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　基于切片覆盖过滤的测试代码推荐系统的设计与实现

作　者　姓　名　门铎

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：**MF1832126**

论文答辩日期：**2020 年 5 月 23 日**

指 导 教 师：  **（签字）**

The Design and Implementation of Test Recommendation System Based on Slicing Coverage Filtering

By

Duo Men

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：基于切片覆盖过滤的测试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： 门铎

指导教师（姓名、职称）： 陈振宇 教授

摘 要

软件测试是软件生命周期中至关重要的一部分，而白盒测试是其中关键的一环，如何帮助无经验人员掌握白盒测试是一个值得关注的问题。传统的白盒测试学习过程通常需要使用到一些测试辅助工具，如覆盖率可视化工具和测试用例自动化生成工具等等。测试用例自动化工具产生的结果过多且可读性差，难以被借鉴。而覆盖率可视化工具产生的结果太过简约，不足以引导初学者进一步改进测试，提高测试质量。为了解决类似问题，一种快速高效的方式是在用户进行测试学习时，通过推荐与被测代码相关的测试代码片段的方式，帮助用户理解被测代码、引导用户挖掘测试方向、指导用户设计测试用例。

本文依托于 慕测 公司的 WebIDE 在线编程平台，设计和实现了一个基于切片覆盖过滤的测试代码推荐系统。本系统采用 Wala 作为程序切片和分析工具，处理 慕测 已有提交测试代码项目数据，得到代码片段集合；使用了 AST 程序分析技术将代码片段与项目模板融合；使用 OpenClover 工具分析代码片段的测试覆盖情况并存入语料库中。用户在测试学习过程中，系统会实时分析用户的测试覆盖信息，使用测试覆盖向量计算 Jaccard 向量相似度过滤得到推荐语料库中相关代码片段。动态地推荐给用户易于理解的优质代码片段，帮助其提高测试覆盖率，从而有助于初学者更快地理解源代码并掌握白盒测试。本系统划分为离线数据处理模块和动态代码推荐模块，其中离线数据处理模块用于构建测试代码片段语料库，动态代码推荐模块用于用户实时学习过程的追踪。本系统为提高吞吐量使用 Nginx 做负载均衡；为实现异步通信使用 WebSocket 技术进行服务器端主动推送消息；为了降低用户等待时间使用 ElasticSearch 提高查询性能；为了保证可扩展性，系统支持测试代码片段语料库的增量扩充。

本系统目前已在 慕测 平台测试环境经历了持续一个月的正常稳定运行，同时本系统构建了一个包含 11 道原创题目，内含 2200 多个测试代码片段的推荐语料库。本文通过可用性测试和可靠性测试的验证以及真实案例分析证明了系统可对无经验初学者的测试学习过程起到帮助作用并提高用户学习效率。

关键词：软件测试，程序切片，测试覆盖，代码推荐

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 Test Recommendation System Based on Slicing Coverage Filtering

SPECIALIZATION: Software Engineering

POSTGRADUATE: Duo Men

MENTOR: Professor Zhenyu Chen

Abstract

Software testing is a vital part of the software life cycle, and white box testing is an important part of it. How to help inexperienced people learn white box testing is a problem worthy of attention. The traditional white box test learning process usually requires the use of some test auxiliary tools, such as coverage visualization tools and test case automation generation tools. Too many results generated by test automation tools and readability is poor, difficult to learn. The results of the coverage visualization tool are too simple to guide beginners to further improve the test and improve the quality of the test. In order to solve similar problems, a quick and efficient way is to recommend test code fragments related to the code under test when the user conducts test learning, to help the user understand the code under test, guide the user to dig the test direction, and guide the user to design test cases.

This article relies on Mooctest WebIDE to design and implement a Test Code Snippets Recommendation System based on Slicing Coverage Filtering. This system uses Wala as a program slicing tool; AST program analysis technology is used to merge the code snippets with the project template; And Use the OpenClover tool to analyze the test coverage of the code snippets and store it in the corpus. During the user's test learning, the system will analyze the user's test coverage information in real time, and use the test coverage vector to calculate the Jaccard vector similarity filter to obtain the relevant code snippets in the recommended corpus. It also dynamically recommends high-quality code snippets that are easy for users to understand and helps them improve test coverage, thereby helping them understand source code faster and master white-box testing. The system is divided into an offline data processing module and a dynamic code recommendation module. The offline data processing module is used to build a

test code fragment corpus, and the dynamic code recommendation module is used to track the user's real-time learning process. This system uses Nginx for load balancing to improve throughput; WebSocket technology is used to actively push messages to achieve asynchronous communication; In order to reduce user waiting time, Elastic-Search is used to improve query performance; And In order to ensure scalability, the system supports incremental expansion of test code snippets corpus.

The system has now experienced normal and stable operation in the test environment of the Mooctest for a month. At the same time, the system has constructed a recommended corpus containing 11 original questions and more than 2,200 test code snippets. This paper proves that the system can help the test learning process of inexperienced beginners and improve user learning efficiency through the verification of usability testing and performance testing and real case analysis.

Keywords: Software Testing, Program Slicing, Test Coverage, Code Recommendation

目录

表 目 录

图 目 录

第一章　引言

1.1　背景与研究意义

从上世纪 50 年代的软件兴起，到 21 世纪的万物互联，软件在人们的日常生活中发挥着愈发重要的作用 [1]。但同时，随着软件应用场景的多样化与复杂化，软件架构和功能也变得越来越复杂。因此，软件能否正确并且稳定运行成为人们关注的一个重点问题 [2]。软件测试是保证程序软件可靠性的一个重要环节 [3]，社会对测试人员的需求量也因此变得越来越大。企业和学校纷纷开展测试教学工作，越来越多的人投入到软件测试学习中。在测试领域，白盒测试做为一种重要的测试手段 [4]，需要测试人员在理解程序内部逻辑的前提下，通过编写合适的测试用例提高软件的测试覆盖率来保证软件的可靠性 [5]。因此为培养大批量优秀的软件测试行业人才，如何快速有效地帮助初学者掌握白盒测试技巧，提高初学者的白盒测试学习效率变得尤为重要。

PractiTest 公司发布的《软件测试行业现状 2019 年度报告》[1]中对来自 80 多个国家的约 1500 名测试人员进行访问调查，结果显示超过四成的测试人员会选择亲自实践、阅读测试书籍以及在线研讨直播课程的方式提升测试技巧。而针对白盒测试的学习，初学者往往通过慕课学习测试相关技术并在本地进行亲自的编程尝试。但学习过程容易遇到壁垒，如没有入门题目、安装软件困难等问题。很多无经验人员卡在了如何安装学习辅助工具或者学习如何使用工具等问题上。并且很多测试辅助工具并不适合初学者在学习阶段使用，例如自动化生成测试用例工具和覆盖可视化工具等。这些辅助工具能够为测试经验丰富的工程师提供显著帮助，但针对测试学习场景下的无经验初学者，这些工具存在以下几点问题，一是自动化生成测试用例虽然可以有效地提高程序测试覆盖但却通常可读性差，无法帮助学生理解源码的执行过程 [6]，因此单纯地使用自动化测试生成不利于初学者通过测试理解源码和学习测试技巧。二是覆盖率可视化工具提供的报告不够直接 [7]，初学者在学习过程中难免会遇到测试覆盖率始终无法提高的情况或者无法理解为什么测试用例未能覆盖原方法的某一个分支，可视化工具无法在这两个方面给出直接有效的帮助。此时需要工具给出用户一些建议去进一步提高当前的覆盖率，而不仅仅是告诉初学者哪一行或者哪一个分支没有覆盖到。在初学者进行测试学习的过程中，提供准确、可读性强的测试用例是一种可行方式，能够直接有效的辅助初学者的学习。

[1]https://qablog.practitest.com/wp-content/uploads/2019/06/STO_2019_ver1_2.pdf

本论文旨在构建一个基于程序切片和覆盖分析的测试代码推荐系统帮助测试领域的初学者进行白盒测试学习。该系统以 慕测 WebIDE 智能在线编程平台为基础，使用程序切片技术对 慕测 近几年来通过比赛形式收集到的用户测试脚本数据进行分析整理并构建测试代码片段语料库，用户在使用 慕测 WebIDE 平台进行测试学习的过程中，动态地分析测试覆盖情况并推荐可执行、易理解的代码片段帮助用户提高被测项目的测试覆盖率。

1.2　国内外研究现状

1.2.1　编程代码推荐技术研究现状

目前学术界有很多针对编程代码推荐以及测试代码推荐的研究，对于编程代码推荐，学术界普遍做法是基于信息检索技术把代码片段当做一组令牌，使用自然语言处理的方式进行分析和索引，但程序语言和自然语言存在的巨大差别导致推荐效果一般 [8–10]。Lei Ai 等 [11] 的研究中考虑到在代码中最小有意义的单元是代码片段中的语句，通常来说是一行有意义的代码。过去的研究并没有考虑这个问题，所以代码片段推荐仍有改进的空间。于是 Lei Ai 等人提出基于代码语句序列信息的代码片段推荐系统 SENSORY。与基于令牌序列的研究不同，SENSORY 以代码语句的粒度执行代码段推荐，该工具首先统一代码片段语料库和用户编程上下文中源代码的编程风格，之后使用 Burrows Wheeler Transform 算法 [12] 处理代码片段，以获得更好的搜索结果。此外，它使用结构信息对搜索结果进行重新排序。为达到更好的推荐效果，该方法通过爬取互联网中已有的开源库信息的方式构建一个包含 100 万个真实环境中代码片段、超过 1500 万行语句的代码片段语料库，工具会比对用户的编程上下文和语料库中的代码片段搜索合适语料进行推荐。

除通过源码推荐源码的研究之外，Mohammad 等人 [13] 提出一种挖掘项目单元测试信息来推荐源码的方法，他们将源码所对应的测试用例用于辅助新代码的开发工作，即一种基于单元测试的新颖且通用的代码推荐方法。该方法通过挖掘单元测试的信息，用于帮助开发者在源码上的开发工作。Mohammad 等人的研究认为通过网络搜索 API 的方式辅助开发过程，利用查询相关测试用例也可以达到相同的目的 [14]。同时他们认为单元测试用例可以代表独立、一致且可信赖的信息源，可以合成有意义、有用的代码示例，并推荐给开发人员使用。具体的方法包含三个步骤：1）识别单元测试和被测试单元之间的关联关系，需要通过程序切片的方式分析每个单元测试用例断言出所测源码的方法，从而建立关联关系；2）从测试中合成代码示例，该过程需要使用抽象语法树分析相关

的技术进行代码片段提取；3）基于合成代码示例构建一个推荐系统，推荐系统中的推荐语料就是上述步骤中构建测试代码和源码关联关系。

1.2.2　测试代码推荐技术研究现状

测试在软件生命周期中扮演着越来越重要的角色，因此测试相关的推荐技术也开始被更多人所重视。在测试代码推荐测试代码领域中，Werner Janjic 等 [15] 针对测试编程推荐做了深入研究，他们发现缺陷测试开发仍是一个劳动非常密集的过程，其要求软件工程师具备高级别的领域知识、专注度和问题意识。因此，任何可以在这一过程中减少人工劳动的技术都可能显著地减少软件开发成本和时间消耗。他们实现了一个工具，通过重用已有测试用例中绑定的知识，帮助开发新软件组件和进行系统测试。他们构建了一个用于软件测试事前推荐的搜索工具，搜索工具所用资源库包含大约 200000 个 Junit 测试文件。之后通过 AST 提取抽象语法树分析得到每个测试方法的方法签名，使用方法签名和当前方法的方法体构造映射关系存入语料库中。推荐时会通过让用户输入一个包含方法签名的查询语句，去语料库中搜索方法签名相近的测试方法进行推荐。

