 具有以下几点优势：

首先，OpenClover 对源代码进行插桩来计算覆盖率，准确率更高的同时，使得覆盖率与源代码行一一对应，满足覆盖分析后反馈到源码行进行着色渲染的需求。而诸如 Jacoco、Jcov 和 EMMA 均对编译后的字节码插桩，导致覆盖分析难以对应到源码行。其次，OpenClover 能够跟踪测试用例与程序类的相关性，因此只需重新运行与修改相关的测试，从而减少运行时耗。另外，OpenClover 支持灵活定义覆盖范围，对于 Getter/Setter 等边缘代码可以选择排除在外，从而只专注于核心代码的测试覆盖情况。最后，作为开源项目[3]，OpenClover 一直持续维护和更新，而 Jcov 和 EMMA 已经逐渐丧失对 JDK8 的支持。综合以上优势考虑，系统最终采用 OpenClover 作为代码覆盖工具。

2.2　程序结构分析

本系统在搜索测试用例前需要获得待测程序的文本、结构、调用等信息。作为描述程序的一种普遍且全面的方法，抽象语法树（abstract syntax code，AST）较好地抽象和压缩了主体程序。本系统通过 JavaParser 和符号解析器完成代码到 AST 的转换，利用 CallGraph 来获取源程序方法间的调用关系。

2.2.1　AST

抽象语法树是源代码语法结构的树状抽象表示，树上的每个节点都标识了源代码中的一种结构。它能够将编程语言特定的标签转换为通用标识符，从而构成语言无关的树状结构。节点包含的信息可能是方法声明、变量初始化、表达式、方法调用等，通过将这些信息做结构化表示，能够更加方便快捷地了解程序的目标和实现方式，以及该程序与其他程序之间的关系。本系统主要利用 AST 提取方法名，参数的个数和类型，代码的去噪表示以及方法间的调用关系。

2.2.2　JavaParser

JavaParser 是一个将 Java 代码与抽象语法树互相转换的开源工具[4]。通过 JavaParser，可以对一个 Java 代码片段进行词法分析和语法分析，提取类、方法、参数等信息，并输出为一个编译单元（Compilation Unit），其内部结构就是抽象语法树。类定义、方法的声明、方法调用语句、变量初始化等全都被抽象为树上

[3]https://github.com/openclover/clover

[4]https://github.com/javaparser/javaparser

的节点，因此对这棵树进行遍历，几乎可以获取源码包含的任何信息。JavaParser 提供了基于访问者模式的 Visitor 类，通过实现对应的子类和方法覆盖，即可遍历节点，定制所需的信息内容。

但是要获取变量的类型信息,还需要在 JavaParser 的基础上配合使用 JavaSymbolResolver。JavaSymbolResolver 是一个 Java 语法语义解析库，提供了强大的符号解析能力。JavaParser 解析代码后可以获得变量的名称，但是无法确认变量所属的类型（基本类型除外）：例如语句 A a = new A()，JavaParser 可以获得对象 a 的声明类为 A，但是无法得知 A 的命名空间。而 JavaSymbolResolver 能够通过不断回溯当前节点的父节点，最终确认子节点类的完全限定名称，即 x.y.A，从而唯一确定对象 a 的类型。

JavaParser 也能够将编译单元转换成代码片段，从而满足自动生成代码的需求场景。EclipseJDT [5]也是一种较为流行的 Java AST 转换工具，但是由于其本身作为 eclipse 服务的一个功能模块，并没有被单独分离，如果使用需要同时引入多个依赖，并容易出现版本不一致和不兼容的问题；JDK 其实也自带了一套编译 API，可以实现将代码转换为 AST，但是这套 API 并没有被 Oracle 和 OpenJDK 发布为公开 API，相关文档也十分匮乏。综合来看，JavaParser 作为一个应用广泛的开源工具，使用方式简洁方便，具有活跃的维护更新和讨论，更适合作为本系统的 AST 转换工具。

2.2.3　Static Call Graph

调用图 [6]是程序控制流程的一部分，表示了方法之间的调用关系。调用图分为静态和动态。静态调用图通过对程序进行静态分析获取，包含了可能存在的所有调用关系，即使某些调用可能永远不会被执行到。动态调用图通过运行程序来获得准确的调用关系，缺点是仅仅能够描述该程序的一次运行，如果需要包含更多的调用关系，就需要使用多种数据多次运行源程序，分析成本较高。

在编写测试的过程中，对于待测项目的私有方法无法直接进行调用。因此本系统借助调用图来寻找调用了私有方法的公开方法，通过对该公开方法进行测试，从而达到测试私有方法的目的。因此，尽管静态调用图只能够得到近似的调用关系，但分析速度快，包含的调用关系全面。经过对效果和效率的双重考虑，系统使用 java-callgraph [7]工具的静态方式实现调用关系分析。

[5]https://www.eclipse.org/jdt/

[6]https://en.wikipedia.org/wiki/Callgraph

[7]https://github.com/gousiosg/java-callgraph

2.3　Word2Vec 技术

2.3.1　技术简介

在本系统中，判定待测方法为同源代码的第一步是对待测方法名进行近义词查找。在编程过程中，开发者可能对实现同样功能的方法采用不同的命名，因此需要比对方法名的词义相似性作为同源代码度量的依据之一。然而方法名作为一种文本，无法直接进行对比，因此系统需要将非结构化的词汇转换为结构化的向量表示，即获得词向量，从而进行计算和匹配。Word2Vec [42–44] 通过神经网络训练词向量，利用词向量来表示词的语义信息，将词汇从只有人类可理解的抽象总结嵌入到计算机能够理解的数学空间中，使得意义相近的词汇在向量空间中的距离较近。

2.3.2　词向量表示方法

在自然语言处理过程中常见的词向量表示方法主要有两种：

One-hot Representation：将字典的长度作为向量的维度来表示一个单词，一个单词所对应的向量仅有一个位置为 1，其他位置均为 0。例如，假设我们所分析的文本中仅包含“how are you”，则各个单词表示如下：

“how”：[1,0,0]

“are”：[0,1,0]

“you”：[0,0,1]

这种方式的优点是简单易懂，但实际分析的文本中可能包含成千上万的词汇，因而造成维度爆炸，且向量存储过于稀疏，导致计算和存储效率很低。此外，这种表示方式也无法反映单词之间的关系。

Distributed Representation：与 One-hot Representation 不同，这种表示方式采用低维、稠密的向量来表示一个词。向量表示示例如下：

[0.692, −0.267, −0.115, 0.019, −0.312, ...]

这种方式最大的好处是让相似或者相关词之间的向量距离更短，通过挖掘词与词之间的关联关系，提高向量的语义准确度。由于 One-hot Representation 在应用方面的重要缺陷，稀疏表示法已经逐渐向低维空间的密集表示过渡。Word2Vec 正是这种表示的一种实现方法。

2.3.3　Word2Vec 实现方式

CBOW 模型（Continuous Bag-of-Words Model）和 Skip-gram 模型（Continuous Skip-gram 模型）是 Word2Vec 中的两个重要模型。两者在概念上最大的区别是，

CBOW 通过输入上下文词关系来预测目标词汇结果作为输出；而 Skip-gram 则恰恰相反，通过输入一个词的词向量，输出该词对应的上下文预测结果。Word2vec 分别基于 Hierarchical Softmax 和 Negative Sampling 设计了两套框架，用于实现这两个重要模型。

在基于 Hierarchical Softmax 的实现方式中，模型包含输入层、投影层和输出层的三层结构。下面以 (Context(w),w) 这一样本为例说明两种模型的区别，其中 w 为中心词，而 Context(w) 指由词 w 前后各 c 个词所组成的上下文。CBOW 的三层含义分别如下：

输入层：包含 Context(w) 的 2c 个词的词向量 v(Context(w)$_1$)，v(Context(w)$_2$)，...，v(Context(w)$_2c$)。

投影层：将输入层的 2c 个向量进行求和操作，即:

$$\mathbf{x}_w = \sum_{i=1}^{2c} \mathbf{v}(\text{Context}(w)_i) \in \mathbb{R}^m \tag{2.2}$$

输出层：对应一棵 Huffman 树，其叶子节点是语料中所有出现过的词，权值为该词出现的次数。

与神经概率语言模型相比，CBOW 模型将从输入层到投影层的操作由拼接变为累加，删除了隐藏层，输出层也由线性结构改为二叉树结构。总的来说，CBOW 通过对计算复杂度较高的地方进行模型简化，大大提高了训练效率。

Skip-gram 的三层含义如下：

输入层：仅含当前样本中心词 w 的词向量 v(w)。

投影层：恒等投影，即：将 v(w) 投影为 v(w)。实际上这一层并不需要，但保留这一层有助于与 CBOW 进行对比。

输出层：同样是一棵 Huffman 树。

由于输入和目标的不同，两种模型各自采用了不同的目标函数和梯度计算方法，这里不再展开详细介绍。

基于 Negative Sampling 的实现方式，是对 Noise Constrastive Estimation 的简化，与 Hierarchical Softmax 相比，这种方式使用较为简单的随机负采样替代复杂的 Huffman 树结构，能够带来更加大幅的性能提升。

2.3.4　优势分析

词义相似匹配的常见手段还有 WordNet [8]。但经过简单实验后发现，尽管 WordNet 的处理效率很高，但语义词典包含范围较小，且不同词性的词语相似度

[8]https://wordnet.princeton.edu/

永远为 0，难以满足实际使用需要。Word2Vec 则不存在这类问题。本系统最终采用 Google 预训练好的模型 googlenews-vector-negative300.bin 作为词汇相似语义分析的预训练模型。由于函数方法名本质上是对所实现功能的一种解释，因此其语义范围其实就是自然语言的语义范围。googlenews 以谷歌新闻作为语料来源，涉及的词汇量广，采用负采样算法实现，在效率上也具有较好保证。

2.4　Levenshtein 距离

2.4.1　Levenshtain 距离简介

Levenshtein 距离[9]这一概念，由 Vladimir Levenshtein 提出，对于度量两个字符串间的差异程度具有较好效果。通常认为 Levenshtein 距离是由一个字符串变换为另一个字符串时所需要的最少字符编辑次数，字符编辑包括对字符的插入、修改和删除等操作，因此 Levenshtein 距离也被称作编辑距离。设字符串 a 和 b 的长度分别为 |a| 和 |b|, 则 $\mathrm{lev}_{a,b}(|a|,|b|)$ 满足下列公式：

$$\mathrm{lev}_{a,b}(i,j)=\begin{cases} i & ,j=0 \\ j & ,i=0 \\ \min\begin{cases} \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i\neq b_i)} \end{cases} & ,otherwise \end{cases} \tag{2.3}$$

Levenshtein 距离在拼写检查和校正领域具有广泛应用。在代码编写过程中，经常出现编程人员对方法名词汇拼写错位的情况，目标方法名与实际编写的方法名之间往往只是一到两个字符的差距，却会导致名称相似匹配的失败。本系统使用 Levensthein 距离来帮助识别和校正待测方法名与语料库方法名之间的拼写错误。

2.4.2　Levenshtain 距离计算的对称思想优化

普通的拼写校正通过遍历字典，计算每个词条与目标单词之间的编辑距离，将编辑距离最小的 n 个词条作为校正选项。可以估算，当字典中包含的词汇量巨大时，对于每个目标单词都要进行全量计算，存储和耗时将超出可接受的范围。即使在此基础上进行优化（例如仅计算与用户输入单词长度相差 2 以内的词条），计算量也十分庞大，可能严重拉低系统的响应速度。

可以考虑只处理目标单词来提高拼写校正的执行效率。对目标单词进行所有编辑距离小于等于 x 的变换 (x 根据实际需求设置)，变换包含了对其中各个字

[9]https://en.wikipedia.org/wiki/Levenshtein_distance

符的删除、修改、添加及操作间的组合，从而获取到若干个与原始单词的编辑距离在 2 以内的新字符串。然后直接查询这些新字符串，如果能够在字典中查询到结果，则拼写检查成功。相对于普通的拼写校正，这一方法在处理效率上大大提升。但如果字符的种类很多，即使只针对一个单词，变换的复杂度依旧很高。

本系统采用对称删除拼写校正[10]。过程如下：首先对字典进行预处理，对字典中的每个词条生成与其编辑距离在 x 以内仅进行删除操作的字符串，并添加到新的字典中。之后所有的操作都基于这个新的字典。然后处理目标单词，生成与该单词编辑距离在 x 以内的仅进行删除操作的字符串。而后在新字典中查询这些字符串，如果查询到结果，再与原始字典进行对应。与前一种方法相比，这种方式只进行删除操作，极大地提升了效率。

2.5　基于 n-gram 的余弦距离

本系统在度量方法的相似性上，除了考虑输入输出结构与方法名相似性外，也将代码体的相似度纳入考量范围，在方法名具有一定相似性的基础上，代码相似度最高的方法将被认为是最接近目标方法的。

n-gram 算法将文本转换为 n 个字符的序列，通过比较序列的相似度或者距离来衡量两段文本之间的相似度。我们采用余弦距离度量字符序列之间的相似度，首先将字符序列转换为向量，例如对于字符串“ABCD”和“ABCE”，选取 n 为 2，则字符序列为：AB、BC、CD、CE，那么“ABCD”可表示为向量 (1,1,1,0)，“ABCE”可表示为向量 (1,1,0,1)。而后计算两个向量夹角的余弦值，作为度量二者相似性的依据，夹角越小，向量越接近，文本也就越相似。

给定向量 A、B，余弦值计算如下：

$$\cos\theta = \frac{A \cdot \mathrm{B}}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.4}$$

在不考虑语义的情况下，余弦距离对于文本相似度量能够达到较好的效果，且具有较高的计算效率。而对于具有特定规则的代码文本而言，自然语言语义相似度的参考意义很小甚至可以忽略。因此综合考虑效果和效率，系统最终采用余弦距离度量方法代码相似性。

2.6　Evosuite

本系统采用 Evosuite 工具进行基础测试用例生成，作为语料库未覆盖方法测试生成的补偿手段。Evosuite 基于 GA 算法产生测试数据 [13]：首先随机生成

[10]https://medium.com/@wolfgarbe/1000x-faster-spelling-correction-algorithm-2012-8701fcd87a5f

一定数量的测试数据，作为程序输入执行插桩后的源程序，然后度量测试数据的优劣性，再利用选择、变异、交叉等遗传算子生成新的测试数据，重复这一过程直到测试数据的效果达到预期。

Evosuite 具有一些非常显著的优势，包括详尽的使用文档、持续稳定的维护、高可配置性等。尽管 Evosuite 的稳定发行版只支持对于整个项目进行测试生成，但经过改造和优化后，系统采用的 Evosuite 可针对单个方法生成测试用例，从而有效地控制测试用例的数量和测试生成的时间。同时系统对 Evosuite 的各个配置项：包括覆盖准则、搜索时间、依赖设置、后置检查等选值进行测试和评估，综合考虑生成效果和时耗，最终确定参数值。

2.7 Elasticsearch

Elasticsearch[11] 支持多种数据类型存储和多种数据来源，并能对所存储的数据建立多种类型的索引，以便支持用户对这些数据的复杂查询，在处理速度和可扩展性等方面具有出色表现。常常被应用于程序、网站、企业等搜索场景，也能够用于日志分析、安全分析和业务分析，在监测和数据可视化领域也有诸多应用。在易用性方面，ElasticSearch 提供简单的 RESTful API，支持分布式和多租户，客户端可适用于包括 Java、Python、PHP 等在内的多种编程语言。

ElasticSearch 的主要优势如下：

第一，将数据检索、数据分析与分布式技术结合统筹，提供 HTTP Web 接口和 JSON 文档，操作简单，开箱即用。

第二，支持各个字段的索引建立和搜索；且整个过程几乎是实时的，大部分情况下仅需 1s 左右

第三，作为分布式实时分析搜索引擎，索引可被存储到多个分片，能够扩展到上百台服务器，处理 PB 级的数据；也可以运行在单机上，支持中小型业务。

总的来说，ES 是一个功能强大，使用简单的分布式的全文搜索引擎。相对于 MySQL 或者 MongoDB，ES 速度快、效率高，容量大，特别适用于大量文本和字符串检索分析的场景。

2.8 LSP

LSP [12]基于 JSON-RPC 的开放协议，由 Microsoft 创建，旨在为编程语言分析人员定义一种通用语言。包括 Codenvy，Red Hat 和 Sourcegraph 在内的多家公

[11]https://www.elastic.co/cn/what-is/elasticsearch

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

司已经齐心协力支持其发展，且该协议正得到迅速增长的编程和语言社区的支持。在 LSP 之前，为实现诸如代码补全、引用查找、悬浮提示、语法诊断等智能提示功能，各个开发工具需要为各种语言独立开发语言服务器。假设有 m 种编辑器，n 种编程语言，则为实现提示功能，需要开发 m*n 个语言服务器。以支持 Python 语言为例，VSCode、Sublime Text、Vim、Eclipse 等开发工具都需要构建自己的 Python 提示插件，由此带来的复杂度和工作量是很庞大的。LSP 提出的想法是将语言服务器与开发工具之间的通信协议标准化，使用统一的高性能语言服务器提供智能提示能力。从而将 m*n 问题转化为 m+n 问题，大大降低了复杂度。

本系统使用 LSP 作为编程语言智能提示通信协议，采用 Eclipse JDT LS [13]作为 Java 语言服务器，采用 plantir 的 python language server [14]作为 Python 语言服务器，结合 WebSocket 和 Socket 通信机制，做适当改造和适配后实现 WebIDE 的智能提示功能。

2.9 Docker

Docker 是一款类似虚拟机的开源工具，利用容器机制更加轻松地创建、部署、运行和停止目标应用程序。虽然都能做到环境的隔离，但相对于传统的虚拟机环境，Docker 容器的启动几乎是秒级的，对硬盘的使用量更小，性能也更加接近原始服务器体验，单台服务器能够支持数十甚至上百个容器同时运行。利用 Docker 部署应用程序具有以下优势：

隔离性。Docker 容器与服务器操作系统、容器与容器之间环境互相独立，包括网络、进程、用户工作空间和文件系统等，对单一容器的操作影响范围仅在该容器内部。

便携性。利用 Dockerfile 即可创建 Docker 镜像，Docker 镜像可以由公开或私有的镜像仓库进行管理，任何服务器只要安装了 Docker 环境，便可拉取镜像，运行容器。

可管理性。资源隔离机制使得操作系统可以为不同的容器分配不同的资源配额。此外，Docker 还提供了完善的 tag 管理机制，可用于管理镜像的不同版本，为不同的应用场景提供支持。Docker-compose 支持多个容器的合并管理，包括启动、运行和销毁。

安全性。资源隔离机制使得单个容器的风险自控制，CGroups 机制和 Namespace 机制确保了 Linux 安全标准的实现。

[13]https://github.com/eclipse/eclipse.jdt.ls/

[14]https://github.com/palantir/python-language-server

本系统涵盖了多个服务，利用 Docker 容器进行部署，Docker-compose 进行管理，使得服务有序部署，同时便于资源分配和服务迁移。

2.10　本章小结

本章主要介绍了系统的技术选型，对各个技术的原理、实现与使用优势进行了简要说明。首先，对测试覆盖提取技术 OpenClover 进行介绍；接着，对系统使用的程序结构分析技术和工具展开说明，主要包括利用抽象语法树提取程序信息，借助 JavaParser 和 CallGraph 实现程序转换和调用关系提炼；而后对相似度分析算法进行阐述，Word2Vec 主要用于词义匹配，Levenshtain 距离用于拼写校正，而余弦距离则注重于判断方法体代码的相似性；接下来介绍了 Evosuite 工具、系统的存储和搜索引擎 ElasticSearch、实现 IDE 智能语法提示的 LSP 协议和用于部署与管理系统组件的 Docker。

第三章　测试开发系统的需求与设计

本章将概述基于同源代码匹配的测试开发系统的需求分析和设计。首先，对系统的应用背景和整体思路展开介绍；而后分析了系统的功能需求和非功能需求，并针对功能需求进行用例描述；接着通过架构图、架构描述以及 4+1 视图阐述系统整体设计；最后详细介绍了系统各个模块的架构设计与实现思路。

3.1　系统整体概述

在软件测试过程中，一些新手可能对编写测试感到无从下手，面临测试结构混乱、用例编写不准确、功能点覆盖率低等问题。针对新手的测试窘境，本系统依托慕测 WebIDE，在提供代码着色和智能提示服务的同时，以 Junit 为基础，利用已有测试用例构建语料库，通过相似匹配等手段，为用户生成新的测试用例以供参考。生成的测试用例具有简洁、明确、可执行率高等特点，致力于帮助用户打开测试编写思路，提高源码覆盖的同时，提升用户测试技能。

图 3.1是本系统基于同源代码匹配进行测试生成的过程，包含离线的语料库预处理和实时方法匹配及测试生成两个流程。本文称目的和实现相似的一组方法具有同源性。在语料库预处理流程中，从 GitHub、慕测主站等平台获取带有测试用例的 Java 项目后，首先通过程序切片将项目中的测试类切割成每个用例仅包含一个待测方法调用的测试片段；接着通过重新组合代码及运行分析获得切割后的测试片段对被测方法的覆盖情况；最后通过数据处理程序转换为特定结构并存储到语料库中。

在实时方法匹配和测试生成流程中，首先对用户的待测类进行程序分析，提取待测方法的各项信息，包括方法名，参数数量和类型，方法体代码等；然后进行待测方法与库中已有方法的相似匹配，在方法输入输出结构相同的基础上，首先进行方法名的完全匹配；如果完全匹配失败，则尝试通过拼写校正寻找相似方法名；如果拼写校正仍然未能获取结果，再进行方法名的词义相似比对，取高于阈值的前 10 个结果返回；获得相似方法名后，通过将各个方法体与待测方法代码进行相似分析，最终确定相似度最高的语料库方法，该方法将被认为与待测方法同源；对该方法的测试用例做上下文改造后返回给用户。如果没有符合条件的方法，则通过 Evosuite 工具来生成数据较为随机的基础测试用例。

通过这两个流程，系统能够借助历史测试数据，搭建语料库，以较高的准确率来匹配待测方法，从而为用户提供简洁、准确、可用的测试用例。

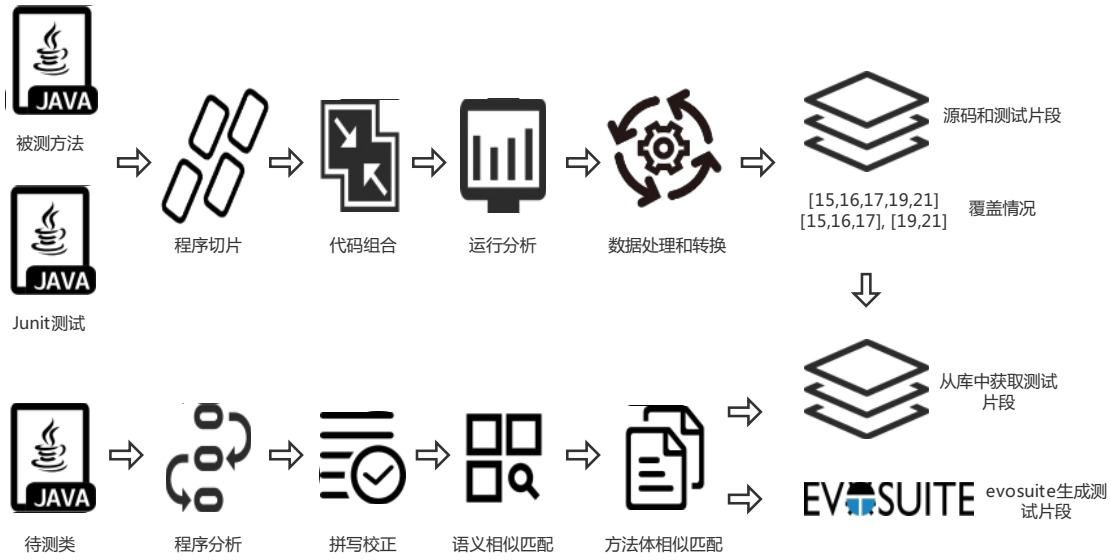


图 3.1: 基于同源代码匹配的测试生成过程

3.2　系统需求分析

3.2.1　功能性需求

系统的功能性需求主要从智能提示、代码着色、同源代码匹配和测试自动化生成四个模块进行分析，涉及的功能性需求如表 3.1所示：

智能提示模块的功能是在用户编写代码的过程中给予代码提示，主要包括代码补全、悬浮提示、定义跳转、引用查找、代码诊断等。系统能够根据用户当前编写的不同编程语言，自动提供不同的语法诊断和提示。

覆盖可视化模块的功能是在用户运行项目代码和测试后，根据测试用例对源码的覆盖情况进行代码行着色。未覆盖行渲染为红色，部分覆盖的行渲染为黄色，完全覆盖的行渲染为绿色。教师用户可以在慕测平台的考试管理页面选择开启或关闭此功能。如果教师选择开启此功能，为防止干扰，学生用户仍然可以在 WebIDE 中自行选择开启或关闭这一能力。

同源代码匹配模块的主要功能是对待测项目进行程序分析，提取待测方法的名称、参数、方法调用等信息。而后通过相似匹配算法在库中搜索相似方法的测试用例。由于在测试场景下，项目代码的变动较少，为提高效率，系统对程序分析结果进行了缓存，学生可以选择清除缓存重新分析项目信息。

测试自动化生成模块的功能是用户在浏览项目源代码过程中，可以选择对指定的方法进行测试用例生成。对于同一待测方法只允许同时存在一个生成进程。用例生成可能是来自语料库的历史测试，也可能是由工具生成的基本测试用例。对于来自语料库的测试，系统根据待测方法上下文改造后反馈给用户。生