在针对初学者测试学习的测试代码推荐研究中，R. Pham 等 [16] 在研究中发现软件开发人员十分依赖于项目之前存在的测试代码来学习如何编写新测试以及如何修改测试代码以供自己使用。因此他们建议从项目的测试套件中向新手战略性地提供有用的上下文测试代码示例，以便学习和编写测试。因此他们实现了一个 IDE 插件 “Test Recommender”，该工具的基本流程是使用版本控制工具分析用户最后一次更改内容，提取变更部分所使用到类，构建一个搜索集合。之后使用该搜索集合到测试代码语料库中检索相关测试用例。最后进行筛选和排序，将结果推荐给用户。

1.2.3　测试代码推荐工具研究现状

以往的研究中产生很多工具可以使用代码示例作为推荐内容帮助缺乏测试经验的初学者，Hummel 等开发 “Code Conjurer” 工具 [17]，该工具通过使用用户查询的语句来搜索相关代码代码示例，推荐数据的来源是从外部来源收集到的一些代码示例。Brandt 等研发 “Blueprint” 工具 [18]，该工具是一个 IDE 插件，可按需提供代码示例——包括其上下文信息，它会挖掘常规网站并从代码示例中获取描述。R. Pham 等人开发 “Test Recommender” 工具 [16]，它通过使用版本控制工具（SVN）来分析用户的变更内容，检索事先构建好的测试用例语料库进行推荐。Evosutie[2]是一个测试用例自动化生成工具，也可以做到测试代码推荐

[2]http://www.evosuite.org/

的功能，但是其生成的测试用例可读性差、易用性差的问题受到很多研究人员的质疑 [6]。

Code Conjurer 和 Blueprint 均集成在 IDE 中，让用户无需离开脱离 IDE 的工作空间即可进行测试学习，但都需要用户给出明确的查询语句。针对于无经验的初学者来说，他们并不清楚下一步要做什么，也就无法给出查询语句，因此无需显式查询的推荐机制更适合初学者。Test Recommender 工具提供通过分析用户变更内容的方式替代查询语句的方式来解决此问题，但 Test Recommender 依赖版本控制工具的特性，导致用户的上手难度很大，不适合无测试经验的初学者使用。并且该工具通过分析本项目中已有测试用例的机制，当测试目标是一个无测试用例或者极少测试用例的项目中时，对初学者学习的帮助作用甚微。

1.3　本文主要研究工作

针对市面上初学者测试学习遇到的诸多问题，如测试工具上手难度大、工具需要进行显式的输入查询、自动化生成的测试用例随机性高和可读性差等等。本文实现一个基于程序切片和覆盖分析的测试代码推荐系统。本文展示的测试代码推荐系统依附于 慕测 WebIDE 智能编程平台，其主要作用是初学者在 慕测 WebIDE 平台上进行 Java 白盒测试学习过程中，推荐有助于初学者理解程序和提高当前测试覆盖的测试用例，帮助初学者提高白盒测试技能和程序分析理解能力。

为达到此目的，本论文通过以下三个方面的设计来实现该目标：

（1）本文借鉴以往推荐研究中的思路，即通过网络爬取开源代码库的方式构建一个用于推荐的测试代码片段语料库。但不同于直接爬取开源代码库，教学场景下，我们很容易收集到用户自发产生的学习过程数据。本系统利用 慕测 平台在以往测试大赛和日常练习考试中收集到的学生提交的测试项目数据来构建测试代码片段语料库。特别的， 慕测 平台每年会产生大量的学生提交记录，针对于每个测试项目均有来自不同学生提交的大量测试数据据存储于服务器中。学生提交数据的特点是针对于同样的测试内容，给出不同的测试结果，同时学生提交的测试代码数据相比自动化生成的测试用例可读性强。因此，本篇论文的方法是借助 慕测 平台已有的测试数据，提取学生提交内容中测试代码用例，得到具体的代码片段，用于推荐给使用本系统进行测试学习过程中的其他初学者。

（2）获取用户提交数据中优质代码片段。优质代码片段在本文中的定义是一段精简、可读性好、覆盖信息丰富、可帮助用户理解源码的代码片段。我们从 慕测 平台收集到的用户提交数据因其质量参差不齐，单个测试脚本不够精简无法直接用做推荐。例如很多用户提交数据中使用自动化生成工具产生很多冗余信

息、或者在测试过程中由于较差的编程习惯将所有的测试内容写入一个测试方法内。对于此问题，传统解决方式是通过抽象语法树或者文本分析的方式，将单个测试方法拆分成多个测试方法。但在测试学习场景下，初学者提交的测试脚本可能只包含一个测试方法，且将所有的测试内容写在此方法中。为解决这个问题，本系统使用程序切片技术 [19]，对学生提交数据中的测试脚本进行分析和切片，得到更多精简、覆盖信息丰富、有助于程序理解的优质测试代码片段。

（3）分析代码片段的覆盖信息作为推荐语义。初步得到的代码片段是不存在任何语义的，本文使用测试覆盖率分析技术对测试代码片段进行覆盖率分析，得到代码片段针对于被测程序的覆盖率分析结果，该覆盖分析结果的表现形式为每一个测试代码片段均可通过覆盖信息映射到原被测代码的某一个分支或者代码行上。最终通过此方式构建一个针对不同源码的测试代码片段语料库。初学者在使用 慕测 WebIDE 进行测试学习过程中，系统通过实时分析当前用户的覆盖信息，进行实时的覆盖结果相似度匹配，从语料库中选择合适的代码片段进行推荐。

结合上述的项目需求和技术选型，本系统使用 SpringBoot 框架进行快速开发，构建基于切片和覆盖分析的测试代码推荐系统。本文设计离线数据处理模块，该模块用于将已有的测试脚本数据进行切片得到测试代码片段，将得到的代码片段融入测试模板并进行测试覆盖率分析工作，最后得到完成的测试代码语料库持久化到数据库中。本文设计动态代码推荐模块，该模块用于分析初学者在测试学习过程中提交的测试脚本得到运行时测试代码的覆盖信息，并使用向量相似度计算的算法选择合适的语料库中的代码片段进行推荐。考虑到传统的关系型数据对于大文本的检索效率问题，本系统使用 ElasticSearch 作为存储测试代码语料的持久化引擎，提高用户推荐过程实时查询效率。同时使用 Docker 技术做为持续集成和持续部署服务的载体，封装本系统涉及的所有模块，规范开发过程和部署环境。

1.4　本文的组织结构

第一章 引言。主要介绍在线测试教学，特别是白盒测试教学场景下，现有学习方式的缺陷以及相关学习工具存在的诸多问题。之后通过介绍代码推荐研究现状、测试代码推荐研究现状以及测试代码推荐工具研究现状，引出目前针对初学者进行测试学习遇到的困难和问题，最后为解决这些问题介绍本文主要工作。

第二章 技术综述。主要介绍覆盖率分析工具 OpenClover 和程序切片工具 Wala 的工作原理和用途，介绍向量相似度计算方法相关内容并特别介绍本文用

到的相似度计算方法 Jaccard 的相关内容。之后介绍系统开发用到的服务端相关技术，如搜索引擎 ElaticSearch 和 Docker 容器技术。在介绍完每个技术后，给出其对应优势，以及本文选择该技术的原因。

第三章 系统需求分析与设计。主要介绍设计该系统考虑到的功能性和非功能性需求，之后根据系统需求给出系统用例描述，然后从 4+1 视图的角度详细介绍系统总体架构，最后为更详细地描述系统设计，分别介绍代码片段切片模块、切片模板融合模块、覆盖分析模块、代码推荐模块的架构设计和数据库存储结构设计。

第四章 系统详细设计与实现。在第三章设计的基础上，本章对文本代码片段切片模块、切片模板融合模块、覆盖分析模块、代码推荐模块的实现细节进行详细分析，分别介绍各个模块的流程设计、类图设计、时序图设计并给出相关模块的核心代码实现，并在本章最后给出系统最终呈现效果的截图和文字描述，最后通过相关测试和实验的设计给出系统存在的意义和必要性。

第五章 系统测试与案例分析。本章首先介绍从系统可用角度出发的功能性测试。之后通过可靠性评估的性能测试评估单机系统的承载压力，为系统的弹性扩展提供数据支持。最后通过详细的案例，展示系统如何帮助无经验初学者的测试学习过程。

第六章 总结与展望，分析系统目前达到的效果，做出整体性总结，并给出系统仍然存在的不足和问题，展望下一步系统的迭代计划。

第二章　技术综述

本章介绍系统使用相关技术、算法和框架的简要内容。其中离线数据处理模块在构建测试代码片段语料库的过程中使用程序切片技术和覆盖率分析技术。动态推荐模块需要用到相似度度量算法过滤候选推荐结果。整个系统的搭建和部署用到 SpringBoot 框架和 Docker 容器技术。测试代码片段存储使用持久化组件 ElasticSearch 搜索引擎。

2.1　程序切片技术

2.1.1　程序切片技术介绍

在上世纪 70 年代，软件复杂度骤增，如何对越来越复杂的软件进行维护和调试成开发人员最关心的热点问题。研究者在那时发现：一个体积巨大的软件被分解为若干个相互不影响的程序片段时更容易进行调试和维护，于是相继有研究人员提出程序分解技术和方法。Mark Weiser 在分析程序源码和控制流图（CFG）之间关系的过程中发现程序输出仅和源码中的部分语句有关，删除无关语句不会影响程序运行结果 [20]。Mark Weiser 把从源码中删除无关语句后得到的程序片段叫做代码切片，而得到代码切片的技术就是程序切片技术。

程序切片经过多年的发展出现了多种不同的切片技术，其中应用最为广泛的两种切片技术分别是基于数据流的切片技术 [21] 和基于图可达性分析的切片技术 [22, 23]。基于数据流的切片技术首先要构建出源程序的控制流图（CFG），控制流图中的节点为程序指令块，节点与节点之间的关系为指令块之间的控制流路径，之后通过分析计算指令块间的控制关系，将产生关联的节点添加到统一的集合中，构成最终的代码切片。基于图可达性分析的切片技术则首先要建立源程序的程序依赖图 (PDG)[24] 或者系统依赖图 (SDG)[25]，图上节点表示程序语句，图上边表示语句依赖关系，包括控制依赖、数据依赖等等不同的依赖类型。得到最终切片代码的方式为图遍历并剪枝得到子图，根据子图得到最终切片。基于图可达性分析的切片技术的处理过程可分为三步：1）构造整个源程序的系统关系图。2) 在图上寻找切入节点，以该节点为入口进行图遍历，构造对应切片程序的子关系图。3) 将该关系图映射到源代码上的具体语句得到切片代码。

程序切片技术广泛应用在逆向工程 [26]，测试生成 [27]，软件调试 [28] 以及程序验证 [29] 等领域。如在软件调试领域，可通过对调试点 D 进行切片，得到和调试点 D 相关的代码语句，之后对得到的代码语句进行分析就可快速定位

出错代码，这样做的好处是可以屏蔽掉无关语句，降低分析代码和定位出错位置的难度。

2.1.2　程序切片工具 Wala

Wala [1]是 IBM 公司于 2006 年开源，是一个支持 Java 语言以及 JavaScript 语言的静态和动态程序分析库。Wala 提供的核心特性包括：Java 系统类型分析和类层次结构分析 [30]、过程间数据流分析、基于上下文敏感的数据切片、指针分析 [31] 和调用图生成等。Wala 因其支持的功能全、鲁棒性好、效率高和可扩展性强的特点在程序分析领域被很多研究人员以及工业界的开发者使用。

Wala 提供两种算法用于程序切片：computeForwardSlice 前向切片算法和 computeBackwardSlice 后向切片算法。这两种切片算法均为上文中提到的基于图可达性分析切片技术的具体实现。

2.1.3　Wala 切片流程介绍

首先，Wala 需要定义切片的作用域，Wala 仅会分析作用域内的资源。以 Java 语言为例，资源包括 class 文件、jar 包等数据。之后 Wala 会分析作用域内的资源得到整体类层次关系图。

接着，根据得到的层次关系进行遍历分析后得到整个作用域的调用关系图。

接着，开发者进行切片入口点的提取，提取的内容为源码中的一行语句来代表入口点。之后将提取到的切点语句映射到上一步得到的调用关系图中相关节点 Statement 上用于分析使用。

同样，开发者需要进行切片结束点的选择，结束点代表切片算法在遍历系统关系图中停止条件，每一个结束点代表源码中的一行语句，对应于系统关系图中的一系列节点 Statement。

最后，Wala 根据切片入口点和结束点对调用关系图进行前向遍历或后向遍历得到所有从入口点语句到结束点语句途径的所有节点。将得到的节点转化为源码中的代码行即可得到最终的代码切片。在本文中的结束点仅表示为测试方法声明语句所在行，切片入口点为 assert 断言语句所在代码行。

本文使用程序切片技术处理 慕测 平台上学生测试项目数据。每个提交的测试项目中均包含针对某一道题目的相关测试代码脚本文件。用户在使用平台编写测试脚本的过程中，往往将很多可拆分的断言片段杂糅在一起，例如一个测试方法中包含全部的测试用例内容。此问题导致本系统在进行测试代码片段推荐时无法直接使用方法级别的代码片段。因此需要使用程序切片技术针对于每

[1]http://wala.sourceforge.net/wiki/index.php/Main_Page

个断言语句进行切片处理，提取断言语句相关的的语句集合。经过整理得到的测试代码片段仅包含一个断言语句，只测试源代码中的一个方法。通过切片得到的代码片段足够原子，在进行推荐时可做到针对某个分支或者某个方法的精确推荐。

2.2　覆盖率分析工具 OpenClover

2.2.1　OpenClover 简述

代码覆盖率是衡量测试脚本优劣的一个参考标准 [32]，代码覆盖率可以表示当前测试用例对于源代码的覆盖情况。代码覆盖率会在开发人员进行软件测试中起到反馈作用，帮助开发者尽早发现程序中的 Bug。OpenClover [2]是一个代码覆盖率分析工具，其工作流程大致可分为三个步骤：首先在待测源码中插入桩代码，之后将插桩后的代码编译为字节码文件，最后通过运行测试，查看桩代码的执行情况并将结果汇集成一份测试覆盖率报告。

OpenClover 分析得到的测试覆盖率报告如下图 2.1所示，file 块可表示为一个源代码文件，当存在多个源代码文件时会存在多个 file 块。line 块表示一条覆盖信息，块中属性 count 表示该条覆盖信息是否有相应测试代码进行覆盖，type 表示该条覆盖信息的种类：method 表示方法覆盖；stmt 表示语句覆盖；cond 表示分支覆盖。

2.2.2　OpenClover 覆盖率种类

OpenClover 工具提供三种类型覆盖率 [3]来综合度量每个类、每个包以及每个项目的整体覆盖情况：

（1）语句覆盖：最基本的测试覆盖类型。用来衡量源代码语句是否被测试代码成功执行，这种覆盖类型不考虑语句逻辑只考虑是否被执行。通常情况下，语句覆盖率仅会仅会计算有效代码行，空行和注释等不会计算在内。语句覆盖和行覆盖区别在于一行代码可以包含多个语句，语句覆盖会比行覆盖更能体现程序的覆盖情况。

（2）分支覆盖：也称决策覆盖，用于度量程序流程控制的覆盖类型。要求将所有分支判断通过与不通过的情况各取一次，且不考虑每个判断内部判定逻辑的取值情况。通常情况下，分支覆盖也较为容易取得较高的覆盖率，也是评价系统覆盖情况的重要参考指标。OpenClover 通过记录执行过程中控件结构的布尔表达式是否为 True 和 False 来进行记录。

[2]http://openclover.org/index

[3]http://openclover.org/doc/manual/latest/general–about-openclover-code-metrics.html

```
<?xml version="1.0" encoding="UTF-8"?>
<coverage generated="1583224791778" clover="4.4.1">
    <project name="Tmp 0.0.1-SNAPSHOT" timestamp="1583224790027">
        <package name="net.mooctest">
            <file path="… " name="ALU.java">
                …
                <line complexity="6" visibility="public"
signature="integerRepresentation(String,int) : String" num="13" count="0"
type="method"/>
                <line num="14" count="0" type="stmt"/>
                <line falsecount="0" truecount="0" num="17" type="cond"/>
            </file>
        </package>
    </project>
</coverage>
```

图 2.1: OpenClover 测试覆盖报告

（3）方法覆盖：OpenClover 中方法覆盖表示源码中的方法是否被测试代码覆盖。如果测试代码用例调用被测方法，就可表示为成功覆盖。

2.2.3　OpenClover 优势

目前市面上有很多成熟的覆盖率分析工具，如 Cobertura[4]、JaCoCo[5]、JCov[6]等，它们均提供强大的覆盖率分析功能，OpenClover 给出了相关工具全方面的对比表[7]。OpenClover 相比其他工具使用源码插桩技术，而 Jacoco，JCov 等工具使用字节码插桩技术。相比字节码插桩技术，源码插桩更强大、更灵活，并解决源码翻译成字节码后会导致覆盖率不准确的问题。同时 OpenClover 支持 HTML、XML、JSON、文本等格式的测试报告，在数据可视化和数据处理方式选择上面更加灵活，用户可通过选择适合应用场景的测试报告格式进行二次开发和使用。

本系统在构建测试代码片段语料库的过程中，会对程序切片得到的代码片

[4]http://cobertura.github.io/cobertura/

[5]https://www.jacoco.org/jacoco/

[6]https://wiki.openjdk.java.net/display/CodeTools/jcov

[7]http://openclover.org/doc/manual/latest/general–comparison-of-code-coverage-tools.html

段使用 OpenClover 工具进行分析得到测试覆盖报告。分析测试覆盖的目的是建立源码和测试代码片段之间的关联，该关联关系通过覆盖报告产生的覆盖向量维系。将测试覆盖报告和测试代码片段进行映射处理后存入到语料库中，用于之后的动态推荐过程。用户使用系统进行动态推荐过程同样会使用 OpenClover 实时分析用户所提交的测试脚本信息，提取其中的测试覆盖报告并分析语句和分支的覆盖情况。针对没有覆盖到的语句和分支，系统会从语料库中使用相似性算法筛选出合适的测试代码片段进行推荐。

2.3　相似度度量方法

2.3.1　相似度度量方法介绍

相似度算法常常在分类、聚类等过程中评估两个样本的相似程度。常见的向量的相似度算法包括欧式距离、曼哈顿距离、闵可夫斯基距离、切比雪夫距离、汉明距离、余弦相似度、皮尔森相关系数、Jaccard 相似度等算法 [33]。其中 Jaccard 相似度算法是一种计算两个给定集合相似性和差异性的算法。在本文中会计算两个 01 二进制向量的相似度，因此使用 Jaccard 相似度算法最为合适。

2.3.2　Jaccard 相似度度量算法

Jaccard 相似度算法如公式（2.1）所示，变量 A 和 B 均代表一组元素的集合，A 集合和 B 集合的交集与并集的比值为 A 和 B 的 Jaccard 相似度。两个集合的 Jaccard 相似度值越大则证明两个集合的相似度越高。当集合 A 和集合 B 均为空时 $Jaccard(A,B)=1$。

在本文中可将用户提交测试代码的覆盖情况定义为 $X=(x_1,x_2,x_3,...,x_n)$ 的覆盖向量，测试语料库中代码片段的覆盖情况定义为 $Y=(y_1,y_2,y_3,...,y_n)$。则最终相似度值可用向量 X 和向量 Y 的 Jaccard 相似度如公式（2.1）计算得出。

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A|+|B|-|A\cap B|} \tag{2.1}$$

在用户编写白盒测试用例的过程中，本系统将采用以下策略进行测试片段推荐：首先分析得到用户提交代码和测试语料库中代码片段的覆盖向量，之后采用上述相似性算法计算两个向量之间的相似度，最终过滤得到匹配度最高的 TopK 个代码片段推荐并推荐给用户。

2.4　搜索引擎 ElasticSearch

2.4.1　ElasticSearch 简述

ElasticSearch[8]是一个基于 Apache Lucene[34] 高可扩展的开源全文搜索和分析引擎，它的实现主要使用 Java 语言，它是工业界用于企业级搜索常用的框架之一 [35]。它允许存储文本、数、结构化、非结构化的数据用于大规模的分析和检索服务，并且实时性高、支持集群弹性扩展。它通常被用作底层引擎和技术，为复杂的搜索功能和要求提供动力。ElasticSeaech 的架构如图2.2所示。

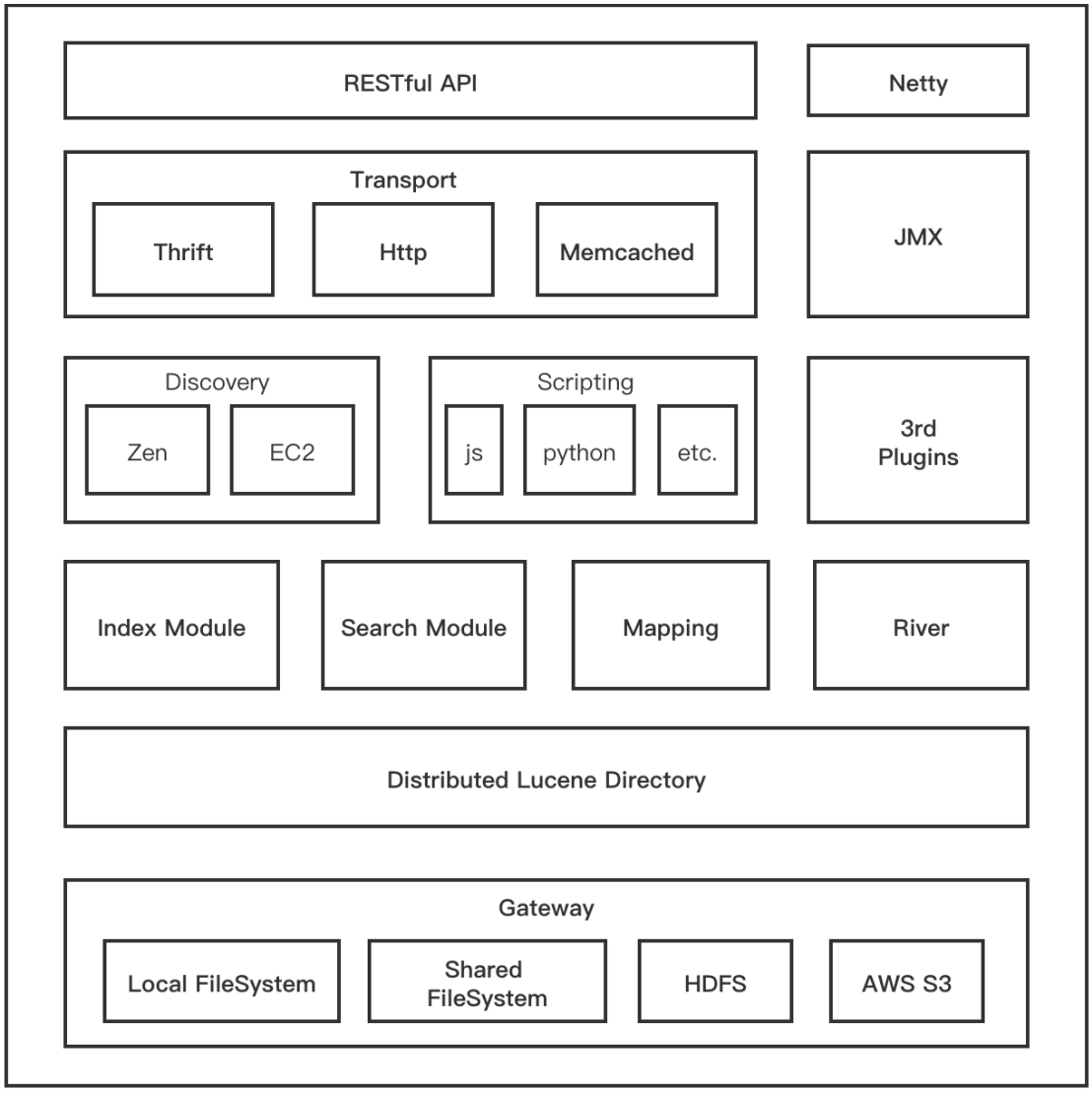

图 2.2: ElasticSearch 架构图

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

其中，底层 Gateway 用于适配不同的文件存储系统，包括本地文件系统、分布式文件系统、HDFS 等等，ElasticSearch 默认将数据存储到内存中，之后通过 Gateway 持久化到文件系统。上一层是 Lucene 分布式检索引擎，ElasticSearch 强大搜索能力就是建立在 Lucene 之上的。Lucene 之上是 ElasticSearch 提供的基础服务模块：Index 模块类似于关系型数据库中表的概念；Search 模块用于检索；Mapping 模块用于数据映射；River 则负责外在数据导入 ElasticSearch 中。基础模块之上是 ElasticSearch 为了集群部署以及主从备份使用到的 Discovery 模块，以及对于 Lucene 扩展的脚本查询引擎和第三方插件模块。最上面两层则是 ElasticSearch 暴露给外围系统使用的通信协议和交互接口。