成过程结束后，系统自动为用户打开生成的测试文件，用户可以选择保留或删除所生成的文件。也可以对原始测试用例进行查看。

表 3.1: 功能需求列表

需求 ID	需求名称	需求描述
R1	智能提示	用户在 WebIDE 编写代码时，系统能够实时进行智能提示。主要包括代码补全、方法和变量提示、引用跳转、悬浮说明等。
R2	代码着色	用户在 WebIDE 运行或提交代码后，系统能够自动对被测试的源代码行进行着色。红色代表未覆盖，绿色代表完全覆盖，黄色代表部分覆盖。
R3	方法级测试代码生成	当用户浏览源代码时，能够对整个项目或者源代码中指定的方法进行测试用例生成。新生成的测试用例应当存储到对应的测试目录下，且自动为用户打开该测试文件。
R4	测试代码修复	对于从库中生成的测试代码，系统应自动对其进行修正，包括包名、类名、方法名的替换。
R5	查看原始测试用例列表	用户触发测试代码生成功能后，如果是从语料库生成的测试代码，用户可以查看用来生成该测试类的原始测试用例。
R6	缓存清除	用户对项目源文件进行修改后，可以进行缓存清除以获取新的程序分析产物。
R7	功能开关配置	对于代码着色和测试生成功能，教师用户可以设置这些功能是否对学生开启。

3.2.2　非功能性需求

表 3.2: 非功能需求列表

实时性	用户在 WebIDE 编写代码时，针对用户的操作，系统能够在 0.5s 内给出提示反馈。
	用户请求测试代码生成时，如果能够从库中生成，系统需要在 5s 内反馈结果。如果无法从库中生成需要调用 Evosuite，则系统应在 20s 内给出生成结果。
可扩展性	如果系统需要集成一种新的语言智能提示服务，应能够在 1 人天内完成。
	对于相似度计算算法，可以在不修改系统代码的基础上扩展和替换，整个过程可以在 1 人天内完成。
可维护性	系统的接口应当被良好地抽象，如果发现缺陷或者服务出现故障，应能在 0.5 人天内排查并修复。
易用性	系统的交互方式应当足够清晰，用户无需任何配置和学习成本。
可靠性	系统服务应当能承载至少 200 人同时使用，如果系统崩溃，服务器数据不能丢失。

系统的非功能需求如表 3.2所示。在实时性方面，为提供有效的代码提示以及流畅的用户体验，系统需要快速响应用户的编程行为。在可扩展性方面，WebIDE 作为一个不断延展的平台，所支持的编程语言也在不断增加，因此系统必须能够快速集成新的 LSPServer，以应对新语言的代码提示需求；且目前系统虽然已

经完成了从相似匹配到测试生成的整套流程框架，但未来还要对相似度分析算法不断迭代和优化，因此系统必须支持服务的扩展和替换。在可维护性方面，系统面向接口编程，所有的功能都被抽象为接口，所有的维护可在实现内部变更，而不会影响外层服务调用。在易用性方面，系统仅需一个浏览器即可访问，无任何配置成本；同时契合本地 IDE 的使用习惯，新的功能具有良好的入口提示；对于用户的错误操作也会显式指明原因所在，用户需要花费的学习成本几乎为 0；在可靠性方面，系统通过拆分服务减轻部署压力，并利用 Nginx 进行水平扩展，能够支持数百人同时使用；且系统的所有数据均已持久化备份，即使服务宕机，数据也可以完好保存。

3.2.3　系统用例描述

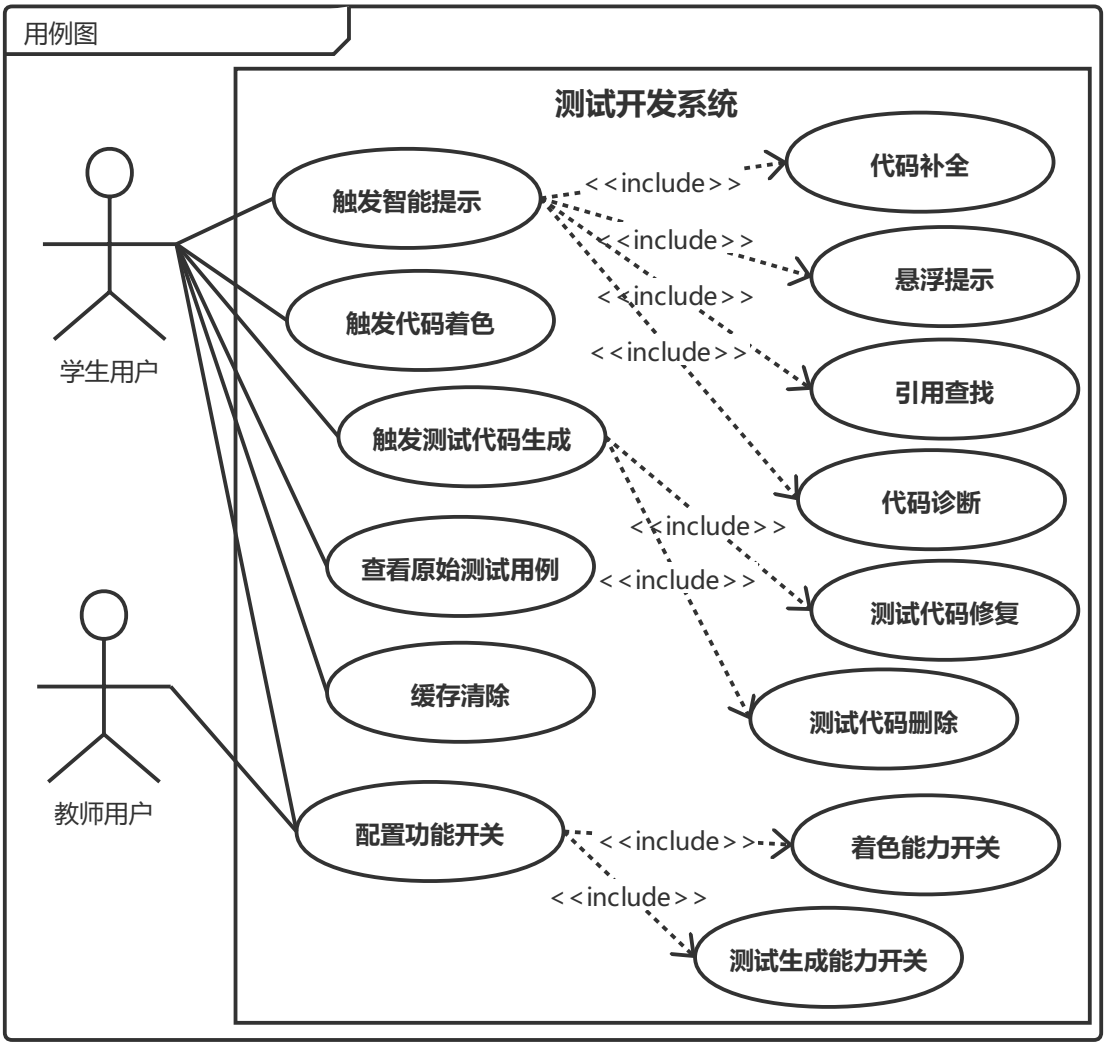


图 3.2: 系统用例图

根据表 3.1中的功能需求描述，系统用例图如图 3.2所示。本系统的主要用户是学生，用户可以在系统中获取代码智能提示，包括代码补全、悬浮提示、引用查找和代码诊断信息；系统自动根据用户编写的测试用例进行源代码着色；用户能够主动触发测试代码生成，系统会对生成的测试类进行自动修复，最大程度保证其可执行；用户也可以查看原始测试用例；还可以根据自身需要清除程序分析的结果缓存。教师用户可以根据教学和考核需要选择开启或关闭系统的各项能力。表 3.3是系统用例表，表明了各个用例与系统功能需求之间的对应关系。接下来对系统用例进行详细描述。

表 3.3: 系统用例表

用例编号	用例名称	对应功能需求编号
UC1	触发智能提示	R1
UC2	触发代码着色	R2
UC3	触发测试代码生成	R3、R4
UC4	查看原始测试用例	R5
UC5	清除缓存	R6
UC6	配置功能开关	R7

表 3.4: 触发智能提示用例描述

ID	**UC1**	**名称**	触发智能提示
参与者	学生用户，目的是在编程过程中获取智能代码提示		
触发条件	用户打开代码源文件		
前置条件	用户编写的代码语言在提示范围内		
后置条件	无		
优先级	高		
正常流程	1. 用户打开代码文件 2. 系统启动相应语言的 LSPServer 对当前文件进行检查 3. 用户打开的文件存在语法错误 4. 系统将错误处标红 5. 用户将鼠标悬停至错误处 6. 系统展示错误原因 7. 用户编写代码，打印字母 8. 系统提示相应的变量、类或方法 9. 用户将鼠标悬停在方法名上 10. 系统显示方法的 JavaDoc 11. 用户想要访问被引用的方法或变量 12. 系统跳转至方法定义或变量声明处		
特殊需求	对于用户的编程行为，系统要在 0.5s 内给出提示反馈		

触发智能提示用例描述如表 3.4所示。用户打开代码文件后，系统自动识别当前打开文件所属的编程语言并启动相应的 LSPServer 提供智能提示能力。系统允许用户同时打开多个文件，能够自动对文件进行代码诊断，并对错误和警告进行标识。对于用户在文件上的编写、悬浮、点击等操作，系统能够在短时间内快速给予正确反馈。

表 3.5: 触发代码着色用例描述

ID	**UC2**	**名称**	触发代码着色
参与者	学生用户，目的是在测试运行后对源代码行进行覆盖着色		
触发条件	用户成功运行项目		
前置条件	1. 着色功能开启 2. 用户项目可被正确运行		
后置条件	着色情况一直保留，直到被新的着色信息覆盖		
优先级	高		
正常流程	1. 用户运行项目 2. 系统生成着色信息并进行着色 3. 源代码行未被覆盖 4. 系统将该行渲染为红色 5. 源代码行被部分覆盖 6. 系统将该行渲染为黄色 7. 源代码行被完全覆盖 8. 系统将该行渲染为绿色		
扩展流程	2a. 项目正确运行，系统生成新的着色信息并进行着色 2b. 项目运行失败，系统给出运行日志并提示运行失败，着色情况保持不变		

触发代码着色用例描述如表 3.5所示。如果教师和学生均开启了着色功能，那么用户的项目被正确运行后将触发系统着色功能。系统根据用户的测试用例对源代码的覆盖情况，分别对行未覆盖、部分覆盖、完全覆盖三种情况进行红色、黄色、绿色渲染。当前着色保留至被下一次着色信息覆盖。如果用户重新打开项目，则代码均为未着色状态。

触发测试代码生成用例描述如表 3.6所示。用户在浏览源代码过程中可以为指定方法生成方法级测试用例。如果系统能够在语料库中找到相似被测方法，则快速从语料库中检索对应测试用例进行代码组合，并自动对可能出现类名、方法名不一致问题进行修复，最大程度确保生成的测试类可执行。如果系统无法从语料库中检索到相似被测方法，则尝试利用测试生成工具生成基本测试用例。用户可自行选择保留、修改或删除生成的测试类。

查看原始测试用例的用例描述如表 3.7所示。系统从语料库中生成测试代码后，用户可以选择查看原始测试用例。原始测试用例可能有多个，每个用例作为

表 3.6: 触发测试代码生成用例描述

ID	**UC3**	**名称**	触发测试代码生成
参与者	学生用户，目的是利用系统自动为待测方法生成测试代码		
触发条件	用户选择为指定方法生成测试代码		
前置条件	测试生成能力开启		
后置条件	新生成的测试文件直接存储到项目测试目录		
优先级	高		
正常流程	1. 用户选择生成方法级测试用例 2. 系统进行测试生成 3. 生成的测试类包含语法问题 4. 系统自动进行测试用例修复 5. 修复后的测试用例仍然存在语法错误 6. 系统允许用户自行修正		
扩展流程	2a. 语料库中包含相似待测方法，则系统从语料库进行测试生成 2b. 语料库中不含相似待测方法，则系统调用测试生成工具生成基本测试用例		
特殊需求	1. 如果系统从语料库进行测试生成，需要在 3s 内完成 2. 如果系统利用测试生成工具进行测试生成，需要在 20s 内完成		

单独的测试文件呈现，存储在项目根目录指定文件夹下，子路径与被测方法包名一致。用户可以选择查看或删除这些原始测试用例文件。

清除缓存用例描述如表 3.8所示。如果用户在生成测试用例的过程中对源代码进行了修改，则可以通过清除缓存删除程序分析的中间结果，重新从源程序进行分析。

配置功能开关用例描述如表 3.9所示。教师可以在慕测主站的练习管理页面选择开启或关闭代码着色和测试生成能力。教师选择开启相关能力后，学生仍然可以自行决定是否关闭该能力。

表 3.7: 查看原始测试用例用例描述

ID	**UC4**	**名称**	查看原始测试用例
参与者	学生用户，目的是查看测试代码生成的原始测试用例		
触发条件	系统生成测试代码		
前置条件	测试生成能力开启，生成的测试代码来自语料库		
后置条件	无		
优先级	中		
正常流程	1. 用户选择查看原始测试用例 2. 系统展示原始测试用例 3. 用户删除原始测试用例 4. 系统删除原始测试用例		

表 3.8: 清除缓存用例描述

ID	**UC5**	**名称**	清除缓存
参与者	学生用户，目的是清除程序分析信息的缓存		
触发条件	用户选择清除缓存		
前置条件	已有缓存信息		
后置条件	缓存信息被清除		
优先级	中		
正常流程	1. 用户选择清除缓存信息 2. 系统删除相关缓存信息		

表 3.9: 配置功能开关用例描述

ID	**UC5**	**名称**	配置功能开关
参与者	1. 教师用户，目的是根据教学和考核需要开启或关闭相关能力 2. 学生用户，目的是根据自身需求开启或关闭着色功能		
触发条件	用户开启或关闭着色功能		
前置条件	允许学生进行开关配置的前提是教师选择了开启该功能		
后置条件	相关能力被开启或关闭		
优先级	中		
正常流程	1. 教师开启或关闭代码着色能力 2. 系统根据要求开启或关闭代码着色能力 3. 教师开启代码着色能力后，学生选择开启或关闭代码着色能力 4. 系统根据学生要求开启或关闭代码着色能力 5. 教师开启或关闭测试生成能力 6. 系统根据要求开启或关闭测试生成能力		

3.3　系统总体设计

3.3.1　系统整体架构设计

基于同源代码匹配的在线测试开发系统的系统架构如图 3.3所示。前端使用 React 开发，编辑器从 CodeMirror 替换为 MonacoEditor 以兼容 LSP 实现智能提示。服务端包含了智能提示、覆盖可视化、同源代码匹配和测试生成四个模块。

智能提示服务模块的主要功能是结合 MonacoEditor 实现代码提示，主要包括代码补全、悬浮提示、跳转定义、引用查找、代码诊断等功能。针对每种编程语言搭建相应的 LSPServer，将 WebSocket 作为通信转发管道，通过中间服务适配层和 LSP 协议实现 MonacoEditor 与不同 LSPServer 之间的统一交互。

覆盖可视化服务模块的主要职责是将测试用例对源代码的覆盖情况直观地展示给开发人员。通过 OpenClover 提取测试用例的覆盖信息，对数据做格式转

换后传递给前端组件，前端通过不同的颜色渲染清晰地显示出未覆盖、部分覆盖和全覆盖三种情况。

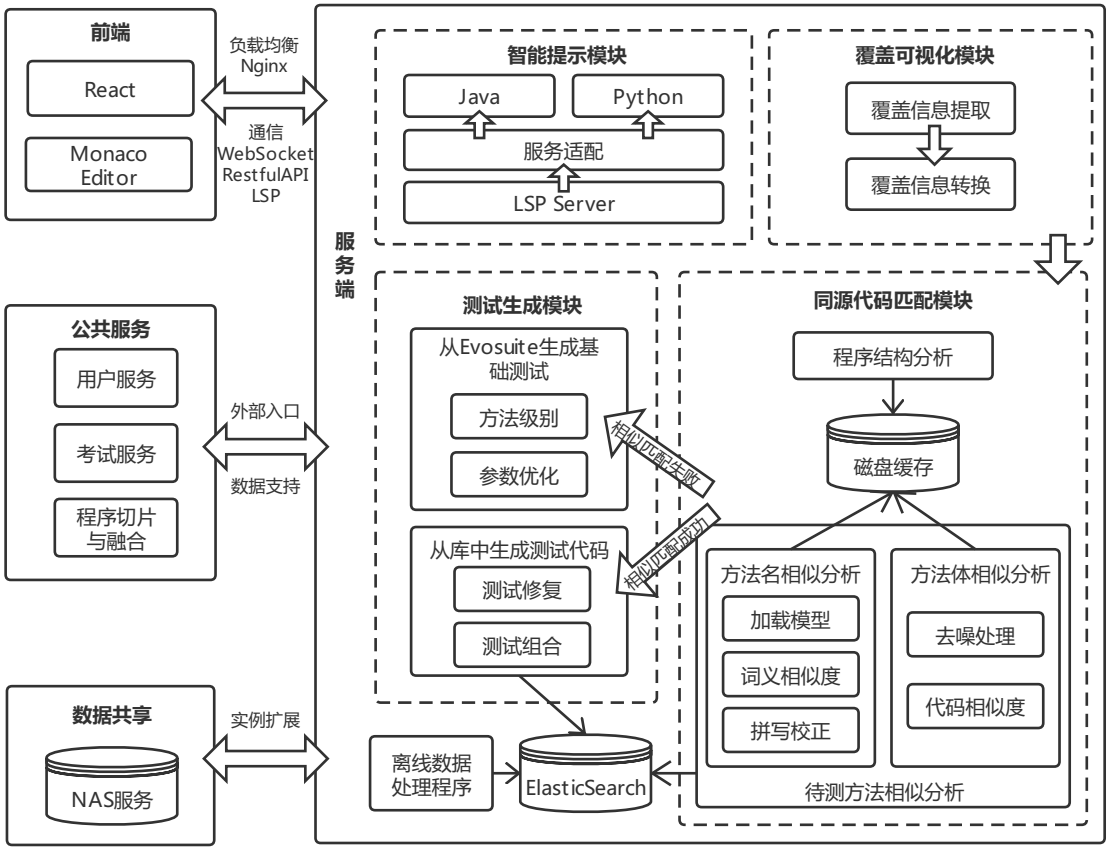


图 3.3: 系统架构图

同源代码匹配模块主要通过同方法结构、字符串匹配、拼写校正、语义相似分析、代码相似度量等手段从 ElasticSearch 中检索可能与用户待测方法相似的方法，然后寻找库中对应该方法的测试用例。同方法结构通过抽象语法树确保库中方法与待测方法必须为相同的输入和输出，包括参数的数量和类型。字符串匹配即寻找方法名的完全匹配，同时为了防止可能存在的拼写错误，系统使用对称优化的 Levenshtain 距离进行拼写校正。词义相似匹配通过 Word2Vec 加载词向量模型，计算库中方法与待测方法名的相似度，达到一定阈值的方法可进行下一步筛选。最后，使用基于 N-gram 的余弦距离来度量方法体代码的相似程度。语料库中的数据来自历史项目和测试用例，首先借助程序切片和融合服务将原始测试代码切割为单个测试用例，然后通过数据处理程序将原项目的被测方法和切片后的测试用例以特定的数据结构存入 ElasticSearch，以便后续检索。

测试自动化生成服务模块通过语料库测试用例改造和工具产出基本测试两种方式自动生成简洁清晰、指向性明确的测试用例。对于同源匹配成功的方法，

首先通过程序分析提取语料库测试用例中的声明、初始化、方法调用等信息，然后根据待测方法上下文对用例进行改造和组合以便获得最大限度可执行的测试代码。对于匹配失败的方法则通过优化后的 Evosuite 提供基本测试用例。

系统使用 Nginx 进行服务水平扩展以应对高负载场景。通过 NAS 实现服务器之间的文件数据共享，从而确保同一用户可以向任意服务器进行请求。利用 ElasticSearch 加快检索效率和响应速度。同时所有独立服务被封装成 Docker 镜像进行部署以方便迁移和复制。

3.3.2　4+1 视图

本小节通过 4+1 视图，从不同的系统参与者视角对系统体系结构进行详细剖析和说明。

场景视图从用户视角看待系统，即根据用户需求描述系统业务的应用场景，通常以用例图的方式进行说明，本系统的用例图如图 3.2所示。

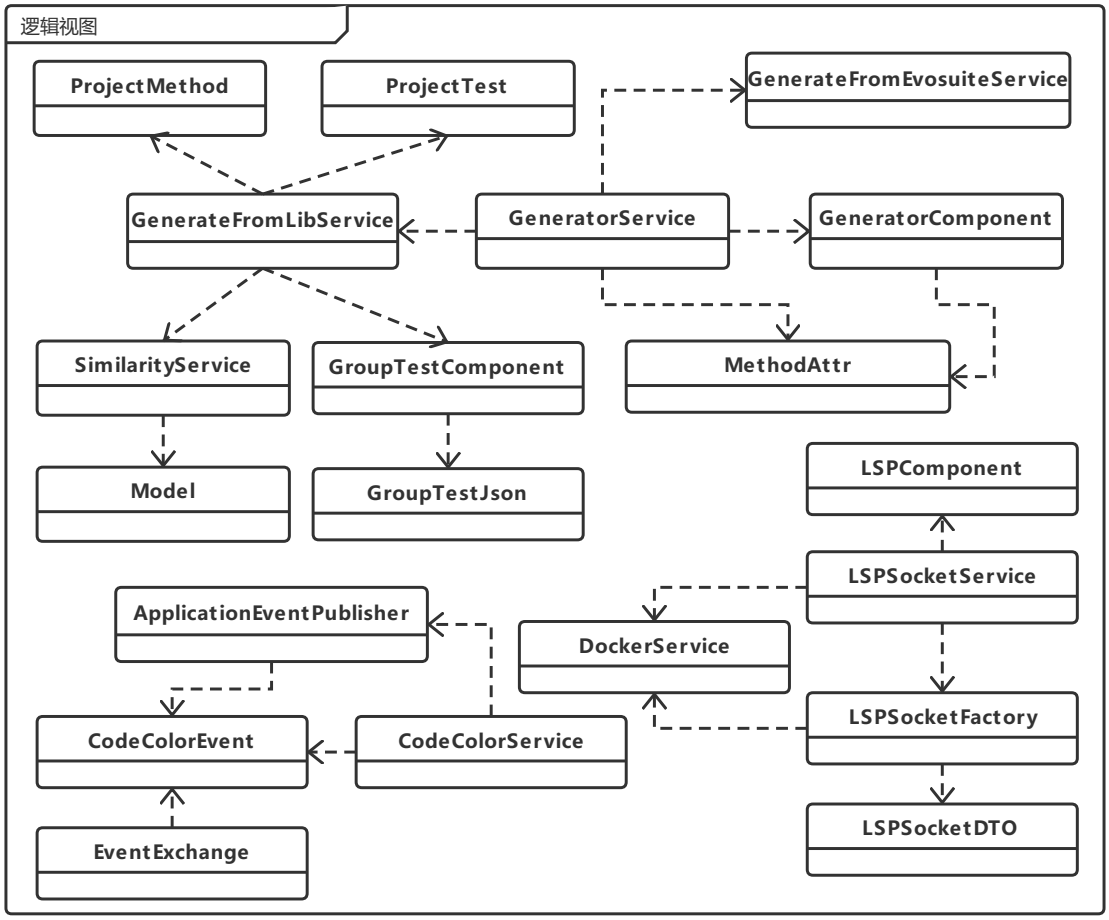


图 3.4: 逻辑视图

逻辑视图从用户角度描述了系统提供的服务，反映系统的服务结构和对象模型设计。可以被分解为一系列关键抽象，对应了 UML 中的类图。本系统的逻辑视图如图 3.4所示。CodeColorService 用以提供代码着色功能。LSPSocketService 负责根据不同语言利用 DockerService 启动相应 LSPServer 容器，从而实现智能提示功能。GeneratorService 提供测试代码生成功能，分为语料库生成和工具生成：GeneratorFromEvosuiteService 提供通过工具产生基本测试用例的能力，而 GeneratorFromLibService 提供了从语料库中检索、匹配和组合测试的能力。GneratorComponent 提供程序分析功能，负责从代码中抽象程序相关信息。SimilarityService 提供方法名相似度计算功能，包括词义相似匹配和拼写校正。GroupTestComponent 提供了修正和组合测试用例的能力，最大程度保证生成测试代码的正确性和可执行。