2.4.2　使用 ElasticSearch 的优势

（1）扩展性：ElasticSearch 提供强大的横向扩展能力，既可以通过扩展新的服务器到 ES 集群中，也可运行在单机上作为轻量级搜索引擎使用。

（2）功能丰富：ElasticSearch 提供全文检索、同义词处理、相关度排名、复杂数据分析、海量数据的近实时处理等功能。

（3）使用简单：ElasticSearch 提供 RESTful 形式的 API 接口，这种通用的处理方式使用简单，大大降低开发时间成本。同时 ElasticSearch 提供 Docker 镜像供开发者使用，降低繁琐的服务安装和部署时间。

本文使用 ElasticSearch 作为底测试代码语料库的存储和搜索引擎，通过 ElaticSearch 提供的强大索引能力，可做到实时的数据查询，降低用户得到测试推荐结果的所用等待时间。

2.5　容器技术 Docker

2.5.1　Docker 介绍

Docker[36] 是一个使用 Go 语言开发的，使用 Apache2.0 开源协议的容器引擎。Docker 提供的核心功能是将开发者开发的应用打包成可运行的标准镜像，并通过简易命令的方式部署应用到任意位置 [37]。Docker 提供的声明式脚本可帮助开发者快速创建一个轻量级、可移植性强的容器。通过使用该容器可做到在任何环境下轻松部署。与 Docker 相类似的是虚拟机技术，但虚拟机存在资源占用多，部署成本高，应用启动慢等缺点，目前已逐渐被企业界弃用。各大企业纷纷转而使用 Docker 作为服务运行托管平台。

2.5.2　使用 Docker 优势

Docker 具有以下几个鲜明的特点：

（1）轻量: 在正常的使用中，每个 Docker 容器均可映射到服务器上的一个进程。部署在同一台服务器上的容器共享服务器资源。

（2）跨平台: 通常服务需要部署在各种不同版本的 Linux 服务器上，而开发会选择 Windows 或者 Mac 等系统作为开发机。Docker 则屏蔽了这种开发和部署环境差异，让用户的服务可透明在不同种类的服务器或者开发机上运行。

（3）安全: Docker 基于 LXC 容器技术实现，底层实现借助 Linux 的 Namespace 和 Cgroup 机制。可以做到容器内不共享用户组、命名空间、磁盘、内存、CPU 等资源。这样的机制保证服务的安全性不受外界干扰。[38]

本文所搭建的系统涉及多个独立的微服务，为方便快捷地测试和部署服务，所有服务均使用 Docker 打包成可运行的镜像上传到公司 Docker 私服中，通过编写 Jenkins 自动化构建脚本的方式，将服务部署到任意服务器中。

2.6　开发框架 SpringBoot

2.6.1　SpringBoot 介绍

SpringBoot 是一个由 Java 语言开发，快速搭建 Web 服务的开发框架。它由 Spring 框架发展而来。Spring 因其开源的特性受到了很多 Java 工程师的欢迎，其提供的 IOC 和 AOP[39] 特性更是让 Java 的开发效率大大提高。但随着项目的规模越来越大、项目越来越复杂，Spring 提供的 XML 配置方式也变得越来越不方便。因此 Pivotal 团队基于 Spring 框架开发和设计了 SpringBoot 框架 [40]。

2.6.2　使用 SpringBoot 优势

本文使用 SpringBoot 作为构建系统的后端框架，考虑到了 SpringBoot 作为开发框架具有以下几点优势：

（1）简化部署：内嵌 Java 服务器，支持 Jetty、Tommcat 等多种 Web 服务器，无需额外部署操作，降低部署成本。

（2）简化配置： SpringBoot 提供多种高效注解，代替复杂的 xml 配置文件。使用“约定大于配置”的设计思想降低程序的复杂程度。

（3）简化编码：通过 Starter 组件快速完成诸如 ElasticSearch、Redis 和 RabbitMQ 等组件的整合。通过导入依赖的方式，大大降低开发者编写相关整合代码的整合难度。

2.7　本章小结

本章主要概括介绍开发此系统用到的相关算法、框架和技术。首先介绍如何从测试脚本中得到针对断言语句的程序切片技术以及对应的工具 Wala 的使用方式和工作流程。接着介绍对测试代码片段进行覆盖率分析的工具 OpenClover。第三，介绍推荐时需要用到的相似度计算方法 Jaccard 算法。第四，介绍存储测试代码片段语料库的搜索引擎 ElasticSearch，ElasticSearch 提供的全文索引能力可大幅加快推荐代码片段的筛选。第五介绍系统编排和部署的容器工具 Docker，最后介绍系统构建需要的服务端组件 SpringBoot。

第三章　系统需求分析与概要设计

本章的主要内容为系统的需求分析和概要设计。首先，对系统进行总体描述，分析并得出系统需求；然后，将需求细化为功能性需求和非功能性需求两部分，并进行用例设计、给出用例描述；最后，给出系统的概要设计，并划分出基本的功能模块。本章将利用 4+1 视图对项目的整体架构、离线处理模块和数据持久化模块的相关设计进行描述。

3.1　系统整体概述

本论文主要讨论如何构建一个基于程序切片和覆盖分析的测试代码推荐系统，用于帮助进入到测试领域的初学者学习白盒测试技能。该系统以 慕测 WebIDE 智能在线编程平台为基础，首先使用静态程序切片技术对 慕测 近几年来通过比赛形式收集到的用户测试脚本数据进行程序切片，接着将得到的代码片段使用覆盖率分析工具进行覆盖分析，最后整理所得结果并构建测试代码片段语料库。用户在使用 WebIDE 平台进行测试学习的过程中，系统实时分析用户测试覆盖情况，并推荐可执行、易理解的代码片段，帮助用户提高被测项目的测试覆盖率。根据以上思路，本系统使用 SpringBoot 框架搭建系统并进行开发。本文设计了离线数据处理模块，该模块用于将已有 慕测 提交的测试脚本数据进行切片，将得到的代码片段融入测试模板并执行覆盖率分析，最后得到完成的测试代码语料并持久化到数据库中。本文设计了动态代码推荐模块，该模块用于分析初学者在测试学习过程中提交的测试脚本得到运行时覆盖报告，并使用相似度计算方法选择合适的语料库测试代码片段进行推荐。

3.2　系统需求分析

3.2.1　功能性需求分析

通过上述分析，我们将系统整体需求拆分成两部分，分别是测试代码片段的离线数据处理和测试用例的动态推荐。本文按照这两个模块进行需求分析，所有主要功能性需求如表3.1所示。

离线数据处理模块主要包括测试脚本切片、项目模板构造、测试覆盖分析和测试用例语料库构建。本部分的最终目的是构建一个可以在动态推荐模块中使用的测试用例语料库。

测试用例动态推荐模块主要包括了编程行为分析和推荐结果获取。该模块功能在用户运行代码时被动触发。编程行为分析的目标为获取运行产生的测试覆盖报告，推荐结果获取的目标是从离线数据处理模块中构建的测试用例语料库中搜索最佳测试用例进行推荐。

表 3.1: 功能需求列表

需求 ID	需求名称	需求描述
R1	查看推荐结果	用户可在编程过程随时查看系统推荐的测试用例，该页面会在有推荐结果返回时出现。
R2	触发动态推荐	用户可在编程过程中触发代码推荐，该过程一般通过运行代码被动触发。
R3	覆盖报告分析	系统在用户触发代码推荐后，分析用户运行测试代码示例产生的覆盖报告，并根据报告提取出需要进行推荐的待测方法签名。
R4	推荐结果获取	用户可在 慕测 平台进行开发者测试学习的过程中获取到测试代码片段。
R5	推荐设置	用户可在 WebIDE 系统进行推荐设置，例如调整推荐结果数、推荐频率等配置信息。
R6	测试脚本切片	系统可对 慕测 以往用户产生的测试脚本进行切片处理，产生独立可运行的测试代码片段集。
R7	项目模板构造	系统将得到的每一个代码切片融入测试模板中得到一个完整的可运行的 maven 项目。
R8	构建测试用例语料库	系统对产生的 maven 项目进行测试覆盖分析得到覆盖报告, 通过该报告可分析出该代码片段的覆盖向量，可表明测试到了待测源代码的哪些方法和分支。 系统将测试覆盖报告和对应的测试代码切片进行关联共同构成测试用例语料并进行持久化存储。

3.2.2　非功能性需求分析

测试用例推荐系统使用切片技术分析 慕测 平台用户提交的测试代码脚本，并构建测试用例语料库用于用户进行开发者测试学习的实时推荐过程。通过上面的分析，得到系统需满足的功能性需求，同时根据实际使用场景的要求，系统需要满足的非功能性需求如下：

(1) 系统易用性 系统应是易于操作的。系统不应因复杂的操作流程干扰用户的使用流程。用户在使用系统过程中，可通过直观、简洁的界面看到自己需要的推荐代码片段。

(2) 系统扩展性 系统应是可扩展的，系统的各个组件如切片功能，推荐获取功能等要做到易替换。为此系统要做到高内聚低耦合，替换过程不影响系统原有功能。系统构建的测试代码片段语料库应是可扩展的，可随时进行代码库的增量扩充。此外，随着用户规模的扩大，系统应具备弹性扩容能力。

(3) 系统可用性 系统的可用性是任何系统存在的前提条件。针对测试用例推荐系统，应通过日志、监控等途径保证系统持续稳定运行。同时选择合适的存储组件保证高并发情况下系统不会宕机。

3.2.3　系统用例图

根据上一节描述的功能性需求，测试用例推荐系统用例图如图 3.1所示。作为平台使用者的测试学习人员可以在系统上在线编程、运行自己编写的测试代码并且进行推荐参数设置。其中用户每次运行代码都会触发动态推荐过程，后台系统会分析用户提交代码的测试覆盖数据，并通过算法过滤得到合适的测试代码片段进行推荐。作为系统开发人员的后台管理者要进行推荐语料库构建的离线数据处理工作，分别包含了程序切片、项目模板构造、测试覆盖分析、生成待推荐语料、以及测试语料持久化等功能。