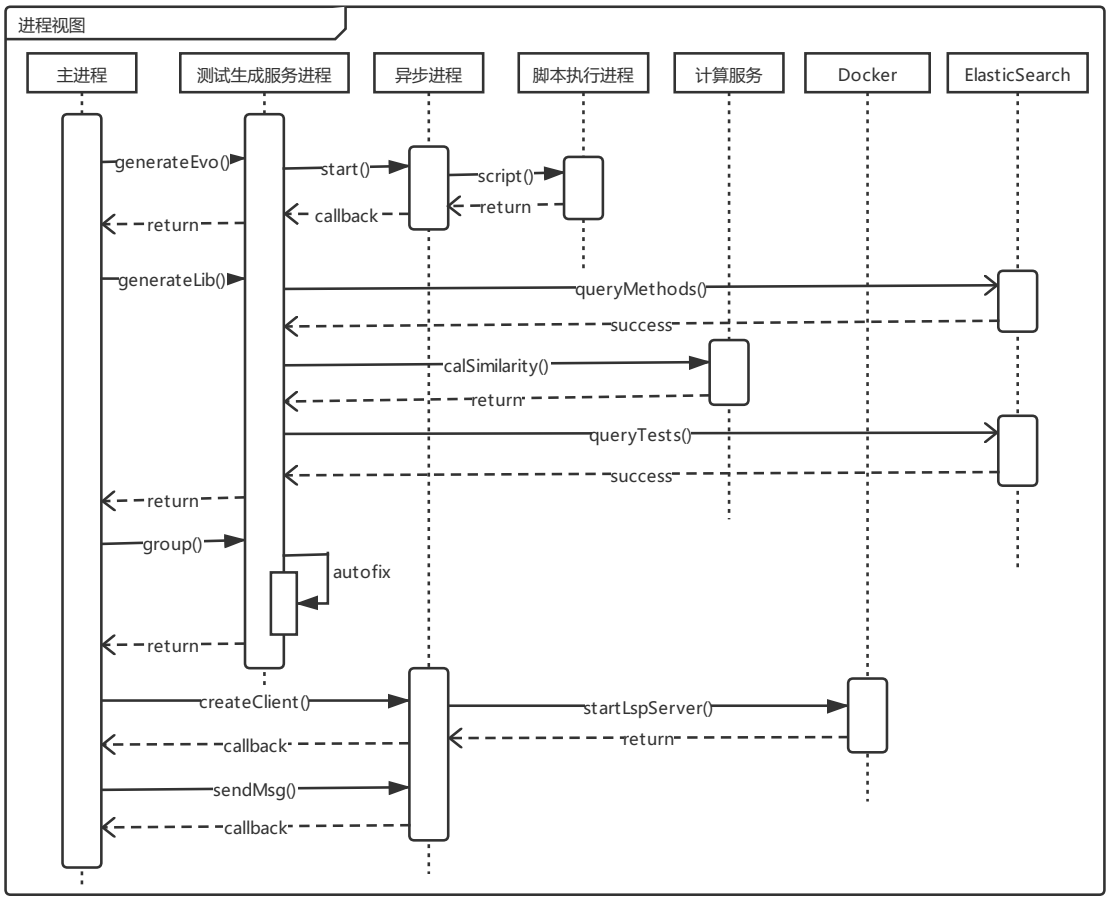


图 3.5: 进程视图

图 3.5是系统的进程视图，从系统集成者角度出发，主要描述了系统的进程、服务及相互间通信。当系统执行测试生成服务时，主进程首先调用测试生成服务进程启动异步进程执行工具脚本。同时测试生成服务进程会从语料库中查询

被测方法结合计算服务进行相似度计算。而后从语料库中查询那些相似度较高的方法的测试用例，通过组合和修复后返回给主进程。当用户触发智能提示功能时，主进程通过异步进程启动相应的 LSPServer，而后主进程和 LSPServer 间通过异步管道进行通信。

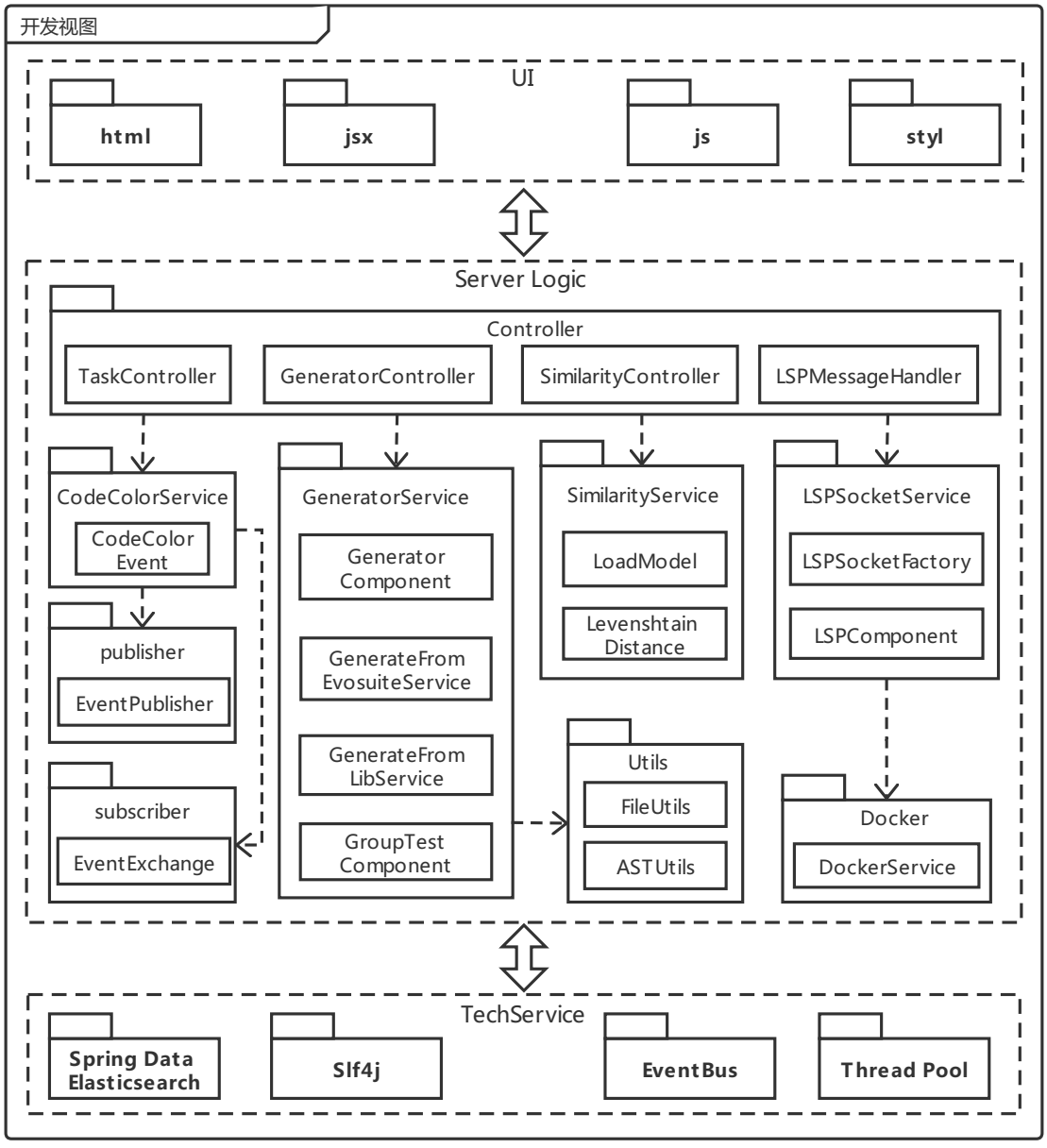


图 3.6: 开发视图

开发视图从开发人员的视角看待系统，对软件开发环境下系统的模块组织和结构管理进行描述。UI 包由 html 页面、jsx 与 js 脚本、以及 styl 样式文件组成。服务端根据系统功能和模块划分。Controller 包负责控制器管理，接收前端

请求并调用服务层处理。CodeColorService 包负责着色信息采集和推送，SimilarityService 包负责词汇相似度分析，GeneratorService 包负责进行测试代码生成，LSPSocketService 包负责提供智能提示。技术服务主要包括 ElasticSearch 接入技术、Slf4j 日志技术、事件发布订阅机制和线程池管理。

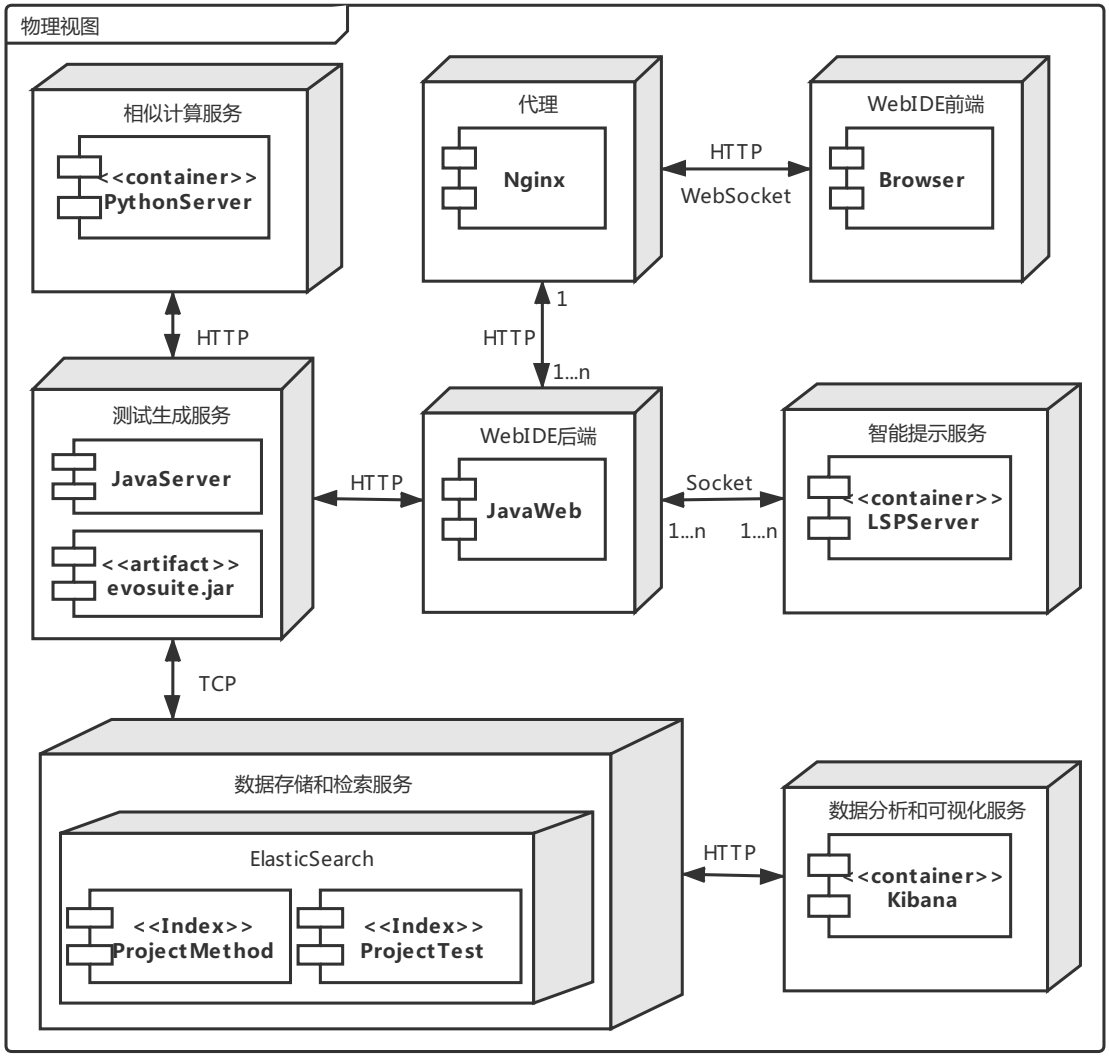


图 3.7: 物理视图

物理视图从系统运维人员的角度出发，对系统各个物理节点的服务部署以及节点间的通信机制进行说明。本系统物理视图如图 3.7所示。用户使用 PC 浏览器能够访问 WebIDE 前端网页，请求经过 Nginx 转发到后端服务器集群。后端节点通过 Socket 管道与 LSPServer 通信，实现智能提示功能。通过 Restful 接口与测试生成服务通信。相似计算服务通过 Flask 被部署为独立的 PythonServer。ElasticSearch 作为数据存储和检索服务使用，Kibana 用于 ElasticSearch 中数据的分析和可视化服务。

3.3.3　测试自动化生成流程总体设计

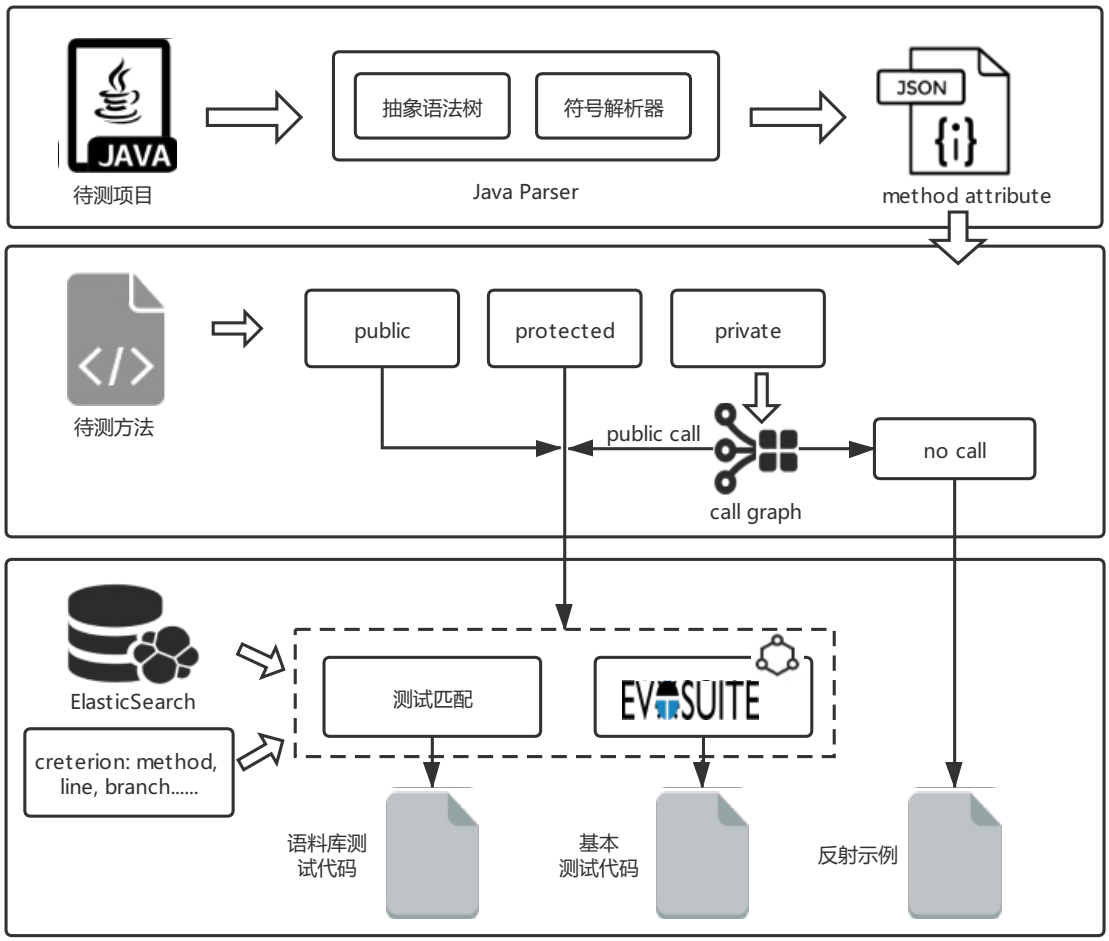


图 3.8: 测试自动化生成整体流程图

图 3.8为测试自动化生成的整体流程设计。自上而下分为三层：程序分析层、策略选择层和测试生成层。程序分析层在用户首次请求测试生成时对待测项目进行程序分析和信息缓存，使用抽象语法树和符号解析器对项目中所有待测类进行方法和调用关系提取，并写入项目缓存文件。策略选择层在收到用户的待测目标方法后，首先根据程序分析缓存文件确定该方法所属类别，即 accessType 为 public、protected 还是 private；由于 private 通常无法被测试方法直接调用，因此需要根据程序调用关系分析是否存在 public/protected 方法调用了该方法；如果存在，则对该调用方法执行生成流程，否则返回反射测试私有方法的示例。代码生成层通过语料库测试匹配和 Evosuite 工具两种方式进行测试生成。首先启动异步的 Evosuite 生成任务并加入任务池，而后尝试从语料库中获取相关测试，如果获取成功则取消任务池中的工具生成任务，否则将工具生成结果返回。

3.4　智能提示模块设计

3.4.1　架构设计

图 3.9为智能提示模块架构图。智能提示模块主要利用微软的 Language Server Protocol 实现系统编辑器页面的代码智能提示功能。为接入 LSP，前端编辑器由 CodeMirror 替换为 MonacoEditor，因为 MonacoEditor 天生可作为 LSP 的客户端。LSPServer 分别采用 palantir 的 python-language-server 和 eclipse 的 eclipse.jdt.ls 作为基础进行改造，并封装为 Docker 容器，对外暴露 Socket 管道用于通信。MonacoEditor 与 WebIDE 后端建立 WebSocket 用于双向通信。WebIDE 后端接收到建立请求后，通过 LSP 适配器根据不同语言启动相应的 LSPServer 容器。容器启动后与 WebIDE 后端建立 Socket 管道，从而实现从 LSP 客户端到服务端的双向通信。

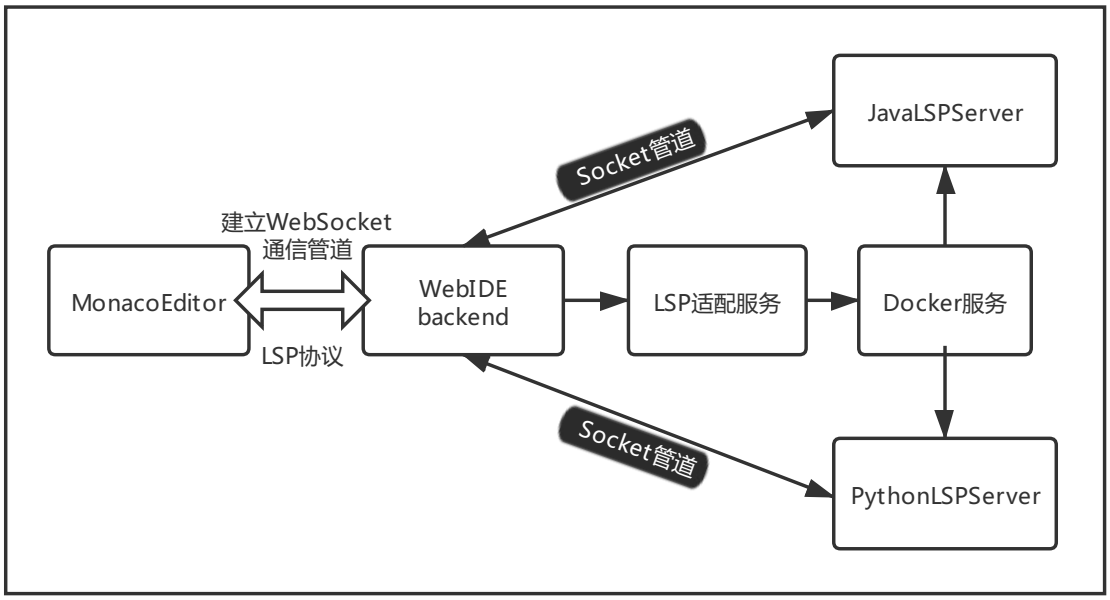


图 3.9: 智能提示模块架构图

3.4.2　核心类图

图 3.10为智能提示模块的核心类图。LSPWebSocketConfig 类配置了 LSP 客户端建立 WebSocket 连接的相关属性，包括授权列表和请求处理器。AbstractLSPMessageHandler 类实现对客户端的请求处理以及对 LSPServer 的建立与销毁，JavaLSPMessageHandler 和 PythonLSPMessageHandler 继承了 AbstractLSPMessageHandler，为不同编程语言保留其特殊实现。LSPSocketService 提供初始化、消息转发和连接断开后的数据清理功能。LSPSocketFactory 接收类型参数，返回对应的 LSPSocket 实例，由于 LSPServer 既可以作为 Socker 客户端，也可以作为

Socker 的服务端，LSPSocketFactory 针对这两种情况进行封装对外暴露了统一的调用方式，因此再接入新的 LSPServer 只需指定新增的对应类型和镜像名称即可。LSPComponent 用于维护异步消息线程，每建立起一个 WebSocket 连接，就会增加一个消息发送和接收的异步线程，用于 LSP 客户端和服务端之间的消息传递。WebSocket 连接断开时，这个线程也就随之结束。DockerService 封装了 DockerClient，负责容器的启动、获取和销毁。

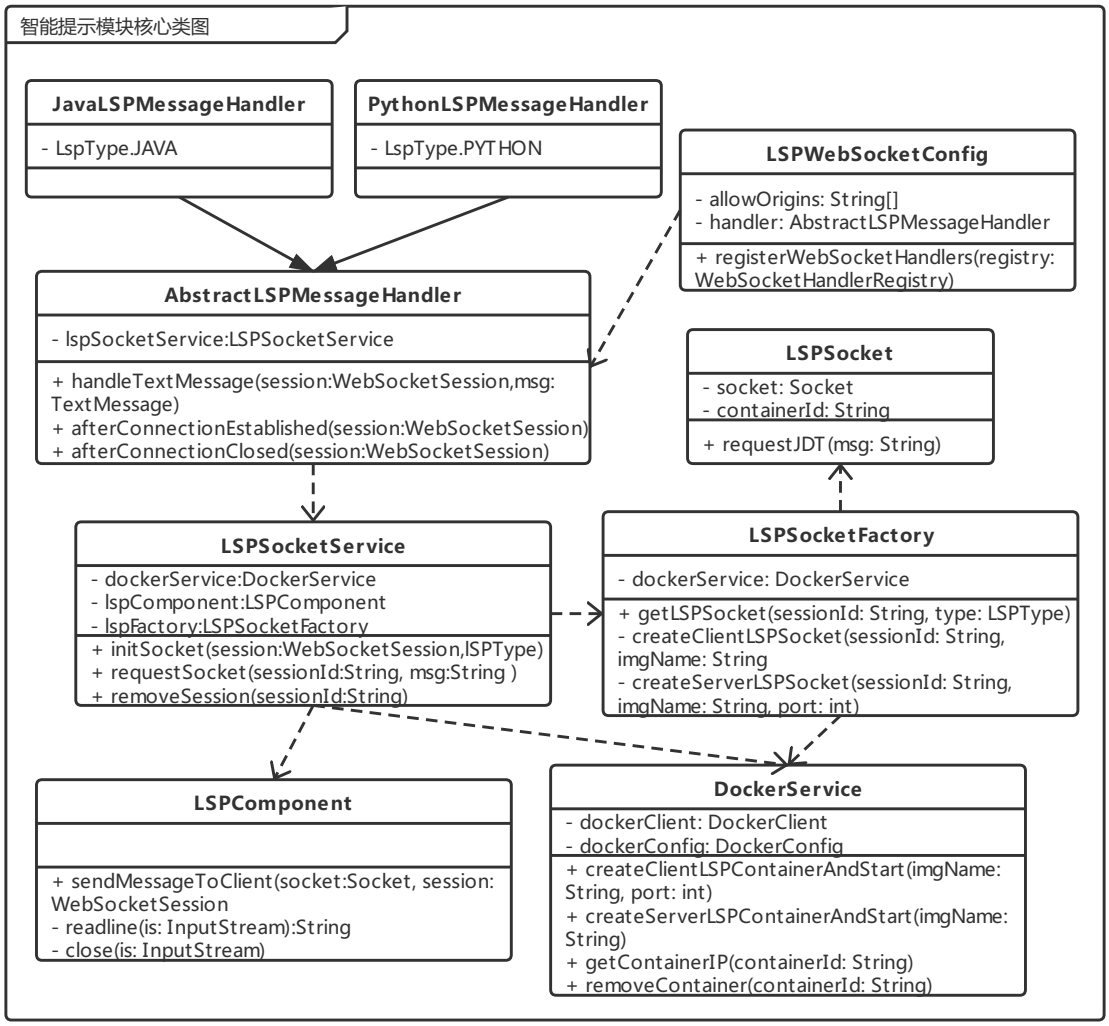


图 3.10: 智能提示模块核心类图

3.5　覆盖可视化模块设计

3.5.1　架构设计

图 3.11为覆盖可视化模块架构图。覆盖可视化模块主要是根据用户编写的测试对源代码的覆盖情况对源代码进行行着色。首先需要对一般的 maven 项目

进行插件配置，以便 OpenClover 能够在编译运行过程中对源码进行插桩，最终统计各个维度（包括方法、行、分支等）的覆盖情况。统计结束后，XMLParser 负责将杂糅在一起的覆盖信息格式化为特定的 Json 文件并存储到项目目录。而后向事件总线 EventBus 发布着色事件，事件的订阅者捕获事件后将着色信息主动推动到 WebIDE 前端，对源代码进行行级着色。未覆盖行为红色，部分覆盖行为黄色，完全覆盖行为绿色，无效行（注释，空行等）不进行着色。

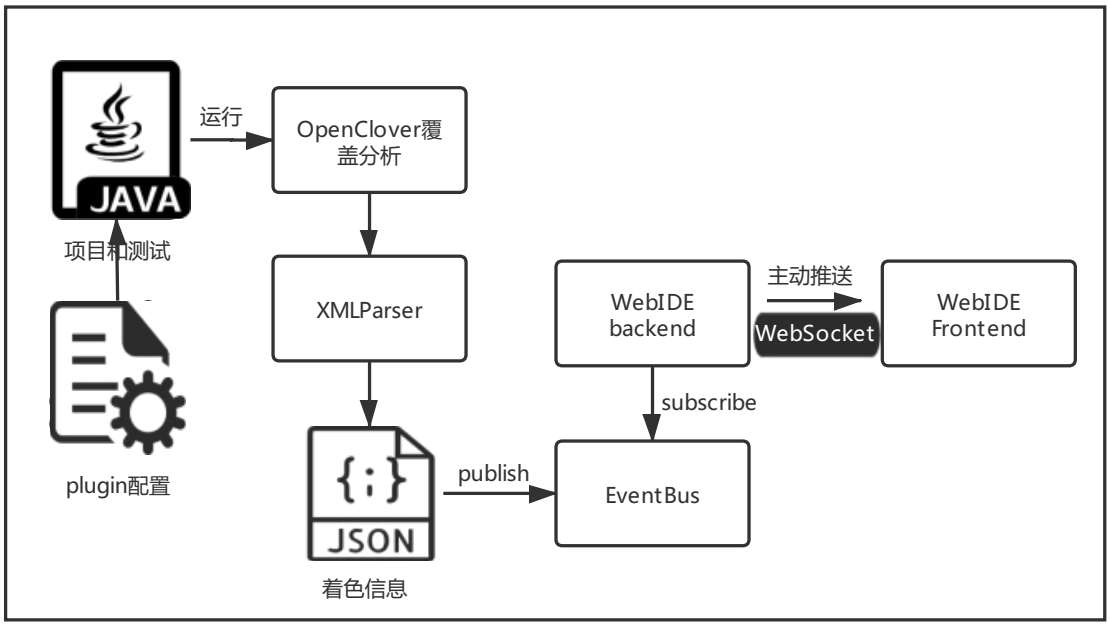


图 3.11: 覆盖可视化模块架构图

```
[
    {
        "path":"hello/service/ALU.java",
        "color":[
            [ 36, 1 ],
            [ 37, 0 ],
            [ 39, 2 ]
        ]
    }
]
```

图 3.12: 着色信息示例

着色信息存储格式如图 3.12所示。每个源代码类被存储为一个列表元素。path 为该类相对项目 Java 文件根目录的位置，color 为该类中所有有效行的着色情况。图中 [36,1] 表示 ALU.java 的第 36 行着色状态为 1，表示第 36 行已被部分覆盖。着色状态 0 表示未覆盖，2 表示完全覆盖。着色状态与颜色的对应由前端自主配置。目前 0，1，2 分别对应红、黄、绿。

3.5.2　核心类图

图 3.13为覆盖可视化核心类图。TaskService 负责项目和测试的运行，并在运行过程中对源码进行插桩和覆盖统计。统计的覆盖信息通过 XMLParser 进行文件格式转换。同时由于覆盖可视化能力是可配置的，所以借助 ProjectService 的 isCodeColor 来判断是否需要进行着色。

如果需要着色，则 CodeColorService 通过 ApplicationEventPublisher 向事件总线发布着色任务 CodeColorEvent，CodeColorEvent 中包含了待着色的项目标识和着色具体信息。EventExchange 作为事件订阅者在接收到事件后通过 SimpMessagingTemplate 向客户端主动推送着色信息。

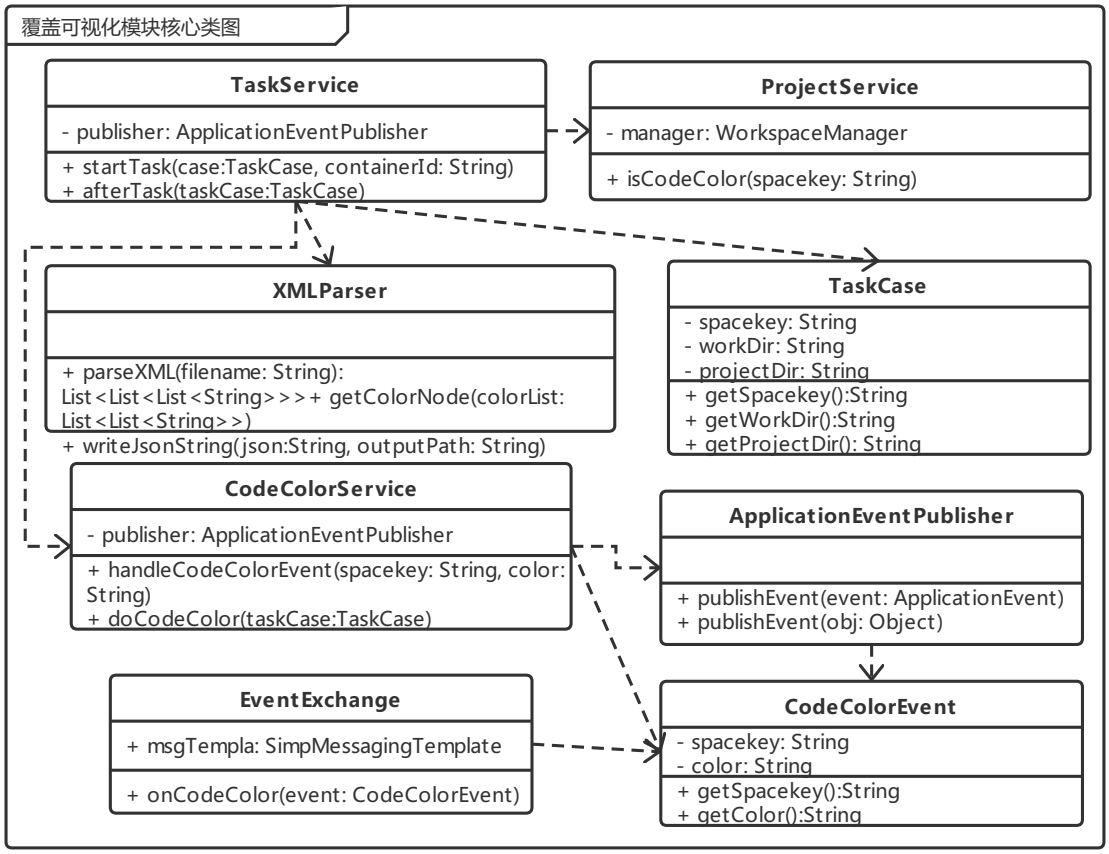


图 3.13: 覆盖可视化模块核心类图

3.6　同源代码匹配模块设计

3.6.1　架构设计

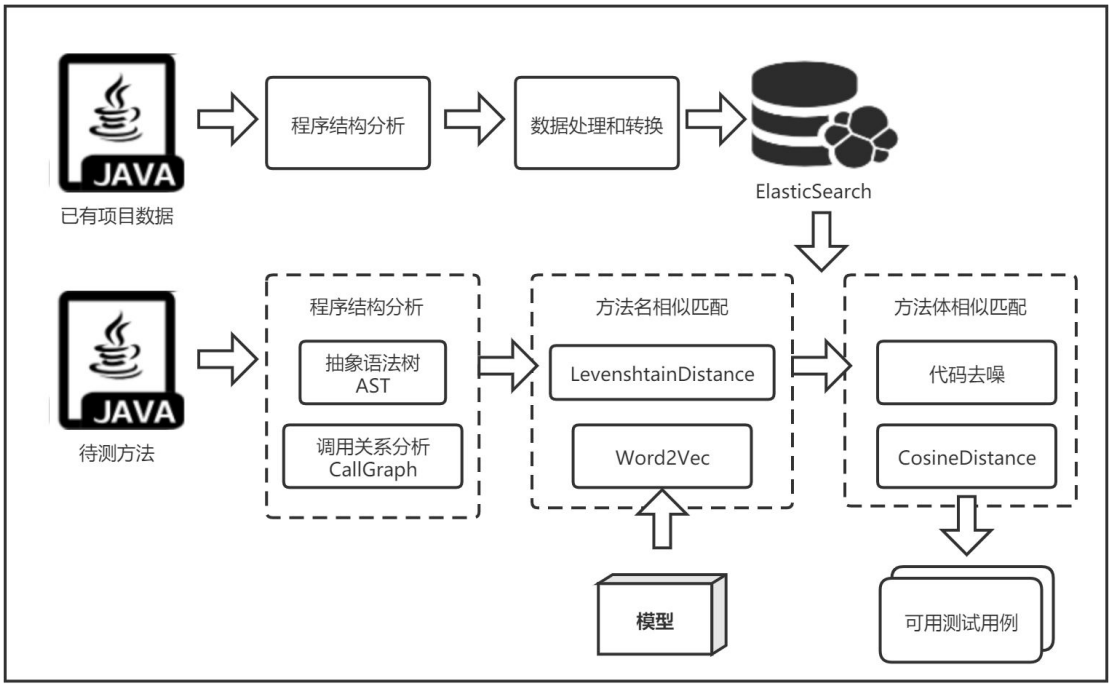


图 3.14: 同源代码匹配模块架构图

同源代码匹配模块架构如图 3.14所示。分为离线数据处理和实时方法相似匹配两个部分。

离线数据处理部分首先对已有项目和测试代码进行程序切片，获取仅调用单个被测方法的测试用例。接着通过程序分析提取项目中所有被测方法的元信息：包括方法名称、参数结构、作用域、行号、所属类等，整合为元方法数据存入ElasticSearch。然后对同项目的测试用例进行调用分析，确认其直接调用的被测方法并与元方法关联，将各个测试的代码及其覆盖行号等信息存入ElasticSearch。

实时方法相似匹配对用户请求进行测试的方法进行程序分析，提取出方法的名称、参数、实现代码等信息。首先通过方法名和结构在语料库中进行完全匹配，利用 Levenshtain 距离的拼写校正来应对常见的方法名拼写错误；然后借助 Word2Vec 加载词向量模型将待测方法名与元方法名进行词义相似匹配；最后将待测方法的方法体代码与前述所得的相似方法方法体进行相似分析排名，主要基于对去噪后的代码的余弦距离度量实现，从而确定与待测方法匹配度最高的元方法。该元方法的测试用例将作为待测方法测试代码生成的直接语料。

3.6.2　核心类图

图3.15展示了数据处理过程的核心类及其相互依赖关系。ProjectMethodConstructService 负责构造元方法并存入 ElasticSearch。ClassParser 对项目目录下所有 Java 文件进行语法分析以提取方法信息。ClassMethodVisitor 继承 VoidVisitorAdapter，visit 方法提供了遍历 MethodDeclaration 的途径。ProjectTestConstructService 负责构造元方法的测试用例并存储。MethodParser 通过 TestMethodVisitor 对测试代码进行语法分析和语法树遍历，主要目的是确定当前测试用例显式调用的元方法。由于调用的形式是多样的，因此 TestMethodVisitor 实现了 getMethodCalls 方法来提取用例中的所有调用语句，包括直接和间接调用。

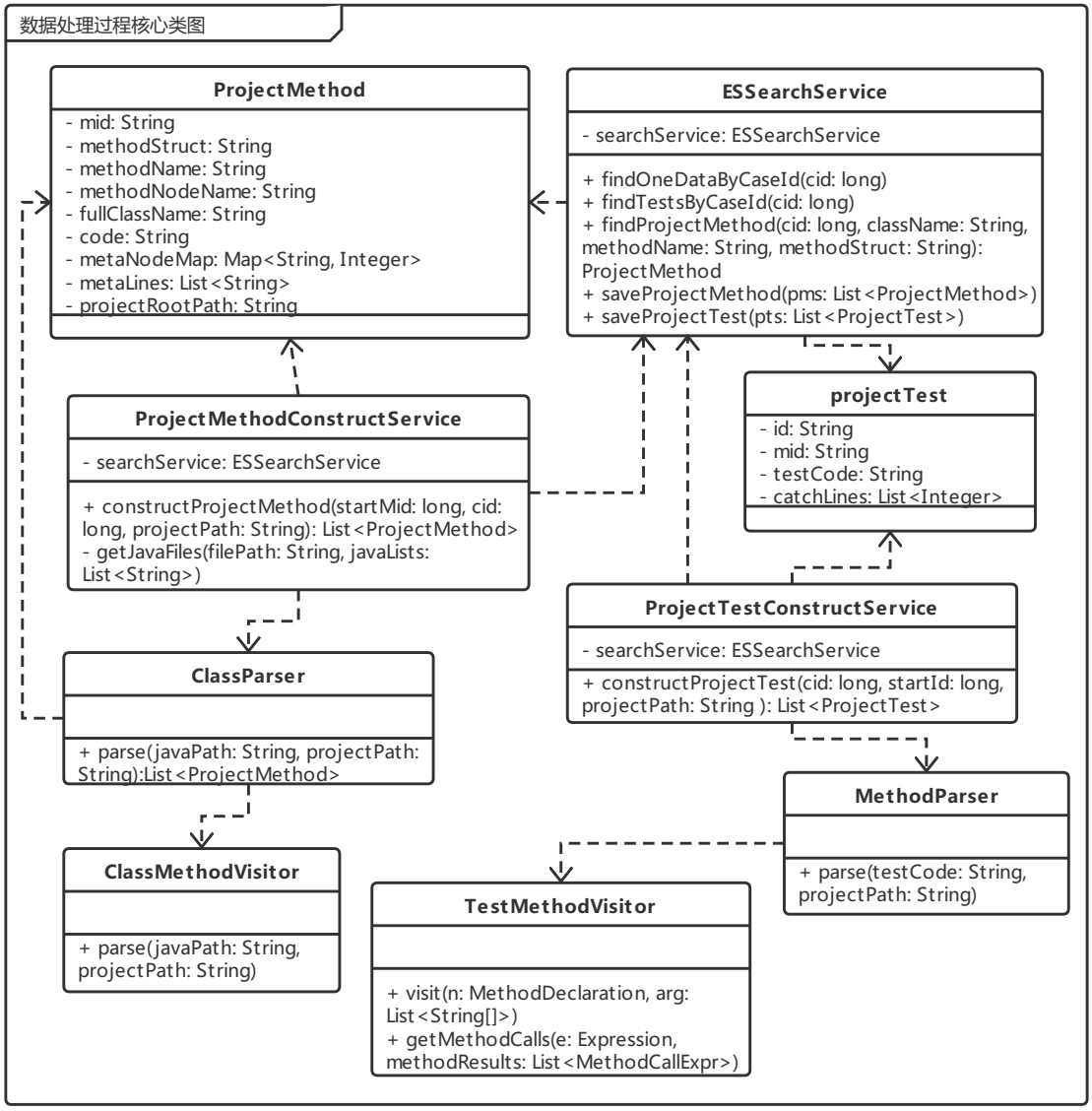


图 3.15: 同源代码匹配模块数据处理过程核心类图

图 3.16展示了相似匹配过程的核心类及其相互依赖关系。GeneratorFromLibService 的 getTestFromLib 方法是待测方法相似匹配的入口。GeneratorServiceComponent 作为程序分析组件，负责转换待测方法抽象语法树并通过 MethodVisitor 进行遍历，从而提取方法名、参数结构、调用关系等信息存储为 MethodAttr 供后续使用。ASTUtils 是语法树分析的工具类，负责将语法树节点信息转换为特定的格式。例如将参数”(String):String” 转换为”(Ljava/lang/String;)Ljava/lang/String;” 这种字节码表示。

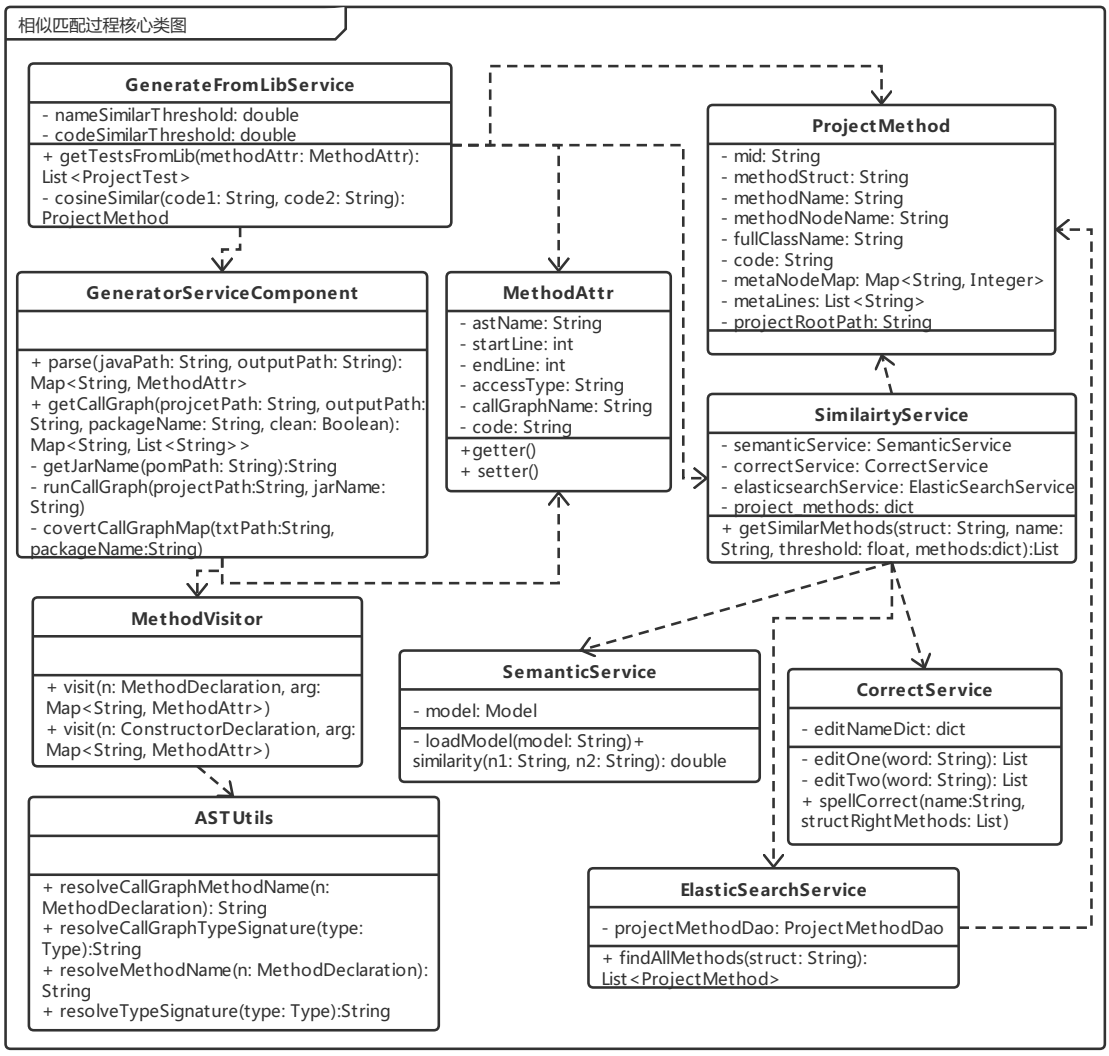


图 3.16: 同源代码匹配模块匹配过程核心类图

SimilarityService 负责方法名之间的完全匹配、拼写校正和语义相似分析。CorrectService 通过对称优化的编辑距离完成拼写校正比对。SemanticService 通过 Word2Vec 加载词向量模型判定方法名之间的语义相似度。consineSimilar 方法负责在确认方法名相似后进行方法体代码之间的相似分析。在方法名相似的

基础上，代码相似度最高的元方法将被认为是待测方法的同源方法，作为搜索测试用例的依据。

3.6.3　数据存储设计

同源代码匹配模块的数据库实体关系图如图 3.17所示。该图描述了在代码匹配过程中涉及的三个实体：Project 代表原项目，ProjectMethod 代表元方法，ProjectTest 代表元方法的测试用例。Project 与 ProjectMethod 为一对多关系，一个软件项目通常包含了多个方法。ProjectMethod 与 ProjectTest 也是一对多的关系，即单个方法可能被多个用例测试。系统在数据处理过程中将已有的项目和测试转换为实体对应的数据并存储，在匹配过程中通过元方法和测试用例检索最终寻找可用测试代码。

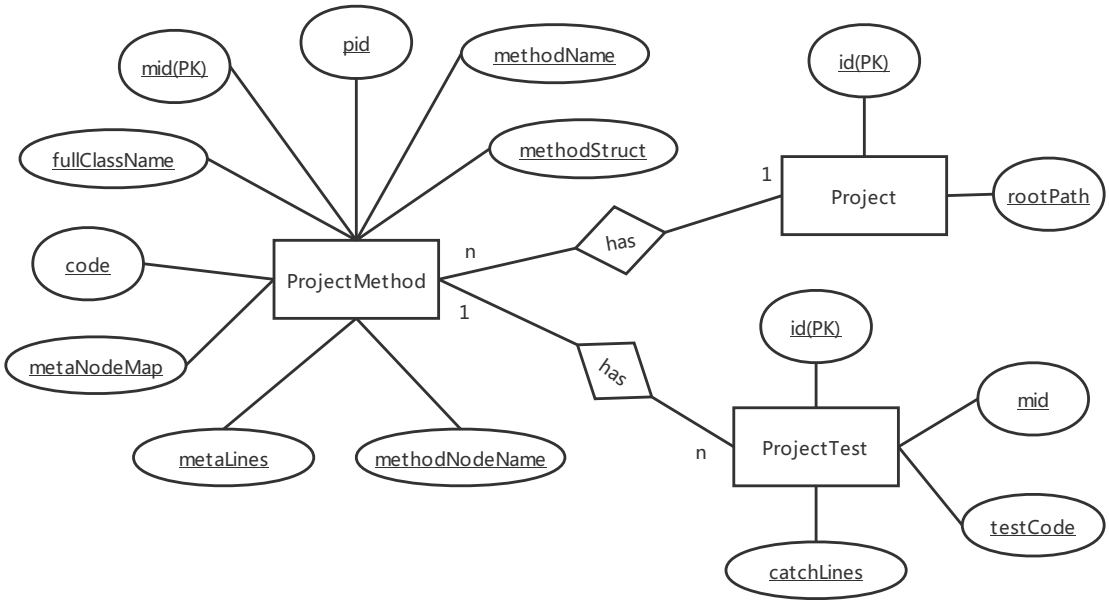


图 3.17: 同源代码匹配模块实体关系图

表 3.10: ProjectTest 索引

字段	字段类型	字段描述
id	keyword	主键，单个测试用例的唯一标识
mid	keyword	被测方法的 id
testCod	text	测试用例的完整代码
catchLines	integer	当前测试用例对于被测方法的有效覆盖行

表 3.10为 ProjectTest 索引的字段说明。该索引保存了测试用例的主要信息，包括唯一标识、被测方法、测试代码以及覆盖行信息。

表 3.11: ProjectMethod 索引

字段	字段类型	字段描述
mid	keyword	主键，单个元方法的唯一标识
pid	keyword	当前元方法所属项目的 id
methodName	keyword	方法的名称
methodStruct	keyword	方法的输入输出结构，形如 (Ljava/lang/String;I)Ljava/lang/String;
methodNodeName	keyword	当前方法在 OpenClover 覆盖统计后的名称
fullClassName	text	当前方法所属类的完全限定名称，形如 org.apache.Parser
code	text	方法体代码
metaNodeMap	text	当前方法在 OpenClover 覆盖统计后包含的所有 node 节点名称
metaLines	integer	当前方法包含的有效行行号集合

表 3.11为 ProjectMethod 索引的字段说明。该索引保存了元方法的主要信息，包括唯一标识、所属项目、方法名、输入输出结构、方法体代码、有效行信息等。

3.7　自动化生成模块设计

3.7.1　架构设计

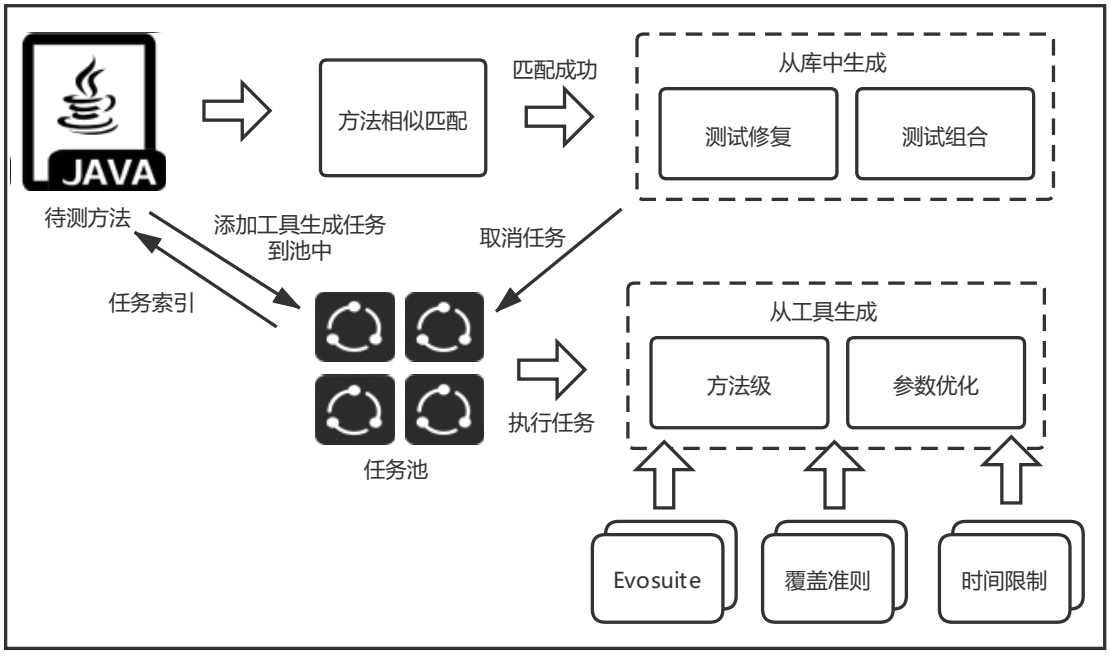


图 3.18: 自动化生成模块架构图

自动化生成模块的架构如图 3.18所示。接收到测试生成任务后，系统首先启动工具生成任务并加入任务池，作为库检索失败的补偿手段。而后进行被测

方法的相似匹配。如果匹配成功则从 ElasticSearch 中查找相似方法的测试用例，并根据需要对测试用例进行修复，例如类的初始化，方法调用的替换等。系统会将同一方法的测试用例组合至同一个测试类中，以便给用户更加便捷直观的体验。如果从库中生成测试类成功（这个过程通常是十分迅速的），则根据之前保留的工具生成任务索引从任务池中删除该任务，防止资源的占用和浪费。如果从库中生成测试类失败，那么工具生成任务将正常执行。主要以 Evosuite 为基础来生成只针对被测方法的测试用例，考虑到最终的效率，覆盖准则选取 line，最长数据搜索时间限制为 20s。