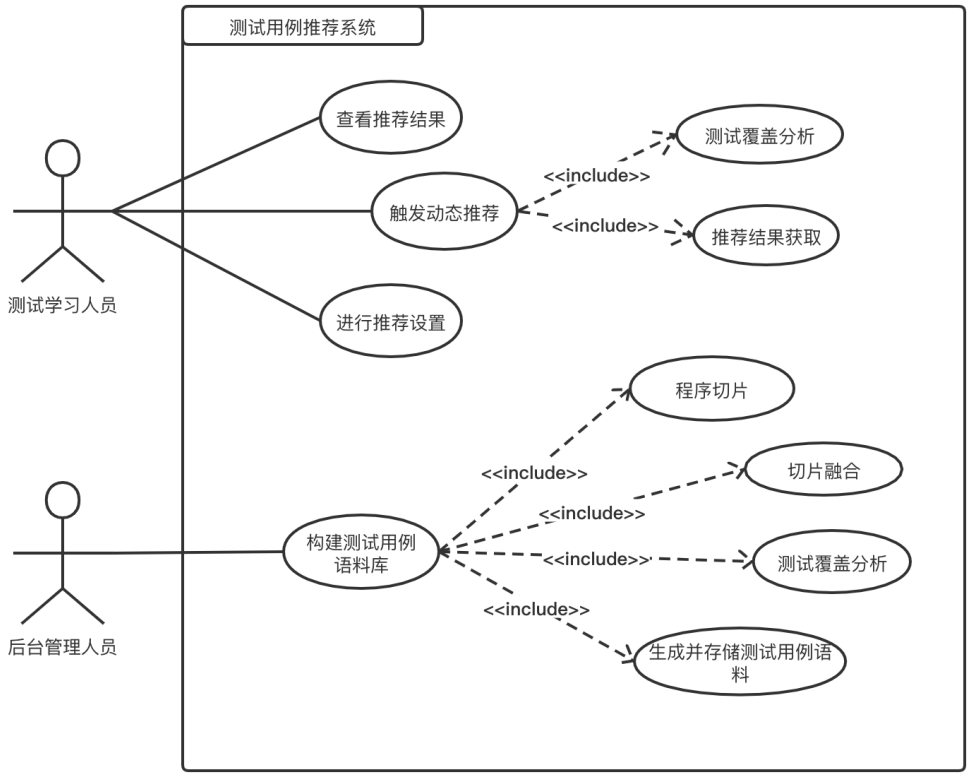

图 3.1: 系统用例图

3.2.4　系统用例描述

通过上一节分析得到了系统的功能性需求：查看推荐结果、触发动态推荐、覆盖报告分析、获取推荐结果、推荐设置、测试脚本切片、项目模板构造、测试

覆盖分析、构建测试用例语料库。下文将进行进一步的分析得到它们的系统用例描述并进行详细介绍。

查看推荐结果的用例描述如表3.2所示。推荐结果查看的功能是系统用于给用户返回推荐测试用例。该界面包含了系统提供的参考测试用例详细信息，测试用例以待测方法签名为标签进行分组展示。

表 3.2: 查看推荐结果用例描述

ID	**UC1**
名称	查看推荐结果
参与者	参与者：平台使用者和测试学习用户 目标：查看推荐的测试用例列表。列表中包含的信息根据方法签名进行智能分类。用户可通过选择一个测试方法，查看该方法相关的推荐测试用例
触发条件	用户在 WebIDE 平台上运行了单元测试
前置条件	用户点击“运行”
后置条件	学生代码编译运行成功
优先级	高
正常流程	1. 用户点击侧边栏“测试代码推荐” 2. 用户点击“方法签名”，进行待测方法选择 3. 用户浏览下方以列表形式展示的推荐测试用例
扩展流程	1a. 用户点击侧边栏“测试代码推荐”，无推荐结果 2a. 用户点击“方法签名”，显示“当前无可推荐测试用例”

触发动态推荐的用例描述如表3.3所示。该功能是用户运行代码的后续操作，用户在进行开发者测试学习时可以通过“运行”功能运行所写单元测试，每次运行均会触发动态推荐。本操作被设计为被动式，大大降低用户的操作学习成本。

表 3.3: 触发动态推荐用例描述

ID	**UC2**
名称	触发动态推荐
参与者	参与者：平台使用者和测试学习用户 目标：调用动态推荐模块，运行推荐相关业务逻辑
触发方式	提交代码后自动触发
触发条件	用户在 WebIDE 平台上点击“运行”按钮运行了单元测试
前置条件	无
后置条件	无
优先级	高
正常流程	1. 用户点击导航栏” 运行” 2. 代码运行成功并返回“成绩”和“日志”信息 3. 系统发送异步事件请求“触发动态推荐” 4. 建立 WebSocket 结果返回管道

覆盖报告分析在用户触发动态推荐操作后执行，其用例描述如表3.4所示。该过程用于提取用户产生的覆盖报告结果得到所测方法的覆盖率分数。具体流程是将运行单元测试后得到的 xml 测试覆盖报告转化成简单、易理解的 json 格式数据。之后分析得到用户正在测试的各个方法的相应覆盖率分数，并按时间序列数据的形式持久化到 DB 中。

表 3.4: 覆盖报告分析用例描述

ID	**UC3**
名称	覆盖报告分析
参与者	参与者：平台使用者和测试学习用户 目标：运行单元测试得到测试覆盖报告，解析测试覆盖报告，分析得到用户正在测试的各个方法以及各个方法的相应的覆盖率，并以时间序列数据的形式持久化到 DB 中。
触发条件	用户在 WebIDE 平台上点击“运行”
前置条件	用户触发动态推荐
后置条件	无
优先级	高
正常流程	1. 系统解析用户产生的测试覆盖报告，转化为 json 格式的测试报告 2. 生成方法签名和方法覆盖率的关联关系存入 DB 中

获取推荐结果是推荐测试用例的核心需求，其用例描述如表3.5所示。在分析完用户的测试覆盖报告后，该功能会查询历史提交记录数据，动态地确定需要进行推荐的方法。该过程借鉴滑动窗口的思想，选取固定窗口大小的提交记录进行分析。最后查询 ElasticSearch 存储语料库中合适的代码片段进行推荐。如未发现可推荐的测试用例片段则返回异常原因信息。

表 3.5: 获取推荐结果用例描述

ID	**UC4**
名称	获取推荐结果
参与者	参与者：平台使用者和测试学习用户 目标：得到以方法签名为分组的推荐测试用例集合
触发条件	用户在 WebIDE 平台上运行了单元测试
前置条件	覆盖报告分析结束
后置条件	无
优先级	高
正常流程	1. 查询用户的推荐设置，选择历史提交记录作为“分析窗口” 2. 筛选覆盖率在“分析窗口”中处于“待推荐状态”的方法 3. 根据方法签名检索 ES 测试用例语料库得到待推荐测试用例集合 4. 对待推荐测试用例集合进行打分、排序和筛选。选择 TopK 进行推荐
扩展流程	1a. 如果查询待推荐测试用例为空，给出具体原因并返回

用户在进行白盒测试学习过程中，由于学习能力以及已有基础的不同，系统需要针对不同的用户提供不同的推荐配置，因此提供给用户“推荐设置”功能，其用例描述如表3.6所示。可设置的参数包括推荐频率、推荐结果数。

表 3.6: 推荐设置用例描述

ID	**UC5**
名称	推荐设置
参与者	参与者：平台使用者和测试学习用户 目标：用户在进行开发者测试学习的过程中调整自己的推荐配置
触发条件	用户点击” 推荐设置”
前置条件	用户已打开 WebIDE 系统，并看到做题界面
后置条件	无
优先级	中
正常流程	1. 用户填写“推荐频率”和“推荐结果数”可选参数 2. 用户点击“确认”，保存设置 3. 系统提示保存成功

测试程序切片、项目模板构建、构建测试用例语料库均为离线数据处理过程。其中测试程序切片为离线数据处理的第一个步骤，该步骤使用切片工具处理 慕测 平台已有用户提交的测试脚本，得到一个针对该项目的测试用例片段集合。其用例描述如表3.7所示。

表 3.7: 测试脚本切片用例描述

ID	**UC6**
名称	测试脚本切片
参与者	参与者：后台管理人员 目标：已有用户提交的测试脚本使用切片工具进行处理，得到一个针对该项目的测试用例片段集合
触发条件	系统管理员运行离线处理任务
前置条件	1. 系统提供用户已提交的测试项目数据，其中包含了测试脚本以及测试脚本的运行上下文环境 2. 用户已提交测试项目可正确运行 maven 命令 3. 测试脚本均包含优质代码片段，优质代码片段指覆盖率高，杀死 bug 数多的代码片段。
后置条件	无
优先级	高
正常流程	1. 解压缩学生提交的测试项目数据 2. 运行 shell 命令生成该项目的字节码文件 3. 分析测试类字节码文件提取方法名作为切片结束点，分析 assert 断言语句作为切片切入点 4. 进行切片将生成的测试片段数据存入待处理目录中，等待下一步骤处理。

项目模板构建为离线数据处理模块的第二个步骤，该步骤的目的是将上一个步骤测试程序切片中产生的代码片段分别融入到给定好的项目模板中，封装成一个独立可运行单元测试的项目。其用例描述如表3.8所示。

表 3.8: 项目模板构建用例描述

ID	**UC7**
名称	项目模板构建
参与者	参与者：后台管理人员 目标：代码片段分别融入到给定好的项目样例中，封装成一个独立可运行单元测试的项目
触发条件	系统管理员运行离线处理任务
前置条件	程序切片步骤中产生了测试代码片段，并存入一个给定的目录中
后置条件	无
优先级	高
正常流程	1. 检查项目模板的正确性，是否可运行 2. 替换 pom 文件，嵌入覆盖率分析组件工具 3. 将代码片段嵌入测试模板并放入项目中 4. 验证项目是否可正常运行

构建测试用例语料库为离线数据处理模块的第三个步骤，该步骤对上一步骤中融合好的代码片段项目模板运行覆盖率分析，得到一份覆盖率分析报告。通过解析覆盖率分析报告得到被测代码和测试代码片段之间的关联关系，将该结果作为一份测试语料持久化到 ElasticSearch 中。该过程如发生异常、无法产生测试代码语料的数据，系统需要抛出易理解的错误信息，来帮助开发人员定位问题，其用例描述如表3.9所示。

表 3.9: 构建测试用例语料库用例描述

ID	**UC8**
名称	构建测试用例语料库
参与者	参与者：后台管理人员 目标：构建一个用于测试代码推荐的测试用例语料库
触发条件	系统管理员运行离线处理任务
前置条件	无
后置条件	无
优先级	中
正常流程	1. 系统进行覆盖率分析，得到覆盖分析报告 2. 读取分析报告，解析代码片段所覆盖的方法、分支、行的覆盖信息 3. 将各分支和行覆盖信息转化为方法级别的 01 向量 4. 关联代码片段，构建测试语料，存储到 ElasticSearch 中
扩展流程	2a. 读取覆盖率分析报告失败，终止存储并抛出异常，返回出错信息

3.3　系统总体设计

3.3.1　系统整体架构设计

本系统是一个面向测试教学场景的辅助学习系统，结合了测试覆盖分析技术和程序切片技术，其架构如图 3.2所示，系统包括 WebIDE 后台业务逻辑模块、代码推荐模块和离线数据处理模块，其中离线数据处理模块又分为三个子模块：测试脚本切片子模块、切片融入模板子模块和覆盖率分析子模块。前端部分是 慕测 已有的在线集成开发环境 WebIDE，用户可通过在浏览器端编写和运行代码触发动态代码推荐，持久化部分使用了 Elasticsearch 集群索引推荐语料代码片段。从工程设计角度描述如下:

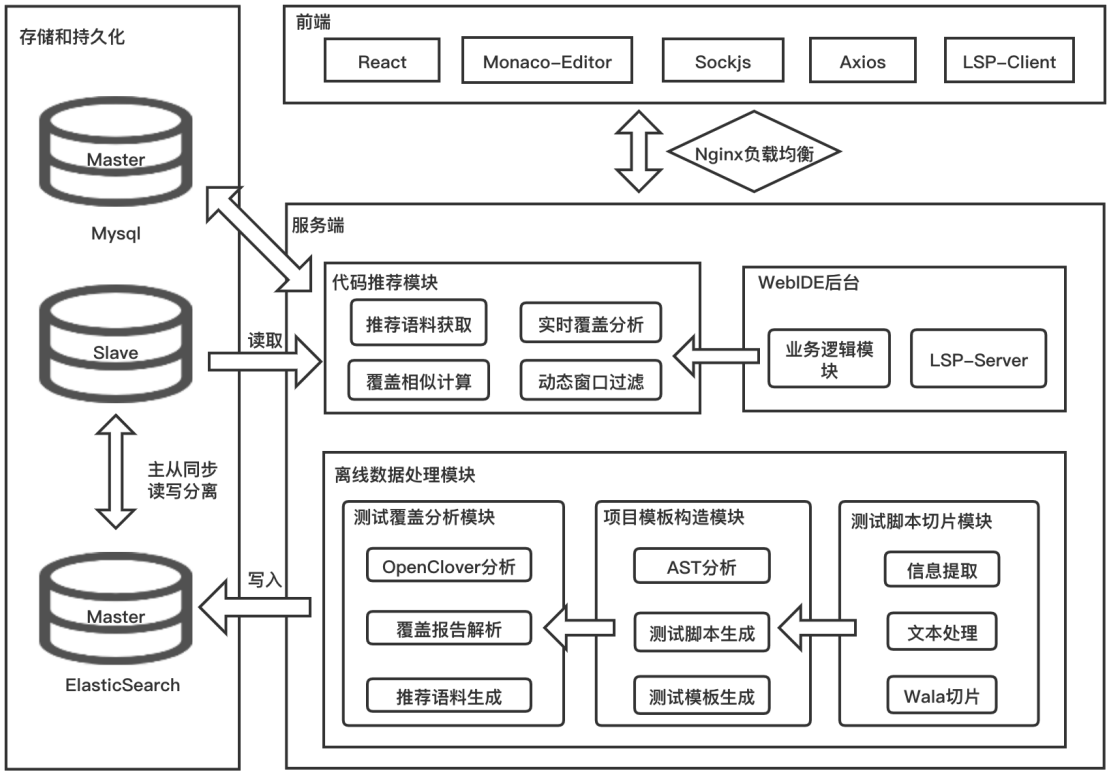

图 3.2: 系统整体架构图

系统前端使用业界比较流行的 React 技术进行开发，它一个组件化的、声明式的前端框架，其提供的丰富组件可帮助渲染复杂前端页面。系统的 UI 界面是基于 慕测 WebIDE 进行的二次开发，通过二次开发的方式避免 UI 界面开发带来的巨大工作量。同时作为一个在线编程学习系统，前端需提供一个丰富的智能化编辑器，系统采用目前支持多种语言编程提示的 Monaco-Editor 编辑器，通过支持 LSP 协议连接到语言服务器上可实现针对语言的智能提示。在和后端进

行通信方面，系统的大部分通信交给封装 Http 请求的 Axios 组件进行处理。对于异步高延时通信，系统采用 WebSocket 通信方式并使用 Sockjs 进行异步通信，避免长轮询请求带来的性能损耗。

系统服务端各个服务均使用基于 Java 语言的 SpringBoot 框架开发，SpringBoot 框架因其社区支持强大以及学习曲线低的特点可帮助快速构建系统。除了前端和服务端需要通过 Http 的方式通信以外，服务端之间的通信使用了 SpringBoot 提供的 RestTemplate 组件，其提供简单易用的 API 可帮助系统快速搭建通信桥梁。由于系统处理例如程序切片，覆盖率分析等任务时，需要调用一些第三方脚本工具进行处理，因此服务端将工具进行封装并使用 Docker 作为运行容器，避免由于环境不同导致的部署问题，并且统一工作环境可方便运维人员的维护工作。

此外，系统为降低用户触发推荐后的等待时间，使用 ElasticSearch 构建持久化搜索引擎，降低每次搜索所用时间，并采用读写分离主从同步的架构模式，大大提高系统的吞吐量。为提高系统的水平扩展性，采用 Nginx 进行动静分离和负载均衡。随着用户人数的增加，系统可动态地增加后端服务数量，实现可伸缩架构。

3.3.2　系统模块划分

如图3.2所示，系统服务端分为了三个模块。WebIDE 后台业务模块，代码推荐模块和离线数据处理模块，其中离线数据处理模块又可细分为测试脚本切片模块，切片融入模板模块，覆盖率分析模块三部分。下文会对离线数据处理模块进行详细的设计介绍。

WebIDE 后台业务模块承接 WebIDE 前端，其提供前后端异步通信功能以及一些业务逻辑功能，如提供通用的基础代码提示功能，触发测试用例推荐，用户信息持久化等。该模块因为只涉及到一些简单的增删查改和库调用处理，不是本系统的重点内容，将不会在本文中进行详细介绍。

离线数据处理模块整体功能是构建一个测试代码片段和测试覆盖关联关系的测试片段语料库，其整体处理过程分成三个子模块：测试脚本切片模块负责测试脚本的信息提取、文本处理和程序切片，该模块用到程序分析和切片技术，通过使用 Wala 程序切片处理工具实现具体功能；项目模板构建模块负责测试模板构建和测试代码片段融入，该模块使用了 AST 技术提取抽象语法树并进行测试模板生成和代码片段的融入工作；覆盖分析模块负责覆盖报告生成、解析以及结推荐语料的生成和持久化，该模块使用 OpenClover 工具产生覆盖分析报告，并使用 SpringBoot-JPA 持久化测试语料，存储到 ElasticSearch 搜索引擎中。

代码推荐模块会在用户进行测试学习时发挥作用，用户每次运行都会触发编程行为分析，该过程同样使用 OpenClover 提取覆盖分析报告。通过窗口过滤筛选待推荐被测方法签名，之后将签名作为查询条件，搜索 ElasticSearch 中存储的推荐语料，最后进行相似度计算，筛选出最佳匹配代码片段返回给用户。该模块包含待推荐方法筛选和推荐测试脚本筛选等功能。

3.3.3　4+1 视图

Philippe Kruchten 在 1995 年的论文《The 4+1 View Model of Architecture》中首次提出 4+1 视图的概念 [41]。如图 3.3所示，4+1 视图指的是由逻辑视图、开发视图、进程视图、物理视图和用户场景共同构成的软件架构视图。通常架构包含的内容过多过杂，直接整体描述会让系统变得复杂难懂，因此采用这种 4+1 视图的形式，从不同的视角描述整体架构，只描述当前视图关心的问题，则可以做到“分而治之”，使系统更易理解。在上一节的需求分析中已经从场景视图的角度描述了系统的所有使用场景。

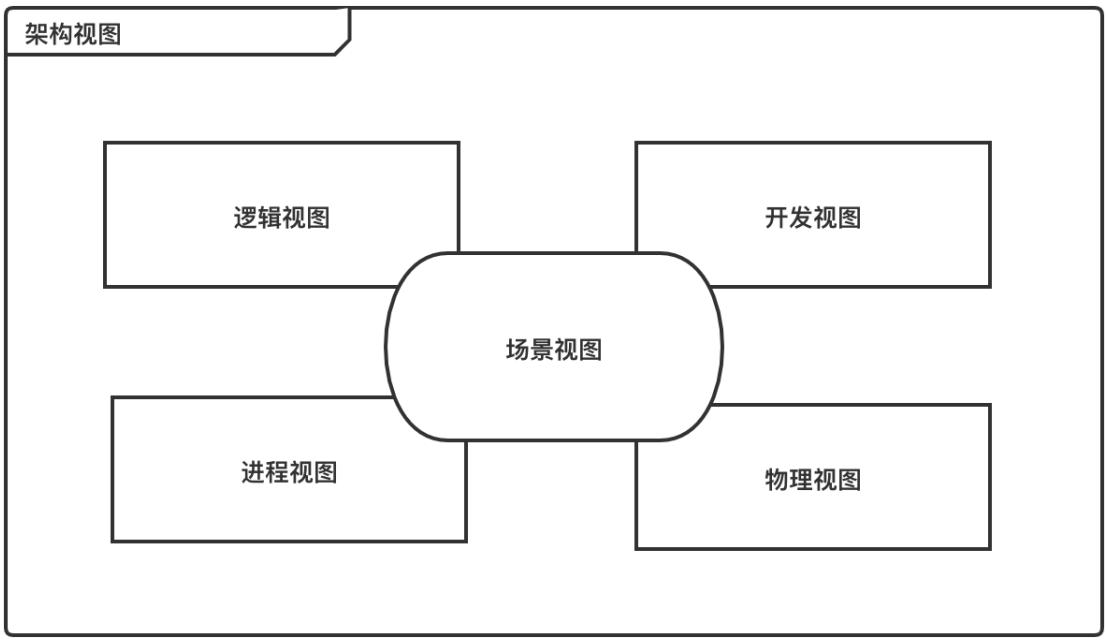

图 3.3: 系统架构视图

逻辑视图是从业务的角度描述系统，本系统采用面向对象的设计方法，逻辑视图也可表述为对象模型，因此可以使用 UML 语言进行描述。通过分析用户的功能性需求，转化得到业务逻辑划分如下图 3.4所示。

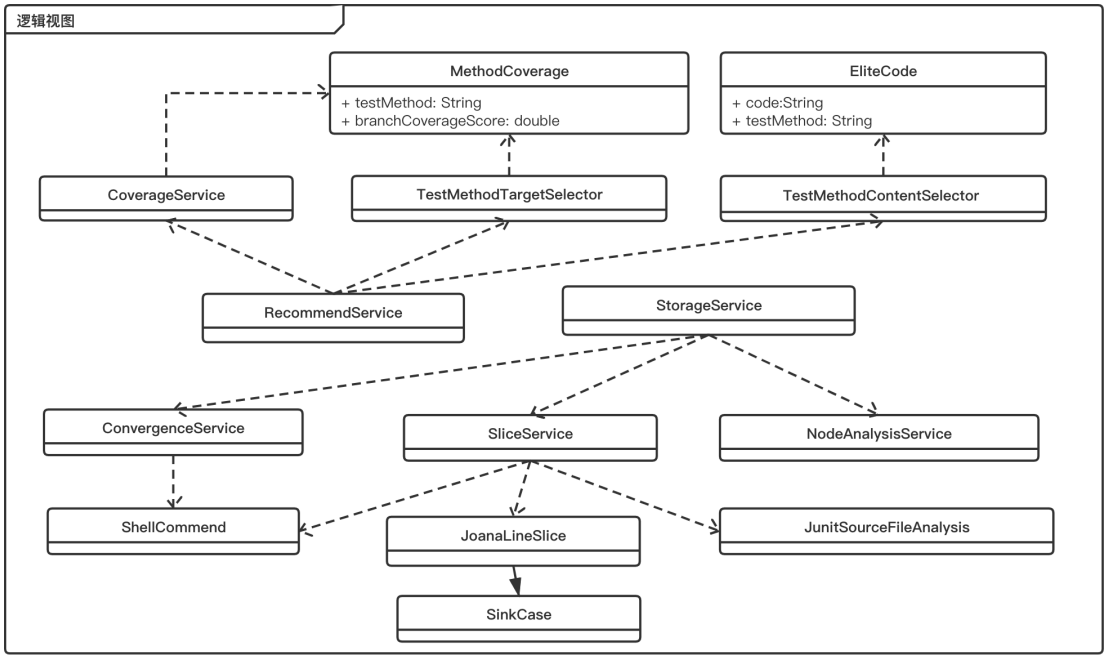

图 3.4: 系统逻辑视图

RecommendService 负责处理用户的推荐请求。CoverageService 负责计算用户提交的测试代码的覆盖率分数并建立映射关系。TestMethodTargetSelector 负责筛选出需要进行推荐的方法签名。TestMethodContentSelector 提供筛选可进行推荐的代码片段的功能。StorageService 提供整合测试语料并进行持久化的功能。SliceService 提供将测试脚本切片得到测试代码片段的功能。ConvergenceService 负责将测试代码片段融合测试模板的功能。NodeAnalysisService 负责分析运行代码片段以及分析覆盖率报告的功能。MethodCoverage，ELiteCode，SinkCase 为系统抽象实体，是系统核心业务逻辑用到的数据结构。