3.7.2　核心类图

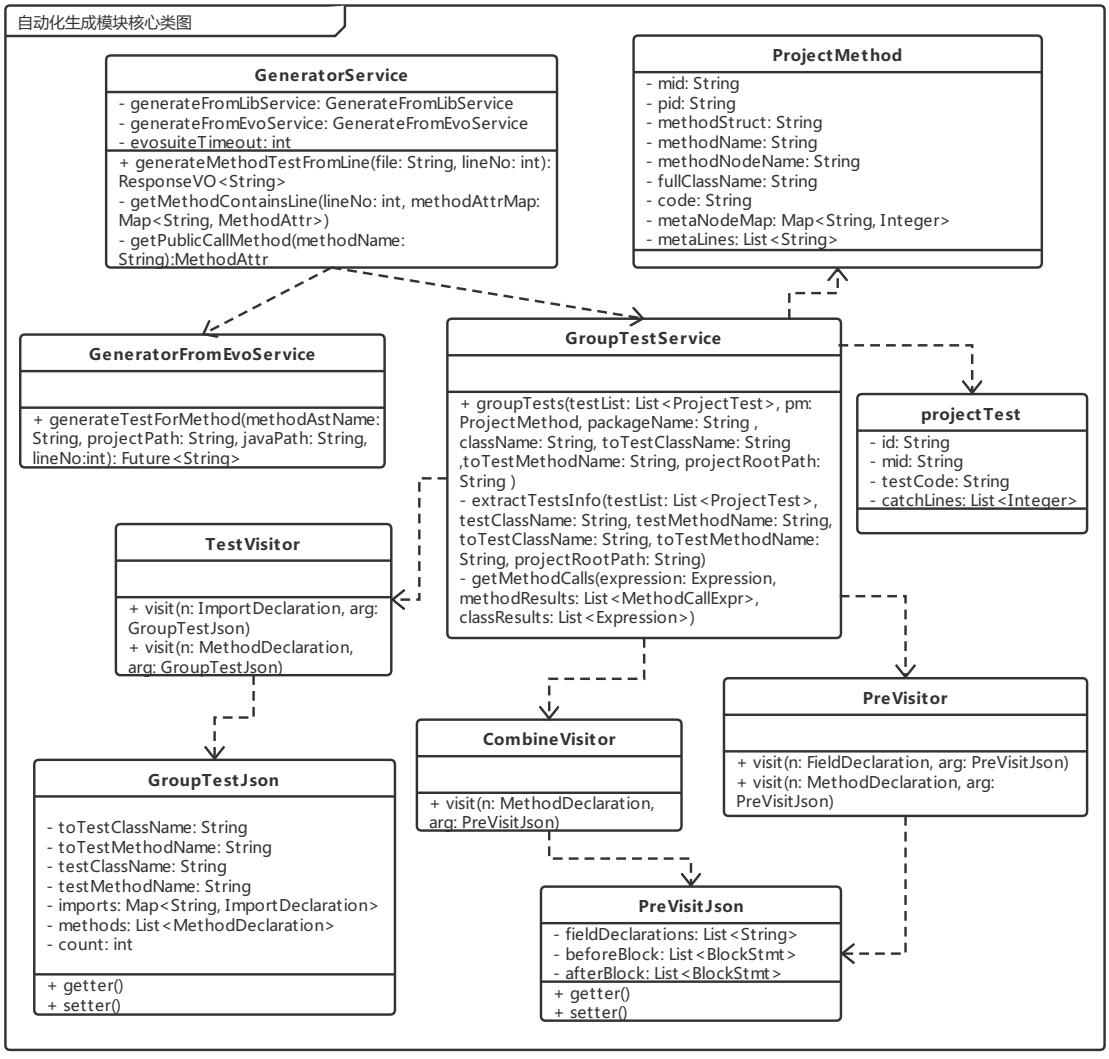


图 3.19: 自动化生成模块架构图

自动化生成模块的核心类图如图 3.19所示。GeneratorService 负责测试生成的统一调度，把控从语料库生成和从工具生成两种方式的执行。GeneratorFromEvoService 通过改造后的 Evosuite 工具配合参数进行基础测试代码的生成。GroupTestService 负责将语料库中检索到的测试用例根据当前测试上下文进行修复和组合。PreVisitor 负责预获取单个测试用例中所有的成员变量声明、统一的前置和后置条件。而后由 CombineVisitor 负责将其全部插入单个测试方法体中。最后 TestVisitor 负责遍历所有测试方法体，根据当前待测方法的上下文进行类名、方法调用等的替换和改造。

3.7.3　参数配置

系统使用 Evosuite 作为工具来生成基本的方法级测试用例，作为语料库中无匹配测试时的补充手段。经过细微改造，Evosuite 的作用范围由只能对整个项目进行测试生成细化到可以对单个方法生成测试代码。系统对一些关键参数进行了优化配置。关键参数配置如表 3.12所示，所有的参数选择都经过基本的实验验证，综合考虑效果、时耗和性能等因素最终确定。系统也预留了参数增删改的空间。此外，除表中所列举的参数外，系统还对诸如输出位置、命名方式、日志分析等进行了个性化配置。

表 3.12: Evosuite 参数配置

参数名称	参数取值	取值理由
criterion	line	该参数是 Evosuite 在生成测试用例的过程中需要尽量满足的覆盖准则。可选准则有 line、branch、exception、weakmutation、method 等。本系统选择 line，因为行覆盖对于用户而言最为直观，生成速度也最快，且基本能够达到 60% 以上的覆盖率
Dcoverage	false	该参数表示是否在生成过程结束后计算此次生成测试用例累计的覆盖率。本系统选择不计算，一方面计算过程需要运行所有测试用例会导致耗时增加，另一方面系统有自身的运行机制向用户展示覆盖率
Djunit_check	false	该参数表示是否对生成的测试用例进行 junit 检查。经过简单实验，99% 的生成都产生了正确的测试代码，处于对效率的考虑，本系统设置为不检查
Dsearch_budget	20	该参数表示生成算法的最大搜索时间，即允许生成测试用例的最长时间，默认为 60s。本系统设置为 20s，经过实验 20s 足够对大多数待测方法产生 60% 以上覆盖率的测试用例，且过长时间会导致较差的用户体验
Dno_runtime_dependency	true	该参数表示生成的测试用例是否不去依赖 Evosuite 的运行环境。系统设置为 true，这样生成的测试用例是标准的 Junit 测试，不需要项目添加额外的运行依赖

3.8　本章小结

本章对基于同源代码匹配的测试开发系统进行整体概述。首先对系统的功能需求、非功能需求以及用例场景等进行了介绍。接着根据整体架构图对系统的总体设计和技术应用展开说明，采用 4+1 视图从多个角度对系统设计进行阐述。而后对系统的四大模块：智能提示模块、覆盖可视化模块、同源代码匹配模块和自动化生成模块展开较为具体的介绍，通过模块架构图、核心类图、存储设计、参数配置等剖析其职责和内部设计。

第四章　测试开发系统的实现

本章基于系统的需求分析和整体架构设计，对系统中各个模块的具体实现展开介绍。利用顺序图描述模块关键组件之间的调用关系，使用关键代码描述实现细节，辅助系统的用户界面展示最终实现效果。

4.1　智能提示模块的实现

4.1.1　顺序图

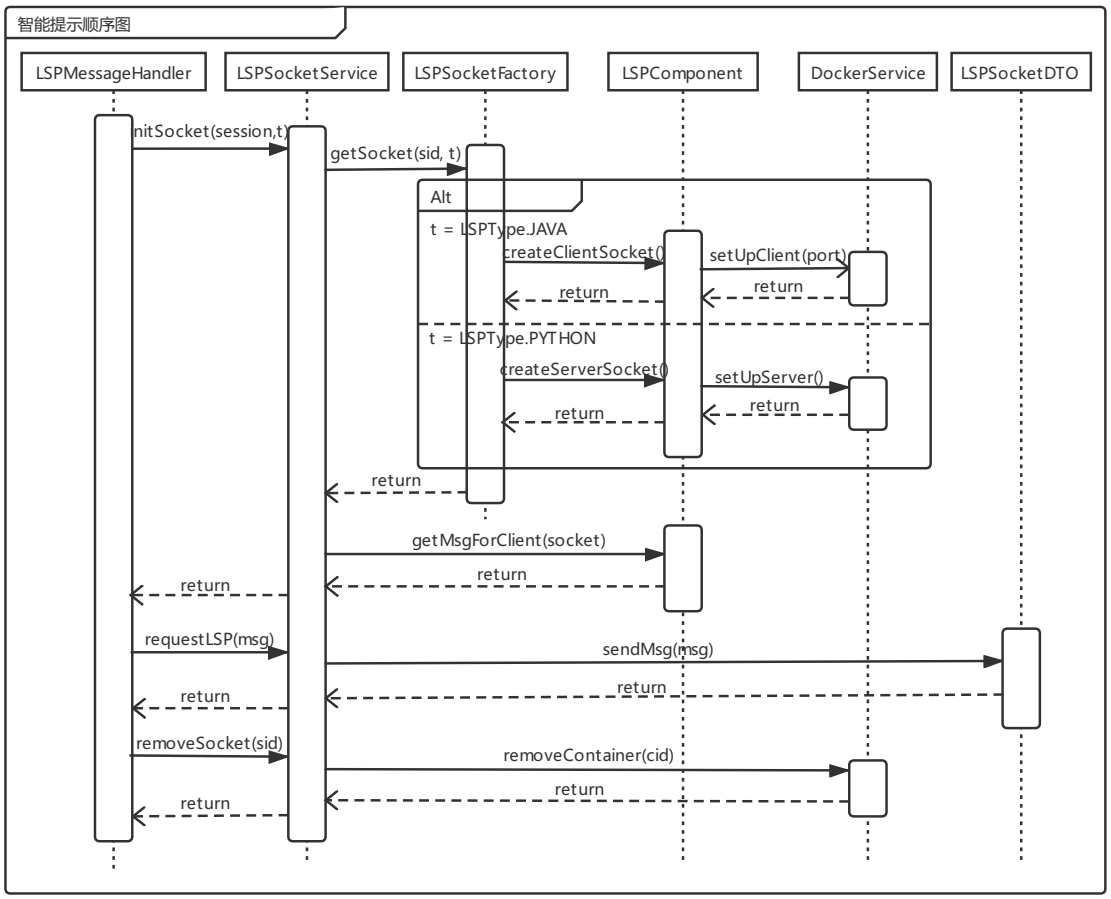


图 4.1: 智能提示模块顺序图

图 4.1是智能提示模块的顺序图。在客户端与服务端建立 WebSocket 连接后，LSPMessageHandler 调用 initSocket 方法获取与 LSPServer 通信的 Socket 实例。LSPSocketFactory 通过对启动的 LSPServer 类型进行区分，按照需要从 LSP-Component 启动 Socket 客户端或者服务端。LSPServer 被封装为 Docker 镜像进

行部署。系统通过 DockerService 控制 Docker 容器的启动、运行和销毁。Socket 通信管道建立后，LSPSocketService 启动一个异步进程通过 sendMessageToClient 方法将 LSPServer 返回的消息返回。

当客户端请求提示时，LSPMessageHandler 通过 requestLSP 对当前的 Socket 通信管道发起请求，由 LSPSocketDTO 对象的 sendMsg 方法负责同步发送消息。

客户端与服务端的 WebSocket 连接断开时，LSPMessageHandler 通过 removeSocket 方法进行资源释放，包括 Docker 容器的销毁，输入输出流的关闭等。

4.1.2　关键代码

```
public LSPSocketDTO getLSPSocket(String sessionId, LSPType type) throws Exception{
        switch (type){
            case JAVA:   //创建 java 语言对应的 LSPServer
                lspSocket = createClientSocket(sessionId, "lsp-java"); break;
            case PYTHON: //创建 python 语言对应的 LSPServer
                lspSocket = createServerSocket(sessionId, "lsp-python", 2087); break;}
        return lspSocket;
}
private LSPSocketDTO createClientSocket(String sessionId, String imageName){
        ServerSocket serverSocket = new ServerSocket(0);
        Future<String> cidFuture = setUpClientLSPServer(serverSocket.getLocalPort() + "",
imageName); // 异步启动作为 Socket 客户端的 LSPServer
        Socket socket = serverSocket.accept();//调用 accept 方法，等待 socket 客户端连接
        return new LSPSocketDTO(socket, cidFuture.get(10, TimeUnit.SECONDS));
}
private LSPSocketDTO createServerSocket(String sessionId, String imageName, int port) {
        String containerId = setUpServerLSPServer(imageName); //启动服务端的 LSPServer
        for(int i=0; i<3; i++){ //连接 Socket，最多重试三次
          try{ Socket socket = new Socket(dockerService.getContainerIP(containerId), port);
              return new LSPSocketDTO(socket, containerId);
          }catch (Exception e){    Thread.sleep(1000);     }   }
}
public synchronized void requestMsg(String message) throws Exception {
        String content = header+split+message; //拼装 jsonrpc 消息体
        os.write(content.getBytes(StandardCharsets.UTF_8)); os.flush();//写入管道输出流
}
```

图 4.2: 智能提示模块代码

图 4.2展示了创建 LSPServer 以及向其发送消息的主要实现代码。getLSP-Socket 接收 sessionId 和 lspType 两个参数，sessionId 标识了当前所属的 WebSocket 连接，lspType 确定需要启动的 LSPServer 类型，返回的对象 LSPSocketDTO 持有与 LSPServer 进行通信的管道引用。createClientSocket 和 createServerSocket 分别展示如何启动作为 Socket 客户端和服务端的 LSPServer，其中包含了异步和重试机制。

requestMsg 方法属于 LSPSocketDTO 对象，synchronized 限制确保同一个通信管道中必然是前一条消息发送完毕后方可发送下一条消息。通信协议遵循 LSP 的 JSONRPC 格式。

4.1.3　页面展示

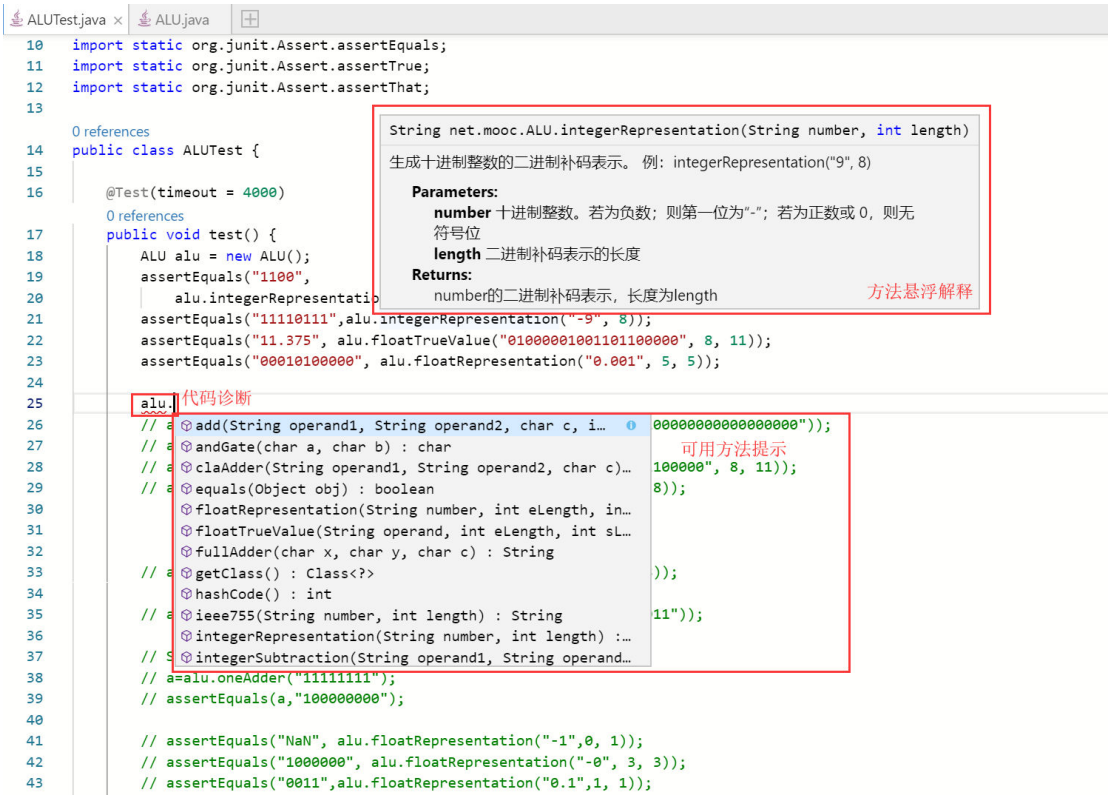


图 4.3: 智能提示功能页面展示

图 4.3展示了智能提示功能使用时的一种情形。用户编写”alu.” 后，页面展示出 alu 变量的可调用方法列表供选择；此时”alu.” 是不完整的语句，因此页面对其下划红色波浪线进行突出标识；将鼠标悬停在”alu.integerRepresentation” 方法上，页面悬浮框显示该方法的 Javadoc 供用户参考。

4.2　覆盖可视化模块的实现

4.2.1　顺序图

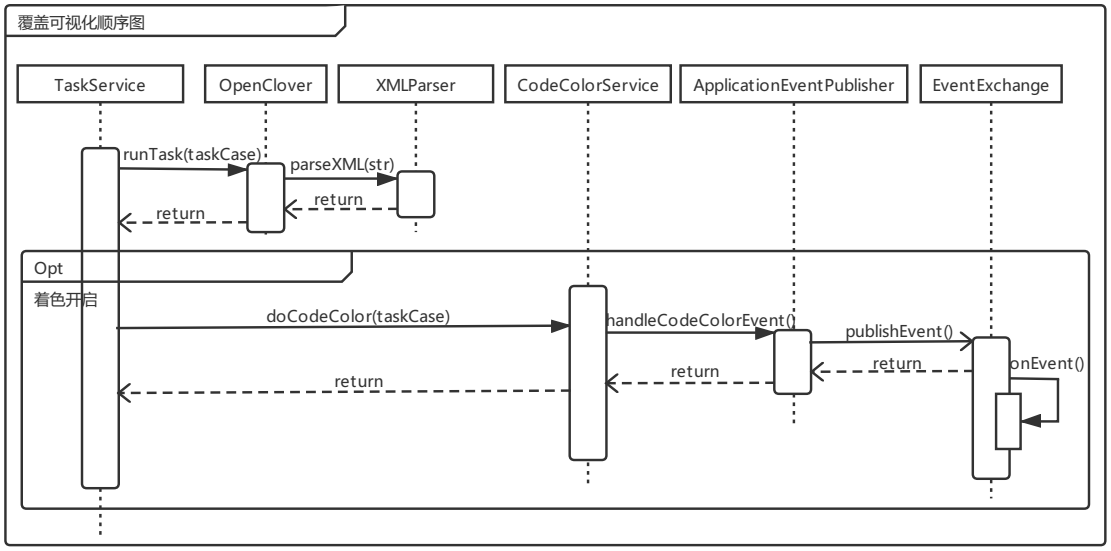


图 4.4: 覆盖可视化模块顺序图

图 4.4是覆盖可视化模块的顺序图。用户运行项目后，系统使用 OpenClover 进行源码插桩统计覆盖信息。而后通过 XMLParser 将原始覆盖信息转换为特定格式的 json 并存储到项目指定文件。项目运行结束后，系统通过 ProjectService 获取是否需要进行着色。如果需要着色，CodeColorService 将通过 Application-EventPublisher 向事件总线发布着色任务。EventExchange 作为事件的消费者，在接收到着色事件后通过 onEvent 回调将着色信息推送到客户端进行页面渲染。

4.2.2　关键代码

图 4.5展示了着色事件发布与订阅处理的实现代码，这一过程在项目运行之后。doCodeColor 负责读取着色信息，handleCodeColorEvent 通过 Application-EventPublish 进行着色事件的发布，@EventListener 注解提供了事件监听能力，onCodeColor 在接收到 CodeColorEvent 后自动触发，利用 SimpMessagingTem-plate 通过 WebSocket 管道主动向客户端推送着色信息。系统采用发布订阅机制的主要原因在于，着色必须在项目运行之后，而项目运行极其耗费时间，因此系统采用异步线程池进行实现，这就要求必须将着色与客户端请求解耦，同时又要确保着色完成后客户端能够感知，因此将发布订阅机制与 WebSocket 技术相结合以实现这一需求。

```
public void doCodeColor(TaskCase taskCase) {
        String codeColorString = FileUtil.readFile(file); //读取着色信息
        this.handleCodeColorEvent(taskCase.getSpaceKey(),codeColorString);
}
public void handleCodeColorEvent(String spacekeys, String codeColorString) {
        applicationEventPublisher.publishEvent(new
                        CodeColorEvent(this,spacekeys,codeColorString));//  发布事件
}
@EventListener //事件监听
public void onCodeColor(CodeColorEvent event) {
        String result = new Gson().toJson(event.getCodeColorString());
        simpMessagingTemplate.convertAndSend("/topic/codeColor/"+spaceKey+"/info",
            result); //通过 websocket 向客户端推送着色信息
}
```

图 4.5: 着色事件发布于处理实现代码

4.2.3　页面展示

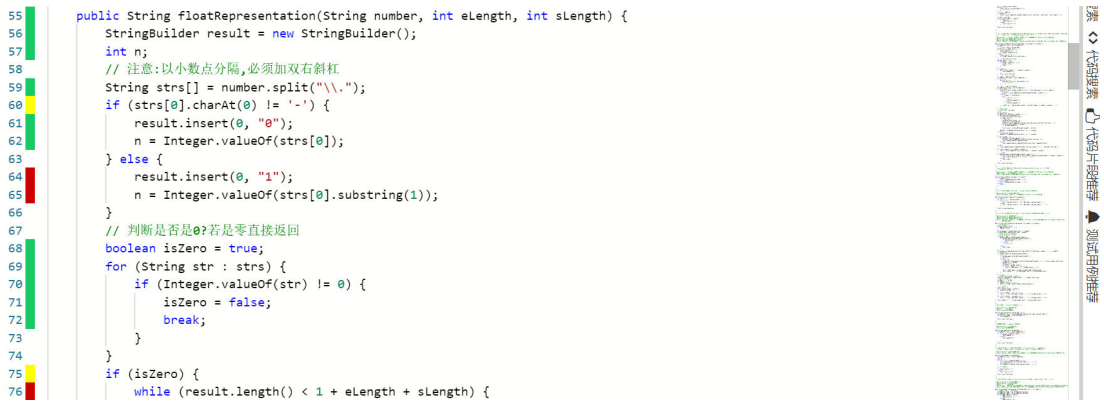


图 4.6: 覆盖可视化功能页面展示

图 4.6为覆盖可视化功能的页面展示样例，行号后的颜色代表了每行源代码当前的覆盖情况。红色的行表示未被覆盖；黄色的行多为分支节点，表示仅有部分分支被覆；绿色的行表示已被完全覆盖。对于无效行，如空行、注释等不进行着色渲染。

4.3　同源代码匹配模块的实现

4.3.1　顺序图

图 4.7是同源代码匹配模块数据处理过程的顺序图。ProjectMethodConstruct-Service 的 parse 方法负责源代码解析。ClassParser 通过 ClassMethodVisitor 对源代码转换的抽象语法树进行遍历，获取所需信息后 construct 方法负责构造 ProjectMethod 并存入 ElasticSearch。ProjectTestConstructService 的 parse 方法负责测试代码解析。MethodParser 通过 TestMethodVisitor 对测试代码转换的抽象语法树进行遍历，获取当前测试代码直接调用的被测方法的所属类、名称和参数信息。而后利用这些信息从 ElasticSearch 中检索对应元方法 id，最后组合出测试用例信息构造 ProjectTest 并存储。

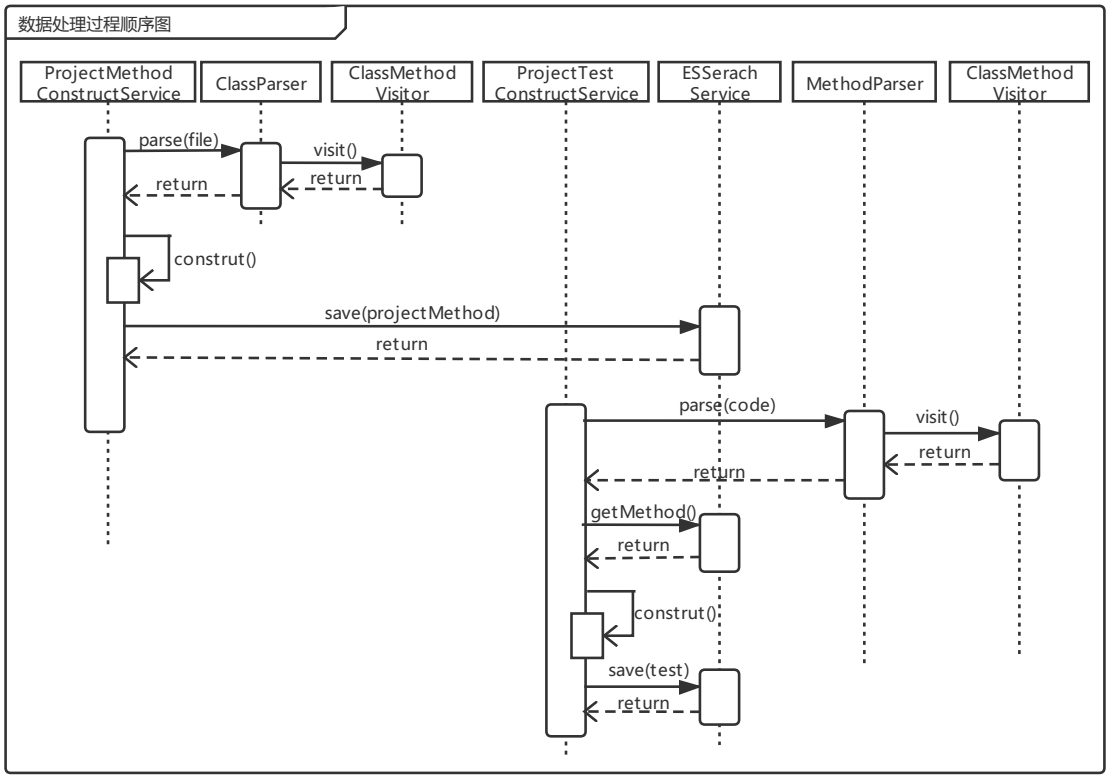


图 4.7: 数据处理过程顺序图

图 4.8是同源代码匹配模块相似匹配过程的顺序图。GenerateFromLibService 作为相似匹配的起点，通过 GenerateComponent 对待测方法进行程序分析，提取方法信息。而后 SimilarityService 负责寻找与待测方法相似的元方法，通过 ElasticService 从语料库中检索出所有元方法：首先进行方法名和方法结构的完全匹配；如果匹配失败，则调用 CorrectService 进行拼写校正匹配；如果拼写校正