开发视图从开发者的角度来描述系统架构，即系统在开发者视角的静态组织结构，如图 3.5所示。其主要分为三层，顶层是展示给用户的 UI，包含了 HTML、CSS、JavaScript 以及构成的交互组件。

服务层 ServerLogic 采用模块化和 MVC 架构设计的思想进行划分，Controller 负责接收并处理所有 Web 请求，Service 层负责提供 Controller 层需要的推荐业务逻辑和离线数据处理模块的业务逻辑。其中 RecommendationService 提供代码推荐相关的方法，该 Service 会调用 Selector 选择器进行用户数据的处理分析和待推荐代码片段的筛选工作。StorageService 负责构建测试代码片段语料库，该 Service 会调用 DataProcess 数据处理包进行测试脚本切片、代码片段融合和覆盖率分析。Entity 包定义了本系统核心的实体类。Common 和 Util 则提供

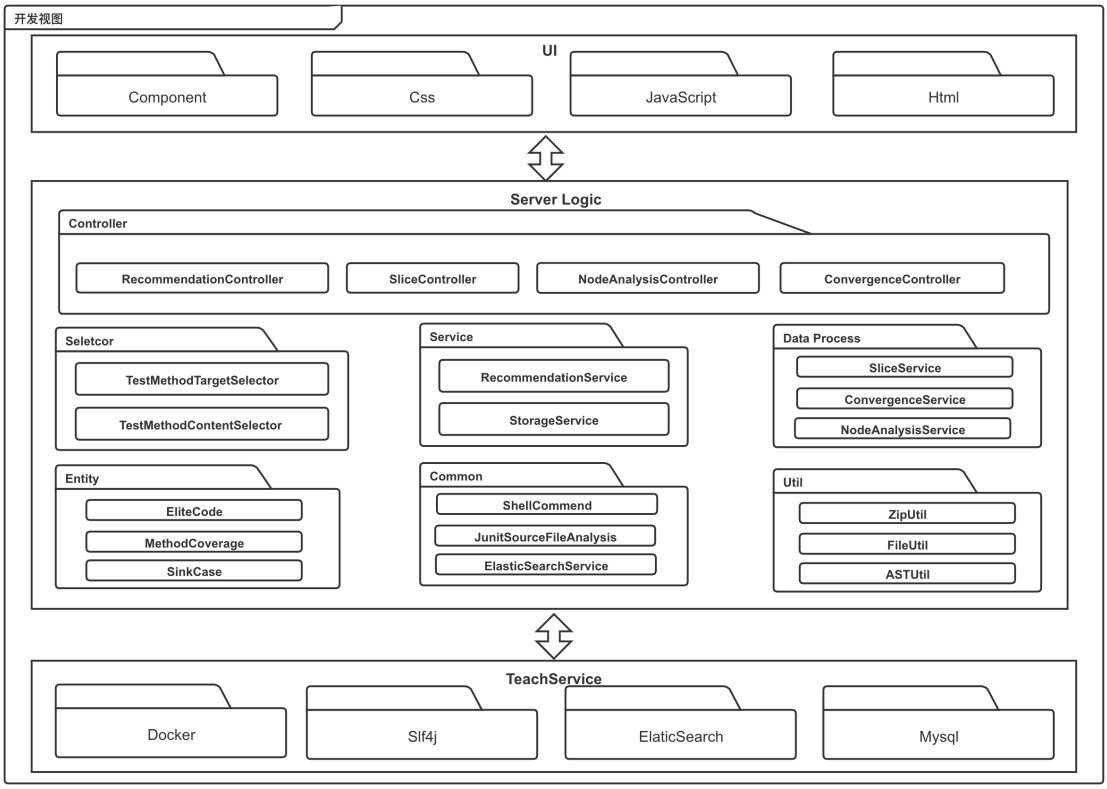

图 3.5: 系统开发视图

Service 层用到的所有通用方法和类。

TeachService 为开发本系统会用到的一些第三方组件和库，第三方库提供大量功能强大的通用方法，帮助开发人员从繁琐的重复逻辑中抽离出来，从而让开发者专注业务逻辑。本系统用到多种第三方组件和库，测试代码运行用到了 Docker 容器；记录运行时日志用到了 Slf4j；数据持久化使用 ElasticSearch 和 Mysql；针对 Json 的序列化和反序列化使用 Gson 库。

进程视图从系统集成人员的角度描述系统架构，可直观描述系统在集成时相关进程和线程之间的通信方式。该视图可有效的帮助开发人员理解程序的内部运行逻辑，方便排查异常和错误。如图3.6所示，当系统需要执行一次推荐请求时，首先会启动一个异步线程，返回给主进程 “推荐正在处理中” 的结果信息。异步线程负责调用 Docker 容器运行和处理推荐。在 Docker 运行结束后异步线程调用推荐服务，分析用户的覆盖率报告将结果存入 Mysql 中。然后调用获取推荐脚本的功能，检索过滤 ElasticSearch 上存储的测试代码片段，筛选得到最终推荐数据并返回给异步线程。异步线程对于所有推荐结果均通过 WebSocket 管道返回给主进程显示。

对于离线处理模块，主进程触发测试语料生成，会调用离线数据处理进程进

行处理。离线数据处理进程首先进行切片和构造测试覆盖运行模板，之后通过运行和覆盖分析得到 Json 格式的测试覆盖数据，经过转化后得到测试语料。离线数据处理进程在得到测试语料后将结果存入 ElasticSearch 中，并返回“处理成功”状态信息。存入 ElasticSearch 中的测试语料会在用户触发推荐请求时被使用到。

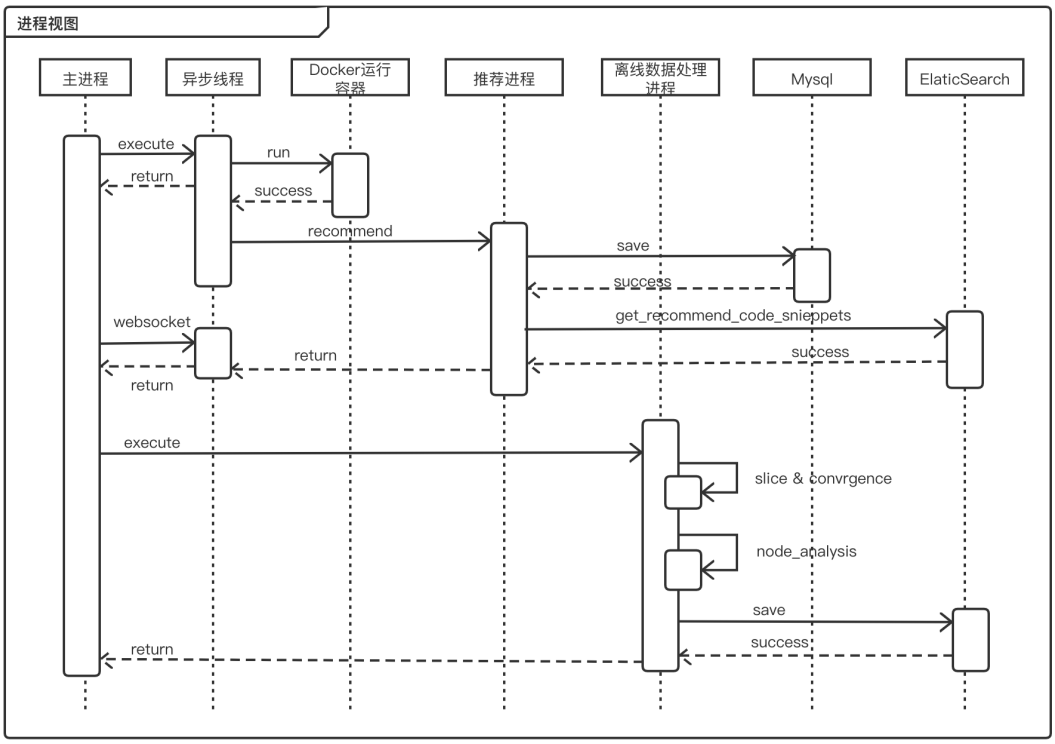

图 3.6: 系统进程视图

物理视图则站在运维人员的视角，描述软件如何和硬件进行一一对应。该视图客观反映真实部署场景下系统运行模式，方便运维人员和开发人员理解系统实际运行情况。如图3.7所示。本系统所有服务均部署在 Docker 容器中，保证系统一次打包任意部署和扩展。用户通过浏览器即可访问测试代码推荐系统，首先会通过防火墙的拦截和过滤，之后服务器将请求发送到 Nginx 的监听进程。Nginx 经过负载均衡后将请求转发给系统所依赖的系统 WebIDE 智能编程系统。对于任意代码推荐相关的请求会经由 WebIDE 发送给 WebIDE-Recommend 子系统进行处理。后台系统所接收到的所有持久化请求和数据访问请求会通过 TCP 的通信方式访问 Mysql 容器以及 ElasticSearch 容器进行处理。