仍未成功，则 SemanticService 负责加载词向量模型从元方法中匹配与待测方法名语义相似的方法，然后将名称匹配成功的元方法返回；GenerateFromLibService 获得名称相似的元方法列表后，再将待测方法的代码与列表中的方法体代码逐一比较，最终将代码相似度最高的元方法确定为被测方法的同源方法。

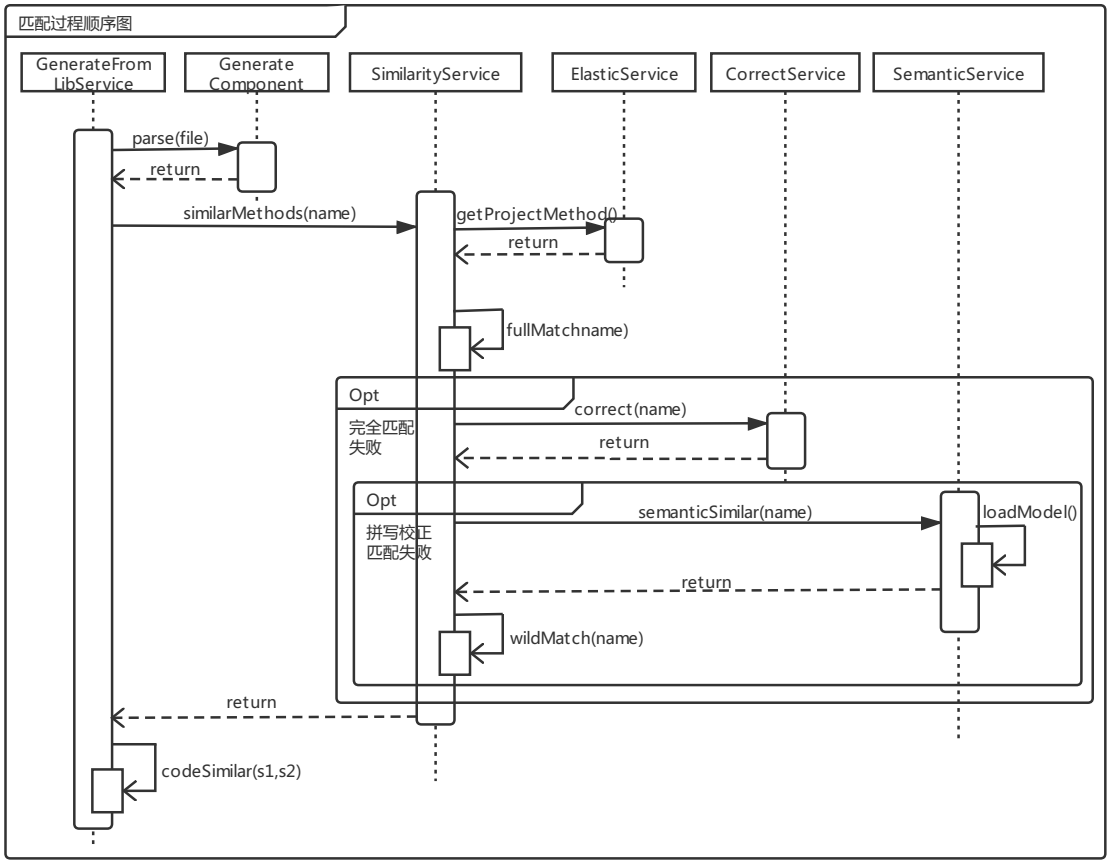


图 4.8: 方法匹配过程顺序图

4.3.2　关键代码

数据处理过程的元方法构造实现代码如图 4.9所示。主要展示了从解析源代码到属性设置，最后批量写入 ElasticSearch 的过程。ClassParser.parse 方法将类代码解析为抽象语法树，并通过遍历语法树返回类中包含的方法信息。metaNodes 是每个方法包含的覆盖点信息，通过检查 metaNode 将覆盖点信息对应到代码行。元测试的构造实现代码与此类似，主要区别在于解析的代码为测试代码，且解析的主要目标是获得当前测试代码直接调用的方法信息，再利用这一信息将测试代码与语料库中的元方法进行关联。

匹配过程的方法名相似匹配实现代码如图 4.10所示。similarMethods 方法首先使用 compeleteMatch 方法进行方法名和方法结构的完全匹配；如果返回结果

数为 0，则通过 spellCorrectES 进行拼写校正，放宽方法名限制；如果仍未能获得结果，则在方法结构匹配的基础上获取 structRightMethods，通过 gensim 加载 word2vec 的词向量模型将待测方法的名称与 structRightMethods 集合中的方法名称进行语义相似分析，语义相似度最高且达到阈值的 10 个方法将加入结果集；同时通过对方法名添加通配符进一步放宽限制，从语料库中检索最多 10 个方法，与语义匹配的方法列表合并后返回。

```
public void constructProjectMethod(long startMid, String projectPath) {
        for(String javaPath: javaPathList){
                List<ProjectMethod>  projectMethodList  =  ClassParser.parse(javaPath,
projectPath); // 解析方法源代码
                for(ProjectMethod pm : projectMethodList){
                    String fullClassName = javaPath.replace(projectPath,"".replace("/",".")
                            .replace(".java","");
                    // 设置属性
                    pm.setCaseId(caseId); pm.setFullClassName(fullClassName);
                    pm.setProjectRootPath(projectPath); pm.setMid(startMid+"");startMid++;
                    // 初始化列表参数
                    metaNodes.forEach(mn -> {
                        String name = mn.getName(); String location = mn.getLocation();
                        // 名称包含，且类名一致
                        if(name.contains("-"+pm.getMethodNodeName())&&
   location.startsWith(pm.getFullClassName().replace(".","/"))){
                                int index = location.lastIndexOf("/");
                                String line = location.substring(index+1);
                                String  lineNoStr  =  line.contains("-")?line.substring(0,
line.indexOf("-")):line;
                                int lineNo = Integer.parseInt(lineNoStr);
                                // 写入 metaNodeList
                                pm.getMetaNodeMap().put(name, lineNo);
                                // 写入待覆盖行
                                if(!pm.getMetaLines().contains(lineNo)){
                                    pm.getMetaLines().add(lineNo); }} });}
                allMethods.addAll(projectMethodList); }
            projectMethodDao.saveAll(allMethods); // 存储到 ElasticSearch
}
```

图 4.9: 元方法构造实现代码

```
def similarMethods():
    # 首先进行方法名+方法结构的完全匹配
    methods = fullMatch(methodStruct, methodName)
    if len(methods) != 0:    return jsonify(methods)
    # 完全匹配未获得结果，进行拼写校正匹配
    methods = spellCorrect(methodStruct, methodName)
    if len(methods) != 0:    return jsonify(methods)
    # 拼写校正也未能获得结果，则在方法结构匹配基础上，进行方法名语义相似匹配
    structCheckResult = es.search(index=INDEX_PROJECT_METHOD,
    body = { "query": {"wildcard": {"methodStruct": { "value": methodStruct}}},"size": 1000})
    structRightMethods = structCheckResult['hits']['hits']
    if len(structRightMethods) == 0:    return jsonify([]) #没有结构匹配的方法直接返回空
    for hit in structRightMethods:
        method = hit['_source']
        sim = float(model.similarity(methodName, method['methodName']))
        methods.append(method)
     # 取语义相似度最高的方法，最多 10 个
     methods.sort(key="sim", reverse=True)
     if len(methods) > 10:
        methods = methods[0:10]
    # 通配符匹配
    methods.extend(wildcardSearch(methodStruct, methodName))
    return jsonify(methods)
```

图 4.10: 方法名相似匹配实现代码

4.4　自动化生成模块的实现

4.4.1　顺序图

测试自动化生成的调用顺序如图 4.11所示。GeneratorService 负责统筹整个流程。当用户请求测试生成时，GeneratorService 通过 GenerateFromEvoService 启动异步的基础用例生成任务，并返回该任务的引用。

GenerateFromLibService 将尝试从语料库生成测试代码。对于从库中检索出的相似元方法的测试用例列表，根据其对元方法能否产生新的覆盖确定是否添加到可用结果集。而后将产生的可用测试结果集组装到一个测试类中。PreVisitor 负责收集每个测试用例的前置后置条件，CombineVisitor 将这些条件与测试方法中的代码进行组合，TestVisitor 负责对组合后的新的测试代码进行变量和调用替换。由于待测方法与元方法是相似关系而非完全相同，为最大程度保证测试代

码可执行，需要对测试方法中涉及元方法调用的部分进行替换，包括直接和间接调用，TestVisitor 的 getCall 方法使用递归遍历测试代码中的所有表达式，返回所有的方法调用。

如果 GenerateFromLibService 能够产生测试代码，则 GeneratorService 停止最开始启动的基础用例生成任务。否则等待基础用例生成任务执行完毕并将结果返回给用户。

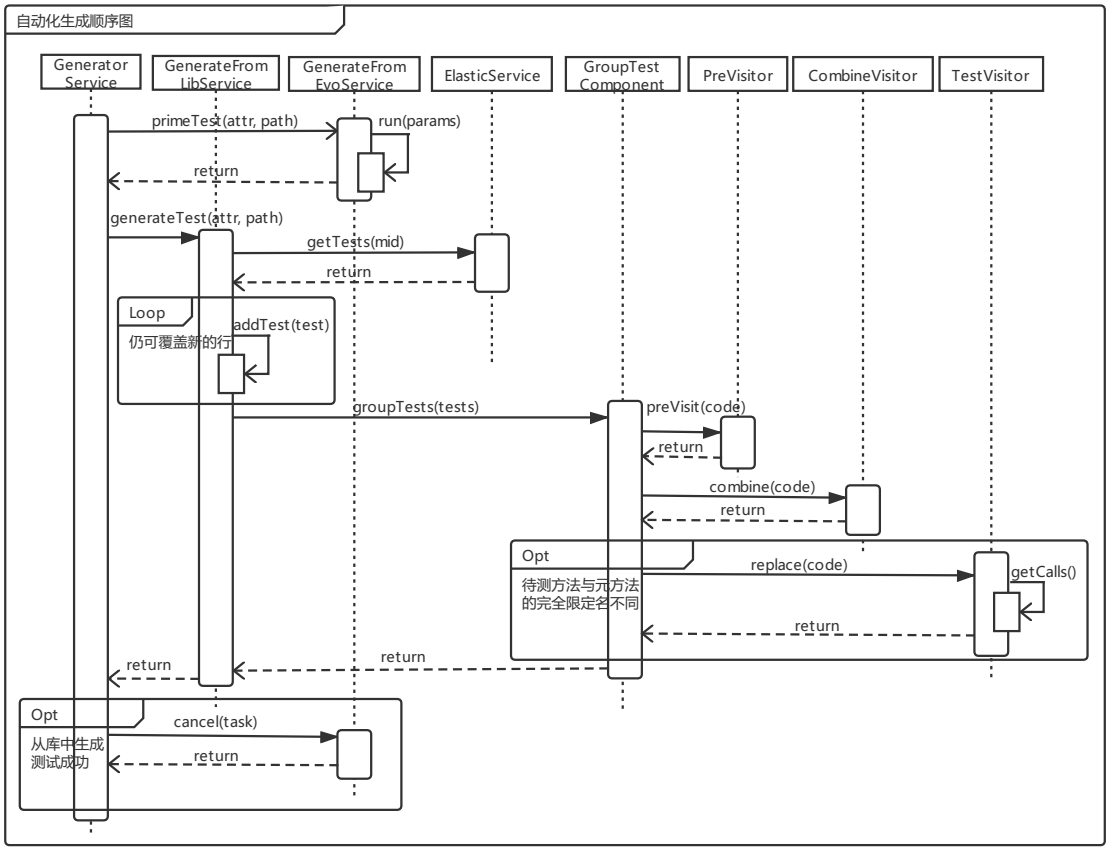


图 4.11: 自动化生成顺序图

4.4.2　关键代码

图 4.12展示了自动化生成模块的部分关键代码。gnerateMethodTest 是测试生成的调度方法，接收待测文件路径和行号两个参数，行号用来确定待测方法的位置。如果 generateFromLibService 生成测试成功，则返回生成的测试代码所属文件路径。此时 generateMethodTest 通过之前保存的 primeTest 的引用，终止该基础用例生成任务，并将其从任务池中移除。如果 generateFromLibService 生成测试失败，则返回 null, generateMethodTest 接收到 null 后会等待 primeTest 执行完毕并获取结果。addTests 展示了 GenerateFromLibService 筛选测试结果集的

过程。tests 为相似元方法对应的所有测试用例，筛选标准为如果新的测试用例能够覆盖新的代码行，则加入结果集，否则丢弃。遍历的终止条件为元方法行被全部覆盖或用例列表全部被访问。

```
public ResponseVO generateMethodTestFromLine(String fileName, int lineNo) {
        Future<String> evosuiteFuture = generateFromEvoService.generateTestForMethod
(methodAstName, projectPath, javaPath, lineNo); // 启动基础用例生成任务
            runningGeneration.put(key, evosuiteFuture); }//加入任务池
        String testFilePath = generateFromLibService.generateTestFromLib(methodAttr,
projectPath, javaPath); // 尝试从库中生成测试代码
        if(testFilePath != null){ // 库中生成测试代码，终止基础用例生成任务
            Future<String> ef = runningGeneration.get(key);
            ef.cancel(true);
            runningGeneration.remove(key);
            return new ResponseVO<>(ServerCode.SUCCESS, testFilePath); }
        Future<String> ef = runningGeneration.get(key); //库中未能生成，返回基础用例
        String testFilePathEf = ef.get(evosuiteTimeout, TimeUnit.MINUTES);
        runningGeneration.remove(key); // 结果已获取，从任务池中移除任务
        return new ResponseVO<>(ServerCode.SUCCESS, testFilePathEf);
}
private List<ProjectTest> addTests(List<ProjectTest> tests){ // 测试结果集筛选
        List<ProjectTest> resultTests = new ArrayList<>();
        for(ProjectTest pt:tests){
            if(metaLines.size()==0)  break;  //没有未覆盖行
            List<Integer> catchLines = pt.getCatchLines();
            List<Integer> copyMetaLines = new ArrayList<>(metaLines);
            if(copyMetaLines.retainAll(catchLines)){  //如果当前测试仍能覆盖新的行
                resultTests.add(pt);
                metaLines.retainAll(catchLines); }}
        return resultTests;
}
```

图 4.12: 自动化生成关键代码

4.4.3　页面展示

图 4.13展示了用户触发测试生成的方式，用户进入 WebIDE 时，系统会提示将鼠标悬停在源代码行号处 2 秒触发测试生成入口，即“生成方法级测试用例”，点击该按钮，系统将为该行所属方法进行测试代码生成。将鼠标移开 2 秒后，按钮自动消失。

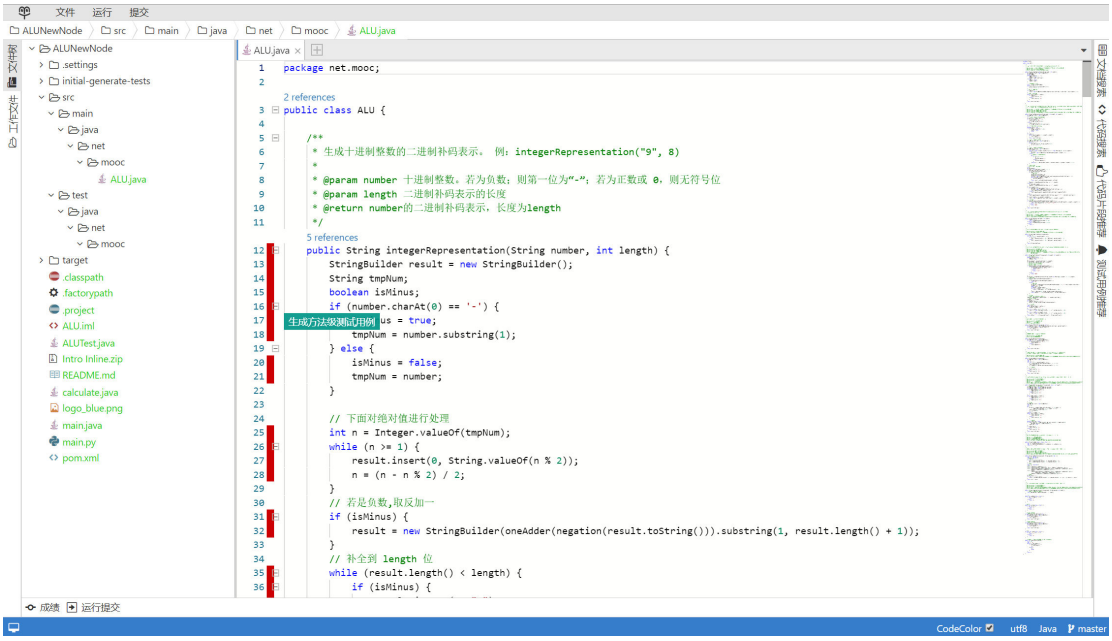


图 4.13: 触发测试生成功能

图 4.14展示了一个工具生成基础测试用例的示例，左侧的文件目录下出现了新的测试文件。测试文件中包含了两个标准的 Junit 测试，都是对待测方法“integerRepresentation”的调用，能够对源码进行部分覆盖。尽管用例中的测试数据较为随机，但具有非常清晰的测试结构，可以为用户提供一定的参考。

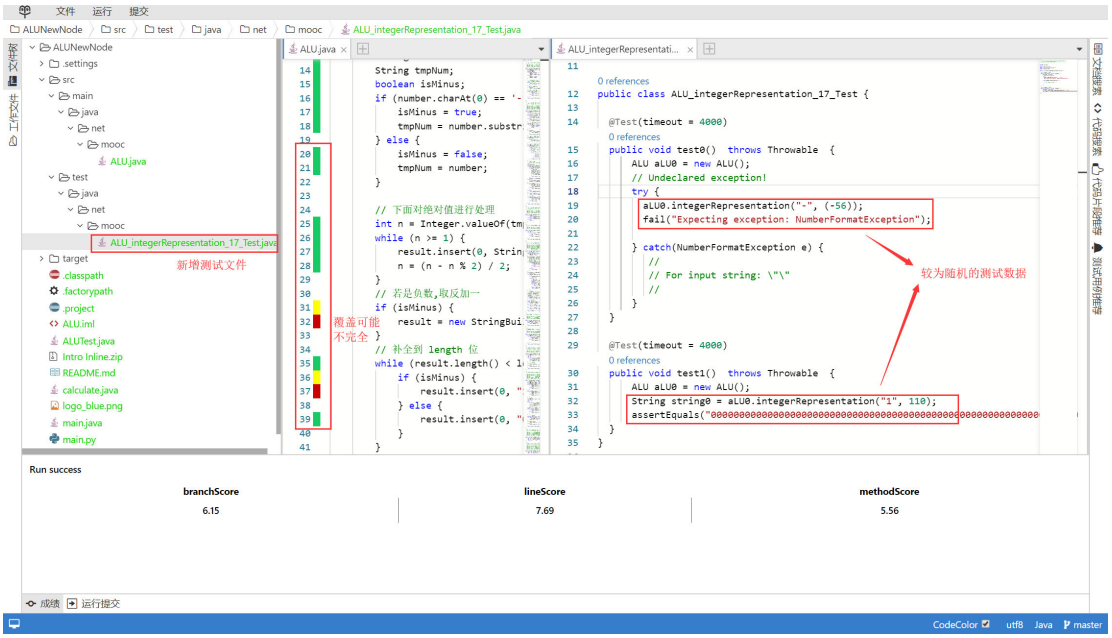


图 4.14: 基础测试用例生成效果

图 4.15展示了一个语料库生成测试用例的示例，生成了对于“integerRepresentation”方法的两个测试用例，与图 4.14相比，这些测试用例拥有更强的覆盖能力，使用的数据也具有更好的可读性，更加契合待测方法的测试需求。

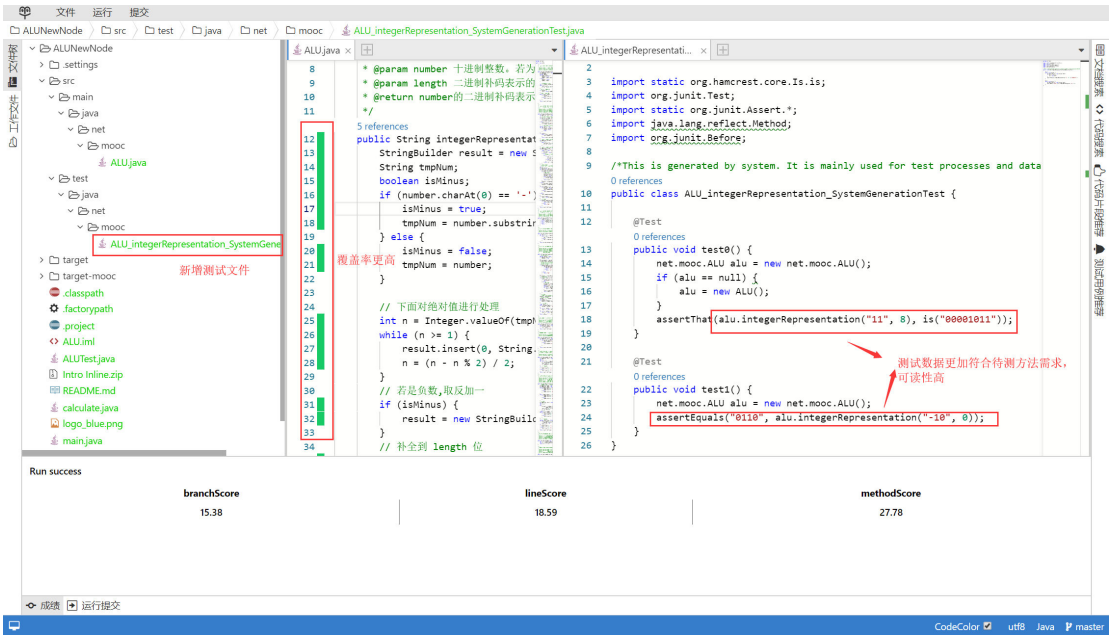


图 4.15: 语料库测试用例生成效果

4.5　本章小结

本章主要对测试开发系统的实现细节进行阐述。包括智能提示模块、覆盖可视化模块、同源代码匹配模块和自动化生成模块四个核心模块的顺序图、关键代码和页面展示，从组件调用流程、细节代码实现以及最终功能效果三个方面展开说明。

第五章　系统测试与实验分析

本章主要对系统的功能与性能测试和实验设计分析展开描述。系统测试主要介绍测试目标、测试环境、功能测试与性能测试的设计和执行，确认系统实现了预期功能，没有明显和错误的缺陷。实验分析包括应用实验和对比实验的设计、执行与结果分析，用来确认系统的价值和意义。

5.1　系统测试

5.1.1　测试目标

本系统测试主要针对系统功能的正确性和系统的可用性两方面展开：

第一，通过功能测试确定系统是否满足业务需求，是否正确实现了目标功能，即能够为用户提供智能提示、代码着色、测试用例生成等能力。

第二，通过性能测试确定系统在高并发高负载场景下的可用性，单台服务器服务能力有限时，能否通过水平扩展提升整体服务能力，以应对多人同时访问的场景。

5.1.2　测试环境

表 5.1: 硬件环境

服务	服务器	CPU	内存
智能提示服务	阿里云，规格 ecs.mn4.xlarge，Ubuntu 16.04	4 核	16G
测试生成服务	阿里云，规格 ecs.mn4.xlarge，Ubuntu 16.04	4 核	16G
相似分析服务	阿里云，规格 ecs.g5.xlarge，Ubuntu 16.04	4 核	16G
数据库服务	阿里云，规格 ecs.mn4.xlarge，Ubuntu 16.04	4 核	16G
负载均衡服务	阿里云，规格 ecs.mn4.xlarge，Ubuntu 16.04	4 核	16G

表 5.1展示了部署各个服务的的服务器资源、操作系统版本、CPU 及内存配置。系统涉及的服务有智能提示服务、测试生成服务、相似分析服务、数据库服务和负载均衡服务。服务器主要选择阿里云实例，操作系统为 Linux，均为 4C16G。

表 5.2展示了各个服务运行的软件环境和版本。智能提示服务和测试生成服务的运行依赖于 JDK8；相似分析服务采用 flask 实现和部署，依赖于 Python3 环

境；数据库服务部署 ElasticSearch 用于存储和检索，Kibana 提供了 ES 中数据的可视化展示。负载均衡服务使用 Nginx 作为反向代理，分发算法为 sticky，确保同一用户的请求始终被分发到同一台服务器。所有的服务均被封装为 Docker 容器运行，以便环境隔离和部署迁移。

表 5.2: 软件环境

服务	软件
智能提示服务	JDK 1.8.0_151，Maven 3.5.2，Docker 17.09.1-ce
测试生成服务	JDK 1.8.0_151，Maven 3.5.2，Docker 17.09.1-ce
相似分析服务	Python 3.8.2，Docker 19.03.5
数据库服务	ElasticSearch 7.5.1，Kibana 7.5.1, Docker 17.09.1-ce
负载均衡服务	Nginx 1.14.0

5.1.3　功能测试

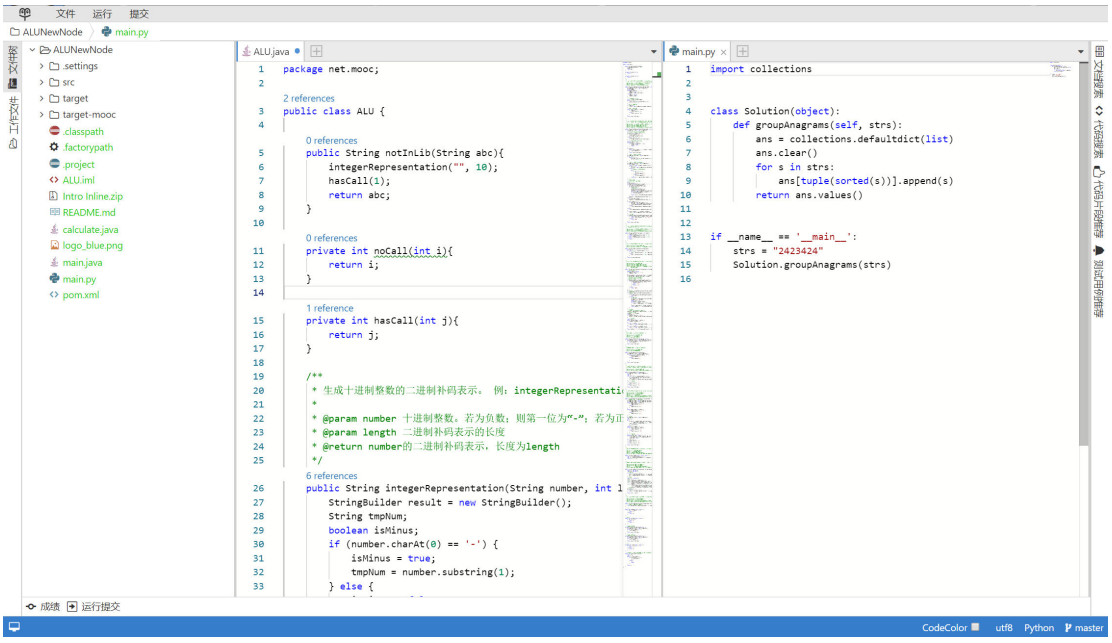


图 5.1: 测试项目结构和源码

本小节对系统需求分析部分得出的功能列表进行测试用例设计与描述，主要是面向用户角度的系统功能测试，尽量对所实现的功能做到完全覆盖，目标是通过对所设计的测试用例的执行结果来确认系统功能的正确性和完整性。用于测试的项目结构和源码如图 5.1所示，测试用例设计包括测试编号、测试功能、测试步骤和预期结果的描述，设计描述中将直接使用图中出现的文件名称和代

码内容，每个测试用例执行完毕后都将测试项目恢复至初始状态。最后通过统计各个测试用例执行的通过情况确认和总结系统功能的实现情况。

表 5.3是触发智能提示的测试用例设计。主要测试的是针对用户的操作行为，系统能否给出正确的语法提示。包括自动识别当前代码的编程语言，根据特定语言进行代码诊断、代码补全、悬浮提示、引用跳转等反馈。测试关注点一是对于用户操作是否有反馈，二是反馈的内容是否正确，三是反馈是否足够迅速。

表 5.3: 触发智能提示测试用例

测试 ID	TC1
测试名称	触发智能提示
测试功能	学生用户通过打开、编辑代码文件触发代码智能提示，包括语法诊断、代码补全、悬浮提示、引用查找等
测试步骤	1. 用户打开 ALU.java 文件 2. 用户删除第 8 行“return abc;”的分号, 看是否有错误警告；如果有错误警告，补上分号，看错误警告是否消失 3. 用户在“return abc;”前插入新行，输入“a”，观察页面是否产生提示，如果产生提示，能否进行选择并直接填充至当前位置 4. 用户在新行出继续输入至“abc.”，观察页面是否产生提示，如果产生提示，能否进行选择并直接填充至当前位置 5. 用户将鼠标悬停在第 8 行“abc”处，看是否出现变量解释 6. 用户将鼠标移动至第 6 行，并悬停在“integerRepresentation”位置，看是否出现方法的 Javadoc 说明框 7. 用户鼠标仍保持悬停在第 6 行“integerRepresentation”处，按住 Ctr 键并点击 8. 用户鼠标悬停在第 8 行“abc”处，按住 Ctr 键并点击 9. 用户打开 main.py，对 python 代码执行类似步骤 1-7 的操作，观察页面反应
预期结果	1. 针对步骤 2 中行为，“abc”下应该出现红色波浪线作为错误警示，将鼠标移至红色波浪线位置，出现错误信息悬浮框，提示“Syntax error”并建议插入分号；补上分号后，红色波浪线消失 2. 针对步骤 3 中行为,“a”后应当出现变量和方法提示框，提示内容包括变量“abc”和以“a”开头的可用方法。用户可以对提示项进行选择，回车后该提示内容被填充到当前光标所在位置 3. 针对步骤 4 中行为，“abc.”后应当出现 String 对象可用方法和变量的提示框。用户可以对提示项进行选择，回车后该提示内容被填充到当前光标所在位置 4. 针对步骤 5 中行为，“abc”附近应当出现包含变量解释的悬浮框 5. 针对步骤 6 中行为，“integerRepresentation”附近应当出现方法的 Javadoc 说明框 6. 针对步骤 7 中行为，光标应跳转至第 26 行“integerRepresentation”方法定义处 7. 针对步骤 8 中行为，光标应当跳转至第 5 行“abc”变量定义处 8. 针对步骤 9 中行为，页面反应与上述结果较为相似，在语法判定和提示内容方面稍有不同，符合正常的 Python 语法定义 9. 用户的所有操作行为，系统应当在 0.5s 内给出反馈

表 5.4是代码着色的测试用例设计，主要测试的是在开发者测试场景下，系统能否根据测试代码的变动相应地改变对源代码的着色渲染，并确认着色渲染的结果与测试的实际覆盖情况一致。同时测试学生用户开启关闭代码着色功能的能力。

表 5.4: 触发代码着色测试用例

测试 ID	TC2
测试名称	触发代码着色
测试功能	学生用户能够触发代码着色功能
测试步骤	1. 确认着色功能已开启 2. 用户点击运行按钮 3. 运行完毕后，用户打开 ALU.java 文件，查看页面是否进行着色渲染 4. 用户在 ALUTest.java 中新增对”integerRepresentation” 的测试代码（原先无测试代码），再次点击运行按钮，运行完毕后，查看页面着色渲染是否发生变化 5. 用户将右下角 CodeColor 改为不勾选状态，观察 ALU.java 的变化
预期结果	1. 针对步骤 3，页面上打开的 ALU.java 中有效代码行应全部被着为红色（此时无测试用例），无效代码行不被着色 2. 针对步骤 4，页面上打开的 ALU.java 中着色情况应当发生改变，被测试执行到的行应当被着为绿色或黄色，仍未被执行的行仍着为红色，无效代码行不被着色 3. 针对步骤 5，页面上打开的 ALU.java 的着色应当消失。

表 5.5: 触发测试代码生成用例描述

测试 ID	TC3
测试名称	触发测试代码生成
测试功能	学生用户能够触发测试代码生成功能，包括从工具生成基本测试用例和从语料库生成测试用例
测试步骤	1. 用户进入系统 2. 将鼠标在任意行行号处悬停 2s 及以上 3. 将鼠标在第 10 行处悬停 2s 及以上，点击生成方法级测试用例 4. 在第 7 行处悬停 2s 及以上，点击生成方法级测试用例，生成结束后点击运行，运行结束后观察 ALU.java 的着色情况 5. 删除所有测试，在第 27 行处悬停 2s 及以上，点击生成方法级测试用例, 生成结束后点击运行，运行结束后观察 ALU.java 的着色情况 6. 将项目包名由 net.mooc 改为 net.test，将“integerRepresetation”方法名重构为“integerRepresentating”，在 27 行处悬停 2s 及以上，点击生成方法级测试用例 7. 将鼠标在第 12 行处悬停 2s 及以上，点击生成方法级测试用例 8. 将鼠标在第 16 行处悬停 2s 及以上，点击生成方法级测试用例 9. 将鼠标在第 7 行处悬停 2s 及以上，点击生成方法级测试用例，重复此操作一次

预期结果	1. 针对步骤 1，系统应提示“在任意行号处停留 2s 可进行测试生成” 2. 针对步骤 2，所选行号处应悬浮出“生成方法级测试用例按钮” 3. 针对步骤 3，第 10 行不属于任何方法，系统应提示其无效行，不进行测试生成 4. 针对步骤 4，系统应从工具生成基本测试用例，20s 内返回结果；运行结束后，ALU.java 对应的“notInLib”方法体行着为绿色。 5. 针对步骤 5，系统应在 5s 内从语料库对“integerRepresentation”方法生成测试代码，运行后 ALU.java 对应的“integerRepresentation”方法体行着色变为绿色。 6. 针对步骤 6，生成结束后，新生成的测试代码中涉及 ALU 初始化和被测方法调用处应与 net.test.ALU 和“integerRepresentating”相符。 7. 针对步骤 7，生成结束后，新生成的用例应为对方法“notInLib”的测试，因为 12 行对应的 hasCall 为私有方法，所以对调用了 hasCall 的公有方法“notInLib”进行测试生成 8. 针对步骤 8，系统应给出“私有方法未被调用，可使用反射进行测试”的提示 9. 针对步骤 9，首次操作时，系统正常进行生成流程，再次操作时，由于第一次生成尚未结束，系统应提示“该方法已有测试生成进行中”

表 5.5是触发测试代码生成的测试用例设计。主要测试的是系统的基础测试用例生成和语料库测试用例生成能力，包括页面交互、错误提示和功能使用等方面。功能使用上主要关注生成的测试代码的有用性，即能否尽可能提高待测方法的覆盖率，同时关注系统的响应速度，语料库生成应控制在 5s 内，工具生成应控制在 20s 内。

表 5.6是查看原始用例的测试设计，主要是确认从语料库生成测试代码后，用户可以查看原始测试用例代码，为用户提供更多的选择以及学习的可能性。

表 5.6: 查看原始测试用例

测试 ID	TC4
测试名称	查看原始测试用例
测试功能	学生触发测试生成后，如果是从语料库生成的测试用例，允许查看用来生成测试代码的原始用例
测试步骤	1. 用户在第 27 行处悬停 2s 及以上，点击“生成方法级测试用例” 2. 用户查看“initial-generate-tests/net/mooc”目录下的原始测试用例文件
预期结果	1. 针对步骤 1，系统从语料库生成测试代码 2. 针对步骤 2，系统在“initial-generate-tests/net/mooc”下生成测试代码的原始测试用例文件 ALU_integerRepresentating_SGTest.java

表 5.7是清除缓存的测试用例设计。主要用来测试程序结果缓存的作用。由于本系统主要针对测试开发，因此存在这样一个推论：当前编程人员对源代码的改动是很少的。因此系统在初次进行测试生成时缓存了程序分析的结果，以便后续的生成流程更加快速。如果对源代码进行了改动，可以通过删除 target-mooc

对应源代码的分析缓存来进行更新，也可以删除整个“target-mooc”文件夹，这样整个项目的源代码都将重新进行分析。

表 5.7: 清除缓存测试用例

测试 ID	TC5
测试名称	清除程序分析缓存
测试功能	学生用户清除程序分析缓存，以更新对源代码的分析结果
测试步骤	1. 用户打开 ALU.java 2. 在第 6 行前插入两个空行 3. 鼠标在第 10 行行号处悬停 2s 及以上，点击“生成方法级测试用例”，观察系统响应 4. 删除文件目录下的“target-mooc”文件夹下，“junitGenerate”和“callGraph”中的“net.mooc.ALU.json”文件 5. 鼠标在第 10 行行号处悬停 2s 及以上，点击“生成方法级测试用例”，观察系统响应
预期结果	1. 针对步骤 3，由于系统仍使用之前的源代码分析结果，之前的第 10 行为无效行，不属于任何方法，因此系统提示不进行测试生成 2. 针对步骤 4、5，系统重新分析修改后的源代码，正常进行生成流程，文件目录下出现新的“target-mooc”文件夹

图 5.8是配置功能开关测试用例设计。主要测试教师用户关闭和开启测试生成能力或代码着色能力后，系统是否同步该特征配置。此功能主要用于教学、考试和练习等场景下。

表 5.8: 配置功能开关

测试 ID	TC6
测试名称	配置功能开关
测试功能	教师用户能够控制测试生成能力和代码着色能力的开启与关闭
测试步骤	1. 教师登录慕测平台，进入考试编辑页面 2. 教师开启着色功能后，学生进入 WebIDE，点击“运行” 3. 教师关闭着色功能后，学生进入 WebIDE，点击“运行” 4. 教师开启测试生成功能后，学生进入 WebIDE，在源码任意行号处悬停 2s 及以上 5. 教师关闭测试生成功能后，学生进入 WebIDE，在源码任意行号处悬停 2s 及以上
预期结果	1. 针对步骤 2，运行结束后，无着色渲染 2. 针对步骤 3，源代码出现着色渲染 3. 针对步骤 4，系统无反馈 4. 针对步骤 5，系统悬浮“生成方法级测试用例”按钮

表 5.9是系统功能测试用例的执行结果。测试人员严格按照测试步骤执行测试，记录测试结果，并与预期结果进行比对。如表中所示，所有测试用例均已通过。说明系统正确实现了业务需求，符合产品功能设计初衷。

表 5.9: 测试用例执行结果

测试用例 ID	对应用例描述	测试结果
TC1	UC1	通过
TC2	UC2	通过
TC3	UC3	通过
TC4	UC4	通过
TC5	UC5	通过
TC5	UC5	通过

5.1.4　性能测试

本节从性能方面对系统进行测试，借助 Apache-Jmeter 以及 Linux 内存管理工具对系统主要接口：智能提示、代码着色和测试生成进行压力测试。Apache-Jmeter 允许同时发送大量请求，以模拟真实情况下的高并发场景。目前可预测的场景是 2-4h 内的考试，约有 200 人同时在线。因此本测试主要确认在合理的资源使用范围内，系统能否满足实际使用的需要。受于资源限制，本小节将对单台服务器的服务能力进行测试，并以此作为实际应用部署时服务器扩展的参考。

由于代码着色是项目运行完成后生成着色信息并推送到客户端，因此着色接口测试实质上是对运行接口的测试，本项测试中主要关注替换覆盖统计内核后，项目的等待时间、运行时间和总耗时，并将其与使用未替换覆盖内核时的耗时相比较。对比数据如表 5.10所示，其中老的测试数据来自一台 8C16G 服务器，新的数据为当前的 4C16G 测试服务器。由表中数据可以看出，即使服务器存在 4 核的差距，替换新的内核后，等待时间、运行时间以及总时间仍大幅下降，主要是由于替换了覆盖内核后，单个项目的运行时间缩减到 20s 内，而原先单个项目的运行时间高达 65s。因此，本系统大幅提升了原先运行接口的效率，满足性能要求。

表 5.10: 运行接口测试结果对比

用例数	服务类型	平均等待时间	平均运行时间	平均总用时	最差耗时	总用时
32	old	102.31s	65.25s	167.57s	269.24s	270.36s
	new	68.625s	19.554s	88.281s	157.681s	158.473s
64	old	230.64s	62.60s	293.24s	513.466s	514.11s
	new	139.85s	18.56s	158.49s	298.865s	230.11s
128	old	478.44	62.75s	541.19s	1017.854s	1019.277s
	new	287.507s	18.62s	306.207s	600.949s	601.547s

智能提示接口的测试将手工测试与压力测试结合。在手工测试阶段，不断地增加智能提示客户端观察服务器 CPU 和内存使用情况。在过程中观察到，最开始建立连接的过程，CPU 使用率最高，4 核均达到 85% 以上。连接建立完成后，尽管后续仍在通信，但 CPU 占用率很低，回落至 3% 左右。因此我们首先关注内存占用情况。

图 5.2是系统内存使用量随智能提示客户端数量增长的变化图像。可以看到，内存的上升与客户端数量几乎呈线性增长。每增加一个 Java 客户端，内存增长约 0.6G，而每增加一个 Python 客户端，内存增长约 0.08G。在实际的部署中，Java 智能提示服务与 Python 智能提示服务相互独立，以获得服务器的最高使用性能。对于当前这台空余内存约 8G 的服务器，可支持的 Java 客户端数量约为 15 台，可支持的 Python 客户端约为 100 台。

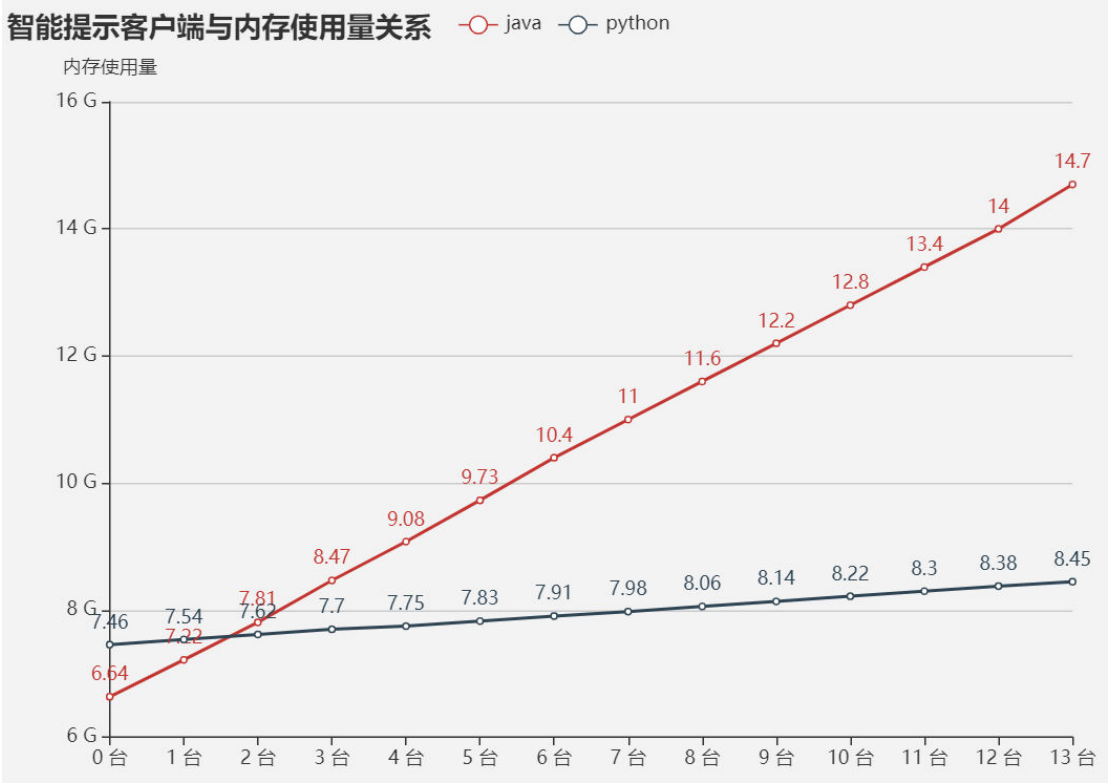


图 5.2: 智能提示客户端与内存使用量关系图

下一步我们使用 Jmeter 的 WebSocket 采样器进行连接建立过程的压测，确认 CPU 的处理能力。一个请求进程包括建立 WebSocket 通信、初始化 LSPServer、发送 4-5 条项目初始化请求和 2 条提示请求。最终获得的压力测试结果如表 5.11和表 5.12所示，由表中数据可知，平均响应时间最高为 1.5s，几乎所有请求都能被正确处理。因此对于 4C8G 服务器配置，能够同时支持 15 台左右的 Java 提示客

户端和 100 台左右的 Python 提示客户端。且压测要求请求同时发送，属于较为极端的情形，实际使用中能够支持的并发量相对更高。

表 5.11: Java 智能提示压测结果

请求接口	并发量	平均值	最小值	最大值	异常率
建立 websocket 连接	8	63ms	55ms	87ms	0.00%
	12	62ms	53ms	91ms	0.00%
	15	60ms	53ms	77ms	0.00%
初始化请求（只写）	32	0ms	0ms	1ms	0.00%
	48	0ms	0ms	2ms	0.00%
	60	0ms	0ms	2ms	0.00%
提示请求	16	608ms	0ms	2036ms	0.00%
	24	864ms	0ms	4160ms	0.00%
	30	1523ms	0ms	5495ms	0.00%

表 5.12: Python 智能提示压测结果

请求接口	并发量	平均值	最小值	最大值	异常率
建立 websocket 连接	30	65ms	47ms	89ms	0.00%
	50	63ms	52ms	81ms	0.00%
	100	129ms	52ms	2063ms	0.00%
初始化请求（只写）	150	0ms	0ms	1ms	0.00%
	250	0ms	0ms	26ms	0.00%
	500	0ms	0ms	2ms	0.00%
提示请求	60	0ms	1ms	1ms	0.00%
	100	0ms	0ms	2ms	0.00%
	200	60ms	0ms	6001ms	1.00%

语料库测试生成接口通过 Jmeter 的 HTTP Sampler 进行压测，结果如表 5.13所示，分别选取样本量 10、20、25、30、40、50、100、150、200 同时请求进行测试，可以看到当并发量较小时，生成时间能够控制在 10s 内；当并发量小于 150 时，生成时间在 20s 内；当并发量达到 200 时，生成时间超过 30s，并且开始出现极少数失败情况。尽管 10-30s 的时间并不短暂，但 Evosuite 等工具对于简单项目的测试生成也要 5min 左右，因此系统在响应速度上仍然具备显著优势。

但在实际的场景中，几乎不会出现所有用户都在同一秒请求测试生成接口的情况。因此我们调整压测线程组的 Ramp-up 为 10s、30s、60s 和 120s 重新进行测试，即所有请求在 Ramp-up 设置的时间内陆续发送完毕，而不是在同一时刻全部发出，这样更加贴近于真实场景。测试结果如表 5.14所示，可以看到，伴

随着 Ramp-up 时间的增加，平均处理时间明显减少，说明系统可以满足实际使用需求。伴随着用户的增加，还可以通过部署多台服务器进一步减轻系统压力。

表 5.13: 测试生成接口压测结果

并发量	平均值	最小值	最大值	异常率	吞吐量
10	388ms	284ms	609ms	0.00%	7.4/sec
20	1461ms	442ms	1952ms	0.00%	8.1/sec
25	1859ms	305ms	2592ms	0.00%	7.6/sec
30	3018ms	1737ms	3418ms	0.00%	7.8/sec
40	5496ms	3784ms	5948ms	0.00%	6.2/sec
50	7868ms	6894ms	8400ms	0.00%	5.6/sec
100	14549ms	7360ms	16667ms	0.00%	5.9/sec
150	21231ms	7889ms	27704ms	0.00%	5.3/sec
200	32143ms	18032ms	38336ms	1.50%	5.1/sec

表 5.14: 测试生成接口压测结果（设置 Ramp-up）

并发量	Ramp up	平均值	最小值	最大值	异常率	吞吐量
100	10s	3494ms	253ms	9567ms	0.00%	5.5/sec
	30s	1338ms	245ms	5734ms	0.00%	3.3/sec
	60s	403ms	237ms	2760ms	0.00%	1.7/sec
	120s	356ms	254ms	1905ms	0.00%	50.4/min
150	30s	1594ms	233ms	4480ms	0.00%	5.0/sec
	60s	641ms	258ms	4829ms	0.00%	2.5/sec
	120s	721ms	241ms	8640ms	0.00%	1.3/sec
200	60s	1558ms	237ms	11495ms	0.00%	3.3/sec
	120s	949ms	231ms	9579ms	0.00%	1.6/sec

综上，通过对代码着色接口、智能提示接口和测试生成接口的压力测试数据，能够验证系统在性能和可用性等方面具有较为良好的表现，满足实际应用场景的需要。

5.2　实验设计

本小节通过实验设计来评估测试生成功能的有效性，主要验证：

第一，本系统提供的测试生成能力有助于测试人员编写测试用例，提高源代码覆盖率的同时提升测试水平。即验证系统的实际应用意义和价值。

第二，本系统采用的相似匹配算法具有一定的创新性，相比于前人的工作，本系统具有自身的独特性和优势。

5.2.1　应用实验设计

本系统的应用场景之一是教学场景：在慕测生态中，由教师布置开发者测试任务，学生通过 WebIDE 编写测试用例完成任务，最终提高测试技能的过程。

实验步骤如下：

1. 收集最近三年软件测试大赛、软件测试课程和培训项目中的 25 个题目包，同时选取得分最高的前十名同学的测试代码作为原料，处理后存储到语料库中

2. 随机选择一个题目包的源码

3. 更改其原有包名、类名

4. 更改部分方法名，包括将原方法名更改为近义词，或将其改为拼写错误的形式

5. 对源代码添加噪音，包括注释、空行等；对方法体实现做部分修改，如更改局部变量名，调整代码组织结构

6. 使用该题目包重新在慕测平台建立试题

7. 在 WebIDE 中打开该试题，逐个方法调用测试代码生成功能并运行项目，观察所生成的测试用例的个数以及覆盖得分情况

表 5.15: 应用实验结果

编号	待测方法	语料库方法	用例数量	分支覆盖	行覆盖
1	integerRepresentating	integerRepresentation	2	15.38	18.59
2	floatRepresentation	floaRepresentation	8	50.77	53.85
3	ieee755	ieee754	2	53.08	55.13
4	integerTrueValue	无	1	56.15	57.69
5	floatTrueValue	floatTrueValue	7	85.38	92.95
6	oneAdder	oneAdd	1	86.51	93.59
7	sum	add	4	92.31	96.79
8	normal	nomalize	1	92.31	96.79

```
@Test
public void test3() {
    net.mooc.ALU alu = new net.mooc.ALU();
    assertEquals("+Inf", alu.floatTrueValue("011111111000", 8, 3));
}
```

图 5.3: 测试用例示例

实验结果如表 5.15所示。由编号 1、2、3、6 的待测方法和对应的语料库方法可以看出，系统能够识别出方法名拼写错误并进行对应；由编号 7、8 的待测方法与对应的语料库方法可以看出，系统能够识别出语义相似的方法名并进行对应；用例的数量在个位数级别浮动，且大部分用例代码如图 5.3所示，由数据准备和方法调用两部分组成，精简且具有很强的针对性，易于阅读和理解。通过逐个对待测方法进行测试生成，可以看到项目的分支覆盖和行覆盖呈上升趋势，当所有方法均被测试，覆盖得分也趋于稳定。因而可以初步确认，本系统的测试生成功能有助于提高源码覆盖率，生成的测试精简易读。

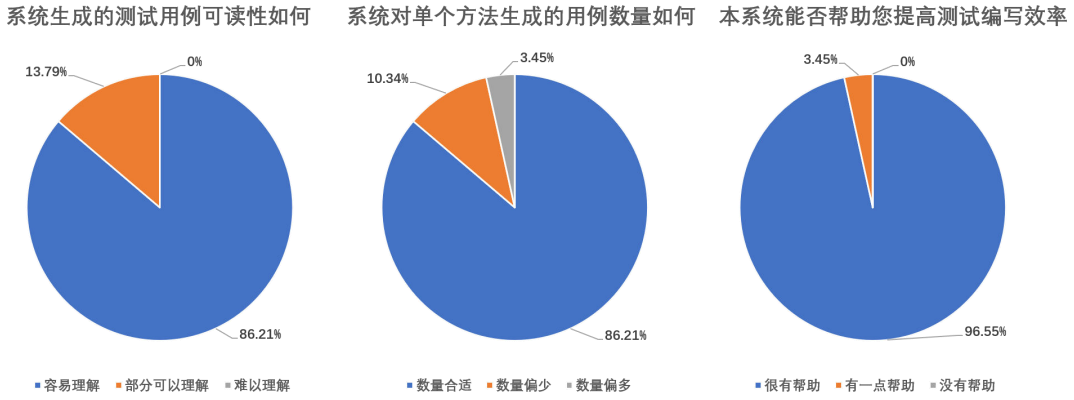


图 5.4: 用户反馈统计

此外我们邀请 29 位软件工程学生利用本系统对待测项目进行测试编写并填写反馈。这些学生具有初步的 Junit 测试使用经验，能够判定用例的可理解性和有用性。其中半数同学曾使用过其他自动化测试工具，可以在曾经的使用经验基础上对本工具效果进行评估。反馈结果统计如图 5.4所示。86.21% 的用户认为本系统生成的用例数量适中并具有较高的可读性，即这些用例达到了覆盖目标且便于理解；96.55% 的用户表示本系统在提高测试编写效率方面很有帮助，表明系统获得了较高的用户支持度和满意度。实验还发现使用本系统工具后大部分用户能够在 10-30 分钟完成测试编写并达到项目覆盖要求，而不使用工具则需要 1-2 小时来达到类似效果。综上，系统确实能够生成易于理解的测试用例，为用户测试编写带来较大帮助，具有较高应用价值。

5.2.2　对比实验设计

本小节将系统的同源匹配算法与 SENTRE 测试搜索引擎 [10] 采用的签名匹配算法进行对比实验，比较两种方式在测试代码匹配方面的成功率和准确性。成功率即能否针对待测方法匹配到相应相似方法，进而匹配对应测试用例，准确性即所匹配到的测试用例是否能够正确地测试目标方法。

表 5.16展示了本系统实现的待测方法同源匹配算法与 SENTRE 的签名匹配算法的实现思路比较。可以发现，本系统的匹配算法综合考虑方法名与方法体的相似度，而 SENTRE 的签名匹配算法着重从方法签名判定相似。为能够比较两种算法的最终效果，本实验对 SENTRE 的签名匹配进行了实现。

表 5.16: 两种相似匹配算法的实现思路

同源匹配算法（本系统）	签名匹配算法（SENTRE）
1. 在语料库中对待测方法的方法名和方法输入输出结构进行完全匹配：如果匹配结果成功，返回匹配到的结果，最多 10 个，转到步骤 5 2. 在匹配方法结构的基础上，利用 Levenstain 距离匹配方法名：如果匹配结果成功，返回匹配到的结果，最多 20 个，转到步骤 5 3. 在匹配方法结构的基础上，将语料库中的方法名与待测方法名进行语义相似分析，取相似度达到阈值且最高的 10 个结果; 4. 在匹配方法结构的基础上，对方法名添加通配符，最多获取 10 个方法，与步骤 3 中的结果一起返回; 5. 将待测方法的实现代码与所获得的相似方法的实现代码逐一进行代码相似匹配，代码相似度最高的方法将被认为与待测方法同源，即判定该方法的测试用例可被用于测试待测方法。	1. 搜索待测方法类名、方法名和输入输出结构的完全匹配; 如果匹配成功，返回匹配到的结果，转到步骤 5 2. 在匹配类名和方法结构的基础上，对方法名添加通配符进行搜索：如果搜索到结果，转到步骤 5 3. 搜索类名和方法结构的完全匹配，不再考虑方法名：如果搜索到结果，转到步骤 5 4. 在方法结构匹配的基础上，不考虑方法名，对类名添加通配符进行搜索，返回搜索结果 5. 如果在 1-4 的搜索中获得了相似方法，则将该方法的测试用例用于待测方法，否则认为语料库中没有相似方法

如表 5.17所示，我们从 GitHub 上选取了 8 个开源项目，这些项目包含了 LeetCode [1]、LintCode [2]、InterviewBit [3]和 GeeksForGeeks [4] 等编程网站上近千道编程题的实现，目的相同的编程题具有天生的相似性，但来自不同的平台以及由不同的人实现又导致不同项目在所包含的方法与实现上有所区别。前 5 个项目包含题目实现和测试用例，将作为原料构建语料库数据，后 3 个项目仅包含实现，将作为待测目标。

分别采用本系统的同源匹配算法和 SENTRE 的签名匹配算法对项目 6-8 中的所有公开方法进行测试自动化生成，实验结果如表 5.18所示。项目编号对应表 5.17中的编号，匹配方法数是成功匹配到了相似方法的项目方法总数；未匹配方法数则是匹配失败的项目方法总数；失败和成功的测试用例数是在测试生

[1]https://leetcode-cn.com/

[2]https://www.lintcode.com/

[3]https://www.interviewbit.com/

[4]https://www.geeksforgeeks.org/

成后，经过简单人工调整，运行测试用例所得到的执行结果统计；测试准确率是执行成功的测试用例数与生成的测试用例总数的百分比；类覆盖、方法覆盖和行覆盖是测试对项目源码覆盖情况的收集结果。

表 5.17: 实验项目列表

项目编号	项目名称	包含方法数量	包含测试数量	项目用途
1	interviewcoder/leetcode	608	2083	构建语料库
2	rekinyz/LeetCode	152	560	构建语料库
3	interviewcoder/interviewbit	131	339	构建语料库
4	interviewcoder/lintcode	47	30	构建语料库
5	interviewcoder/geeksforgeeks	48	50	构建语料库
6	gouthampradhan/leetcode	1025	0	待测项目
7	nagajyothi/InterviewBit	633	0	待测项目
8	Blankj/awesome-java-leetcode	136	0	待测项目

表 5.18: 测试自动化生成实验结果

项目编号	算法	匹配方法数	未匹配方法数	失败用例数	成功用例数	测试准确率	类覆盖	方法覆盖	行覆盖
6	同源	116	909	8	161	95.27%	27%	19%	19%
	签名	88	937	14	124	89.86%	18%	12%	13%
7	同源	81	552	12	117	90.70%	32%	17%	20%
	签名	47	586	15	76	83.52%	13%	7%	8%
8	同源	47	89	3	68	95.77%	70%	37%	47%
	签名	51	85	9	65	87.84%	70%	35%	49%

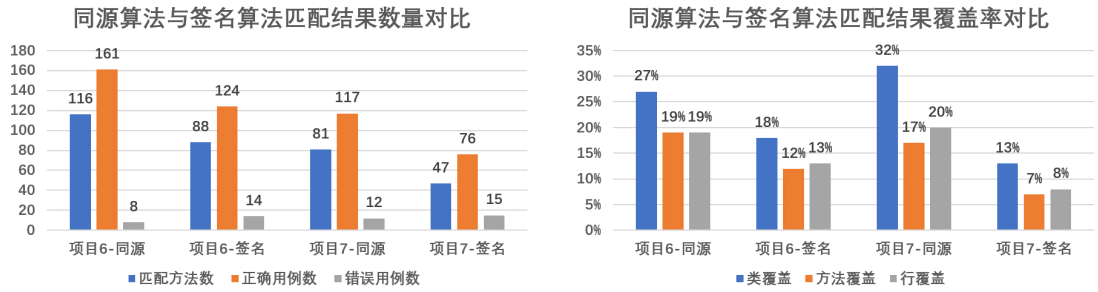


图 5.5: 对比实验结果

对于项目 6 和项目 7，图 5.5可直观看出，针对相同待测对象，同源算法的匹配方法数相对签名算法明显增多，生成的正确测试用例更多，而错误用例更少；且同源算法产生的用例在类覆盖、方法覆盖和行覆盖方面均有显著更优表现。

对于项目 8，同源算法的匹配方法数少于签名算法。但是经过对测试数据的分析发现，用来构建语料库的原料项目 1 中所有的类命名都采用了 Solution，而待测项目 8 中所有的类命名也都采用了 Solution，导致对于签名算法而言类名匹配失去区分度，使得签名算法尽管匹配了更多的方法，但也导致了更多的错误用例，在覆盖率方面也仅仅与同源算法持平，但错误的测试用例却会导致开发人员花费更多的精力去甄别。

表 5.18中的测试准确率一栏可以用来度量测试用例是否能够正确地测试目标方法，即生成测试用例的准确性。可以看到，签名算法的结果在 80%-90% 之间，而同源算法在产生得测试用例数量更多的前提下，却能够保持 90%-95% 的准确率。签名匹配算法的缺陷在于，当语料库方法的类名与待测方法存在相似性且方法输入输出结构满足条件时，即判定该方法与待测方法相似，进行测试用例复用，这可能导致测试用例的误推。例如用户希望测试 ValidParentheses.hasBalancedBrackets 方法，但签名匹配算法却为其推荐了 LongestValidParentheses.longestValidParentheses 方法的测试用例，尽管两者具有相同的方法输入输出结构，但所实现的功能却完全不同。而同源匹配算法能够综合考虑方法签名与方法体代码的相似性，从而推荐真正相似的方法 Solution.hasBalanceBrancket 的测试用例。

综上实验表明，相对于签名匹配算法，正常情况下，本系统的同源匹配算法能够匹配更多待测方法，生成较多测试用例的同时保持 90% 以上的测试准确率，对源代码的类覆盖、方法覆盖和行覆盖也具有符合期望的表现。

5.3　本章小结

本章对基于同源代码匹配的测试开发系统进行了系统测试和实验设计。通过功能测试确认了系统实现的正确性，通过对主要接口的性能测试确认了系统的性能和可用性，结合多种压测手段，并与以往测试数据比较，最终确认系统优势。在实验设计上，应用实验验证了系统的有用性，而对比实验证明了在相似方法匹配方面，系统采用的同源算法具有一定的优越性。

第六章　总结与展望

6.1　总结

人们对软件质量要求的不断提高，使得软件测试成为软件生产活动中愈发重要的一环。为了更加全面地测试软件系统，同时为测试开发人员提供更多的参考，本文设计并实现了基于同源代码匹配的在线测试开发系统。

本系统利用线上 IDE 环境，为用户提供测试自动化生成的入口，免去安装和配置的成本。系统实现了智能代码提示，遵循本地 IDE 的使用习惯进行设计，用户能够进行无缝切换。在测试可视化方面，系统将用例对源代码的覆盖情况通过行着色直观地展示给用户，有助于开发人员对用例有用性进行判断。在测试生成方面，系统对历史项目和测试数据进行收集后建立语料库，通过语义相似分析、拼写校正和代码相似匹配等手段从库中检索相似方法，对该方法的多个测试用例进行筛选、组合与改造后展示给用户，最大程度提高测试可用性。对于那些语料库中不包含的项目方法，系统通过改造后的 Evosuite 工具进行基本测试用例的生成。相对于已有的测试自动化生成工具，本系统的优势在于：

第一，已有的测试自动化生成工具大多作为本地 IDE 插件，或者独立工具，需要用户安装和配置，可能面临环境和版本兼容等问题，本系统作为在线平台，访问即可用，免去环境准备的成本。

第二，大部分工具对覆盖率的统计维度是项目和类的层次，本系统能够精确到行级覆盖，给用户带来对测试用例能力更加直接和清晰的认识。

第三，许多工具虽然具有测试生成的能力，但多为针对整个项目，生成过程耗时久，生成的测试代码体量大，数据相对随机，阅读和理解成本较高。本系统通过历史语料库进行生成，响应速度更快；既可以面向整个项目，也可以面向单个方法，将测试用例维持在单个待测方法级别，指向明确；且测试数据是由前人编写而来，更加符合待测方法的需求，也具有更高的可理解性，有助于测试开发人员学习和技能提升。

第四，部分工具尽管能通过一些手段提供测试代码参考样例，但这些代码大部分不可运行，或者仅包含测试框架。本系统在前期语料库建立过程中尽量确保单个测试用例的可执行，生成过程中会对检索到的测试用例针对待测方法的上下文进行改造，包括对象声明和初始化类型调整，方法调用名称替换等，最大程度使其可直接运行。

本文的主要工作如下：

首先，对系统的项目背景和意义进行介绍，综述了目前国内外对于在线编程系统、测试自动化生成和程序相似度分析的研究现状，经过整理和分析，确认了系统与当前已有研究的不同点，并对可用技术进行了调研和实测。

其次，进行了系统的需求分析和总体设计。针对系统的使用目标、使用对象、使用方式和使用场景进行剖析。将系统划分为智能提示、覆盖可视化、同源代码匹配和测试自动化生成四个核心模块，并对各个模块的架构和实现思路进行总体设计。

最后，完成了系统的实现、测试评估和实验分析。通过顺序图、关键代码以及页面展示对智能提示、覆盖可视化、同源代码匹配和测试自动化生成四个模块的实现细节展开介绍，包括流程设计、缓存和异步机制等的抉择与实现。测试部分依照工业流程对系统进行功能测试，通过压测工具对系统主要接口的并发能力和响应速度进行性能测试。实验分析部分通过设计应用实验和对比实验论证了系统的实用价值与优越性。

6.2 工作展望

尽管基于同源代码匹配的在线测试开发系统已经开发完成，但系统仍有待后续工作进行完善。

第一，方法名的语义相似度量。目前本系统采用 googlenews 的词向量模型进行相似度计算，尽管这一模型已经涉及了较为广泛的词库，但仍然可能存在未被包含的词汇。未来系统将继续在机器学习和神经网络领域进行探究，寻找效果更佳的语义相似度量方法，针对语料库数据的变化动态调整相似阈值，以期达到更好的匹配效果。

第二，系统的语料库构建受限于自动化切片技术的实现效果，目前主要针对 Junit 测试框架具有较好的切分效果，虽然可以扩展到其他测试框架，但部分片段需要人工检查以保证正确性。未来系统将在程序分析领域继续研究，深入分析程序依赖关系，改进切片技术，以进一步扩增语料库，构建更加广泛的测试数据。

第三，教学场景策略优化。未来将考虑在练习、考试等场景下加入权值动态优化，以在分数计算时区分自动生成的用例与用户编写的用例，应当鼓励用户在参考生成的测试基础上更多地自行编写测试，从而提高自身测试能力。同时可以考虑加入监督学习策略来辅助教学方面的应用。

参考文献

[1] 董威, 单元测试及测试工具的研究与应用, 微型电脑应用 24 (5) (2008) 24–28.

[2] R. Pham, Y. Stoliar, K. Schneider, Automatically recommending test code examples to inexperienced developers (2015) 890–893.

[3] M. Goldman, G. Little, R. C. Miller, Real-time collaborative coding in a web IDE (2011) 155–164.

[4] C. L. Ignat, M. Norrie, G. Oster, Handling Conflicts through Multi-level Editing in Peer-to-peer Environments.

[5] D. Mirkovic, S. L. Johnsson, CODELAB: A developers' tool for efficient code generation and optimization (2003) 729–738.

[6] T. Dvornik, D. S. Janzen, J. Clements, O. Dekhtyar, Supporting introductory test-driven labs with webide (2011) 51–60.

[7] G. Fraser, A. Arcuri, Evosuite: automatic test suite generation for object-oriented software (2011) 416–419.

[8] O. Hummel, W. Janjic, C. Atkinson, Code conjurer: Pulling reusable software out of thin air, IEEE Software 25 (5) (2008) 45–52.

[9] R. Pham, Y. Stoliar, K. Schneider, Automatically recommending test code examples to inexperienced developers (2015) 890–893.

[10] W. Janjic, C. Atkinson, Utilizing software reuse experience for automated test recommendation, in: 8th International Workshop on Automation of Software Test, AST 2013, San Francisco, CA, USA, May 18-19, 2013, 2013, pp. 100–106.

[11] L. Yu, Y. Lei, R. Kacker, D. R. Kuhn, ACTS: A combinatorial test generation tool (2013) 370–375.

[12] K. Lakhotia, M. Harman, H. Gross, AUSTIN: an open source tool for search based software testing of C programs, Inf. Softw. Technol. 55 (1) (2013) 112–125.

[13] 薛猛, 姜淑娟, 王荣存, 基于智能优化算法的测试数据生成综述, 计算机工程与应用 54 (17) (2018) 16–23.

[14] 邓爱萍, 程序代码相似度度量算法研究, 计算机工程与设计 29 (17) (2008) 4635–4639.

[15] 熊浩, 晏海华, 郭涛, 黄永刚, 郝永乐, 李舟军, 代码相似性检测技术：研究综述, 计算机科学 37 (8) (2010) 9–14.

[16] 刘欣, 段云所, 陈钟, 程序可执行代码同源性度量技术研究, 软件学报 15 (Suppl.) (2004) 143–148.

[17] O. K. J, An algorithmic approach to the detection and prevention of plagiarism, SIGCSE Bulletin 8 (4) (1997) 30–41.

[18] Halstead, H. M, Elements of Software Science, Elsevier, 1997.

[19] H. Berghel, D. L. Sallach, Measurements of program similarity in identical task environments, SIGPLAN Notices 19 (8) (1984) 65–76.

[20] G. S. A, A tool that detects plagiarism in pascal programs, SIGCSE Bulletin 13 (4) (1980) 15–20.

[21] F. J. A. W, R. S. K, An empirical approach for detecting program similarity and plagiarism within a university programming environment, Computer Education 11 (1) (1987) 11–219.

[22] K. L. Verco, M. J. Wise, Software for detecting suspected plagiarism: comparing structure and attribute-counting systems (1996) 81–88.

[23] S. Schleimer, D. S. Wilkerson, A. Aiken, Winnowing: Local algorithms for document fingerprinting (2003) 76–85.

[24] J. H. Johnson, Identifying redundancy in source code using fingerprints (1993).

[25] J. H. Johnson, Substring matching for clone detection and change tracking (1994).

[26] S. Ducasse, M. Rieger, S. Demeyer, A language independent approach for detecting duplicated code (1999).

[27] G. Cosma, M. Joy, Towards a definition of source-code plagiarism, IEEE Trans. Education 51 (2) (2008) 195–200.

[28] O. A, Measure source code similarity using reference vectors, IEEE Computer Society 2 (2006) 92–95.

[29] 于世英, 袁雪梅, 卢海涛, 基于序列聚类的相似代码检测算法, 智能系统学报 02 (2013) 52–57.

[30] N. Kawamitsu, T. Ishio, T. Kanda, R. G. Kula, C. D. Roover, K. Inoue, Identifying source code reuse across repositories using lcs-based source code similarity (2014) 305–314.

[31] G. Kondrak, *N*-gram similarity and distance (2005) 115–126.

[32] Z. Li, S. Lu, S. Myagmar, Y. Zhou, Cp-miner: A tool for finding copy-paste and related bugs in operating system code (2004) 289–302.

[33] 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.

[34] Y. Semura, N. Yoshida, E. Choi, K. Inoue, Ccfindersw: Clone detection tool with flexible multilingual tokenization, in: 24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017, 2017, pp. 654–659.

[35] 刘楠, 韩丽芳, 夏坤峰, 曲通, 一种改进的基于抽象语法树的软件源代码比对算法, 信息网络安全 (1) (2014) 38–42.

[36] M. Chilowicz, É. Duris, G. Roussel, Syntax tree fingerprinting for source code similarity detection, in: The 17th IEEE International Conference on Program Comprehension, ICPC 2009, Vancouver, British Columbia, Canada, May 17-19, 2009, 2009, pp. 243–247.

[37] Z. L, L. D, L. Y, Z. M, AST-based plagiarism detection method, Springer, 2012.

[38] Y. Higo, Y. Ueda, M. Nishino, S. Kusumoto, Incremental code clone detection: A pdg-based approach, in: 18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011, 2011, pp. 3–12.

[39] Y. Higo, S. Kusumoto, Code clone detection on specialized pdgs with heuristics, in: 15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, Germany, 2011, pp. 75–84.

[40] Y. Higo, S. Kusumoto, Enhancing quality of code clone detection with program dependency graph, in: 16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France, 2009, pp. 315–316.

[41] A. H. Patil, K. Rangrajan, Generating effective test suite for multiparameter software using acts tool and its verification using code coverage tools.

[42] Q. V. Le, T. Mikolov, Distributed representations of sentences and documents, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, 2014, pp. 1188–1196.

[43] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[44] Y. Goldberg, O. Levy, word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method, CoRR abs/1402.3722.

简历与科研成果

基本情况 袁阳阳，女，汉族，1996 年 12 月出生，江苏省宿迁市人。

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

参与项目

1. 国家自然科学基金项目（重点项目）：智能软件系统的数据驱动测试方法与技术（61932012），2020-2024
2. 腾讯项目：软件测试课程实训体系与实验平台，2018-2019

专利申请

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：201810470192.0，等待实审提案。
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：201810470195.4，等待实审提案。

致　　谢

论文近尾声之际，我想要感谢所有给予过我关心和帮助的人，感谢你们在迷雾中给予我灵感，点拨我方向；在困难时施我以援手，赠我以信心；在我松懈时鞭策鼓励，督促向前。

首先要感谢我的导师陈振宇老师。在整个研究生生涯中，陈老师给予了我非常多的学术和工程指导。从毕设的初步想法、开题准备，到应用场景、研究思路，陈老师给出了很多建设性意见，并且为我答疑解惑；定期检查我的研究进展，指出设计方案可能存在的问题，督促我不断改进和完善。陈老师对待学术严肃认真，身体力行、主张实干。同时关注新兴技术并能与课题研究相结合，常常能给我们带来许多启发。

感谢实验室的黄勇老师、房春荣老师和徐剑锋老师，黄老师在工程实现上为我提供了许多思路，教会我化繁为简，从小的切入点入手，找到问题的解决方案。房春荣老师帮助我找出论文的问题和缺陷，提升了论文的整体水平，并与徐剑锋老师一起从工程和产品角度为我的毕设提出了十分宝贵的意见，并且在展示和重点突出方面给予了专业性指导。

感谢实验室的赵源、孙伟松学长，他们在实现细节和方案评估方面给予我重要支持，帮助我打破思维瓶颈。同时要感谢一直与我并肩战斗的韩奇同学和门铎同学，作为一直以来的队友、朋友和伙伴，他们帮我一起攻克了许多难题，也带给我许多关心和欢乐。

感谢南大，从本科到研究生，陪我走过六年大学生活。感谢南大为我提供的资源和平台，也感谢南大让我遇见可爱的老师和同学，大学生活即将画上句号，但南大永远铭刻心中。

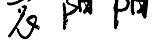
感谢我的家人，他们的支持为我提供了坚实的后盾，用爱、温暖和鼓励化作力量激励我不断前进。感谢研究生生涯给予过我关心和帮助的每一个人。

最后，感谢在百忙之中抽出时间参加我论文评审和答辩的专家们，向你们的辛勤和专业致敬！

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。

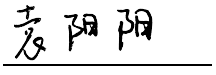
本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名： 

日期: 2020 年 5 月 28 日

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名： 

2020 年 5 月 27 日

论文题名	基于同源代码匹配的在线测试开发系统设计与实现				
研究生学号	MF1832230	所在院系	软件学院	学位年度	2020
论文级别	☐学术学位硕士 ☑专业学位硕士 ☐学术学位博士 ☐专业学位博士 （请在方框内画钩）				
作者 Email	MF1832230@smail.nju.edu.cn				
导师姓名	陈振宇				

论文涉密情况：

☑ 不保密

☐ 保密，保密期（_____年_____月_____日 至 _____年_____月_____日）