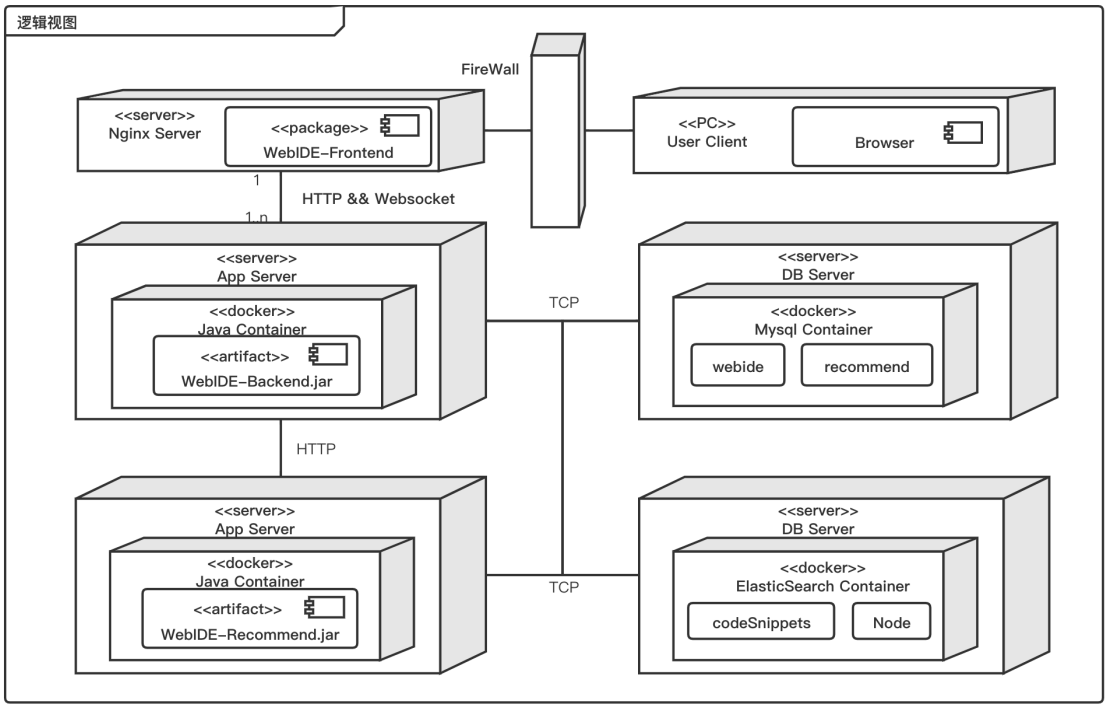

图 3.7: 系统物理视图

3.3.4　离线数据处理模块设计

离线数据处理模块旨在构建一个测试代码片段的语料库，持久化到搜索引擎 ElasticSearch 中，其总体架构设计如图3.8所示，主要步骤如下：

（1）由开发人员触发离线数据处理请求，后台 Controller 开始处理。

（2）Controller 调用外部服务从 慕测 平台获取待处理的、完整可运行的测试代码项目压缩包。

（3）获取成功后，等待下一步处理工作；获取失败则返回出错信息。

（4）封装异步处理任务交给线程池进行异步处理。

（5）通知开发人员离线数据已成功开始。

（6）对测试代码项目包进行切片处理，得到测试代码片段集合，存入本地文件夹中备用。

（7）清理测试代码项目包，得到不含任何测试代码片段的空项目包。

（8）将每一个测试代码片段融入空项目包中，得到待运行测试脚本项目包。

（9）运行项目包进行覆盖率分析处理，得到测试覆盖分析报告，之后和代码片段建立关联关系，得到测试代码片段语料。

（10）将测试代码片段语料存入 ElasticSearch 中。

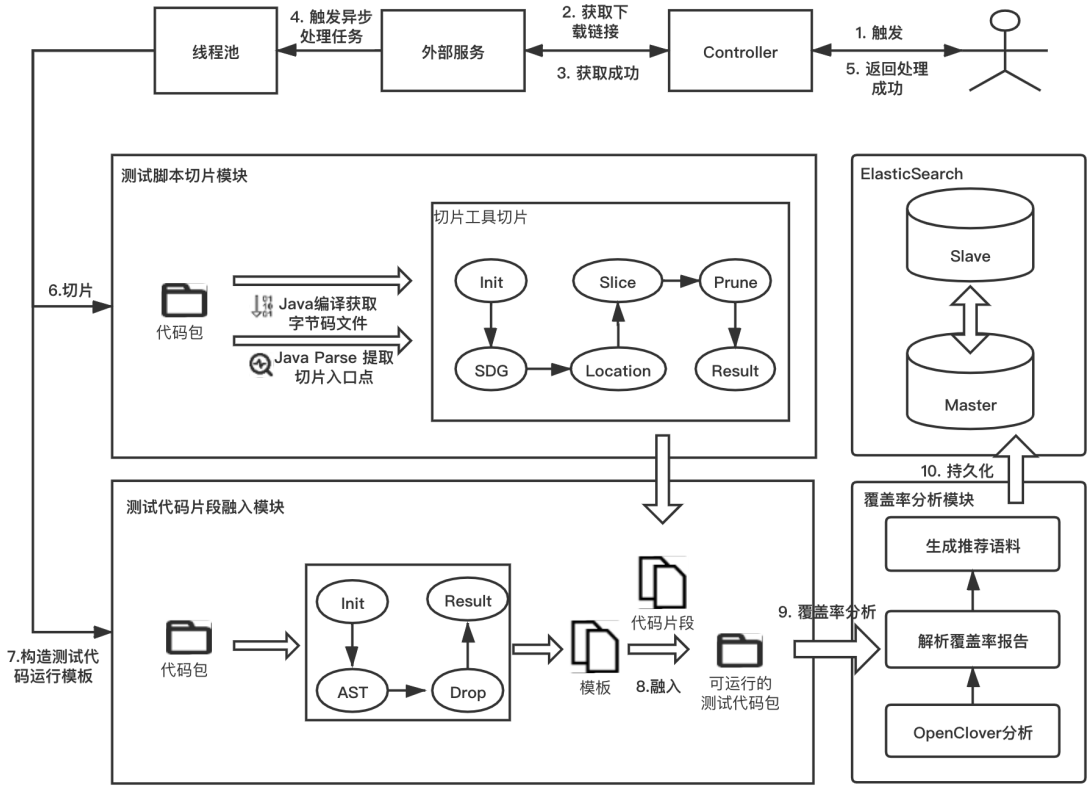

图 3.8: 离线数据处理模块总体架构

到此，离线数据处理模块的主要步骤结束，该模块由于内容较多，其中的测试脚本切片子模块、代码片段融入模块、覆盖率分析模块等将在后续几个小节中进行分析和设计。

3.4　持久化模型设计

3.4.1　测试代码片段语料

上文说明的离线数据处理模块，构建了一个包含大量测试代码片段的测试代码片段语料库。测试代码片段为本文所实现系统的核心数据结构，为能够明确系统的各部分设计目的，在本节中对系统中测试代码片段的定义进行详细的说明和解释。

代码片段常用来描述一段可以实现一个功能或者表达某种特定含义的由几行代码组成的代码块。在本文中测试代码片段是一种特殊的代码片段，其结构如图 3.9 所示，它是单元测试用例的部分代码，仅包含一个 assert 断言语句和与其产生调用关系所有语句的集合。该关系通常表现为系统调用关系图（SDG）中的一条可达边。例子中给定的 assert 语句用到了变量 alu, 因此 alu 变量所在行和

assert 语句共同构成一个测试代码片段。将得到的测试代码语句聚合按顺序融入到一个给定好的测试脚本文件中，可通过覆盖率工具得到当前代码片段所测内容的测试覆盖率。本文称这样的测试代码语句集合为测试代码片段。代码片段和对应的测试覆盖率共同构成用于推荐的测试代码片段语料。

```
package net.mooctest;
//省略 import 代码
public class ALUTest {
    @Test
    public void test() {
        //----代码片段开始位置----
        ALU alu = new ALU();
        assertEquals("01000000010010010000111111011010",
                alu.ieee754("3.1415926", 32));
        //----代码片段结束位置----
    }
}
```

图 3.9: 测试代码片段示例

3.4.2　测试代码语料库模型设计

测试代码片段语料库存储结构如下表 3.10所示，该结构为离线处理阶段产生的测试代码片段原料库。包含了切片阶段用于唯一标识一个测试代码片段的 uuid、测试代码片段信息 codeSnippet、文件名 fileName、测试代码片段所属方法名 testFunctionName、该测试代码片段语句覆盖情况 nodeCatchJsonString、该测试代码片段完整的覆盖报告 metaNodeJsonString、完整的独立可运行测试脚本文件 codeSnippetJavaFile。提取该数据结构的目的是为了系统的可扩展性，当涉及测试代码片段新的需求来临时，可以使用该测试代码片段原料库进行二次处理提取更多有价值的信息。

针对本次用于测试代码片段的需求，由表 3.10进行二次处理可以得到用于推荐的最终语料库数据如表 3.11所示。codeSnippet 为测试代码片段的文本存储结构。testFunctionName 为该测试代码片段覆盖到的源方法列表，该字段建立了全文索引，会作为检索条件，在动态推荐时使用。functionCatchVector、branchCatchVector、statementCatchVector 是根据原料库中的 metaNodeJsonString

表 3.10: 测试代码片段原料库

字段名	数据类型	描述
uuid	text	唯一标识符，唯一标识一个测试语料
caseId	long	测试题目 Id
fileName	text	测试语料所属测试文件
testFunctionName	text	测试语料覆盖到的原方法列表
codeSnippet	text	测试语料代码片段
nodeCatchJsonString	text	测试代码片段完整的覆盖信息，json 格式，true 表示覆盖
metaNodeJsonString	text	测试代码片段完整的覆盖报告，标注了被测项目所有的测试方法、分支、语句
codeSnippetJavaFile	text	可运行的完整测试脚本

和 nodeCatchJsonString 计算得到的测试覆盖向量。例如在 branchCatchVector 可以表示为 0001,1000,0101 的以逗号分隔的 01 向量。其中两个逗号中间的内容表示被测项目的一个测试方法，1 表示该方法的一个分支被成功覆盖。这三个测试覆盖向量会在动态推荐阶段用于计算测试代码片段的动态分数，系统会根据分数的高低进行推荐。

表 3.11: 测试代码片段语料库

字段名	数据类型	描述
uuid	text	唯一标识符，唯一标识一个测试语料
caseId	long	测试题目 Id
fileName	text	测试语料所属测试文件
testFunctionName	text	测试语料覆盖到的原方法列表
codeSnippet	text	测试语料代码片段
functionCatchList	text	测试代码片段覆盖的方法列表
functionCatchVector	text	01 向量，总长度为该项目所有被测方法数量，1 表示成功覆盖
branchCatchVector	text	测试代码片段分支覆盖情况，以逗号分隔，两个逗号间为一个方法的全部分支，1 表示分支成功被覆盖
statementCatchVector	text	测试代码片段语句覆盖情况，以逗号分隔，两个逗号间为一个方法的全部语句，1 表示分支成功被覆盖

3.4.3　动态推荐用户覆盖模型设计

在动态推荐过程中，系统需要实时分析用户运行单元测试产生的覆盖信息，存储结构如下表所示3.12所示，主要包括了考试 Id、题目 Id、用户 Id，三者唯一确定用户一次测试覆盖。testMethod 被测方法签名表示每次提交所覆盖到的被测方法。coverTime 表示本次覆盖发生的时间。branchCover,statementCover 表示每个方法的分支覆盖率和语句覆盖率。branchCoverageVector 表示该方法的分支覆

盖向量，使用 01 向量形式表示。statementCoverageVector 表示该方法语句覆盖向量，使用 01 向量形式表示。

表 3.12: 用户覆盖信息表

字段名	数据类型	描述
id	BIGINT	唯一标识符，唯一标识一条用户覆盖信息
examId	BIGINT	考试，练习的 Id
caseId	BIGINT	题目 Id
userId	BIGINT	用户 Id
testMethod	VARCHAR(255)	用户测试测到的被测方法签名
coverTime	BIGINT	覆盖发生时间
branchCover	DOUBLE	被测方法分支覆盖率
statementCover	DOUBLE	被测方法语句覆盖率
branchCoverageVector	VARCHAR(255)	01 向量，被测方法所有分支覆盖情况，1 表示成功覆盖
statementCoverageVector	VARCHAR(255)	01 向量，被测方法所有语句覆盖情况，1 表示成功覆盖

3.5　本章小结

本章描述测试用例推荐系统的整体概要设计，首先在明确系统目标前提下，进行系统的功能性需求分析和非功能性需求分析。之后根据需求进行用户场景分析，以及对用例进行详细描述并通过用例描述表的形式展示。之后通过分析用例，本文对系统的整体框架进行设计并进行具体技术介绍和分模块介绍。为更清楚的描述设计的细节，本文从 4+1 视图的角度对系统进行不同视角介绍。之后对离线数据处理模块进行流程设计介绍。最后分别对测试代码语料库模型和动态推荐用户覆盖模型进行阐述。

第四章　系统详细设计与实现

4.1　测试代码切片模块详细设计与实现

4.1.1　流程设计

测试代码切片子模块是构建测试代码语料库的第一个步骤。 慕测 平台每年通过比赛和考试收集了大量学生提交数据，其中开发者测试的题目有很多和白盒测试相关，它们可帮助学生从测试角度理解被测内容以及学习如何进行测试。提交的数据结构为项目包的格式，项目包中包括完整项目结构和测试脚本文件。本子模块的目的为分离源码中的测试脚本文件，得到一个测试代码片段集合。测试代码片段的详细定义在第三章进行明确的阐述。

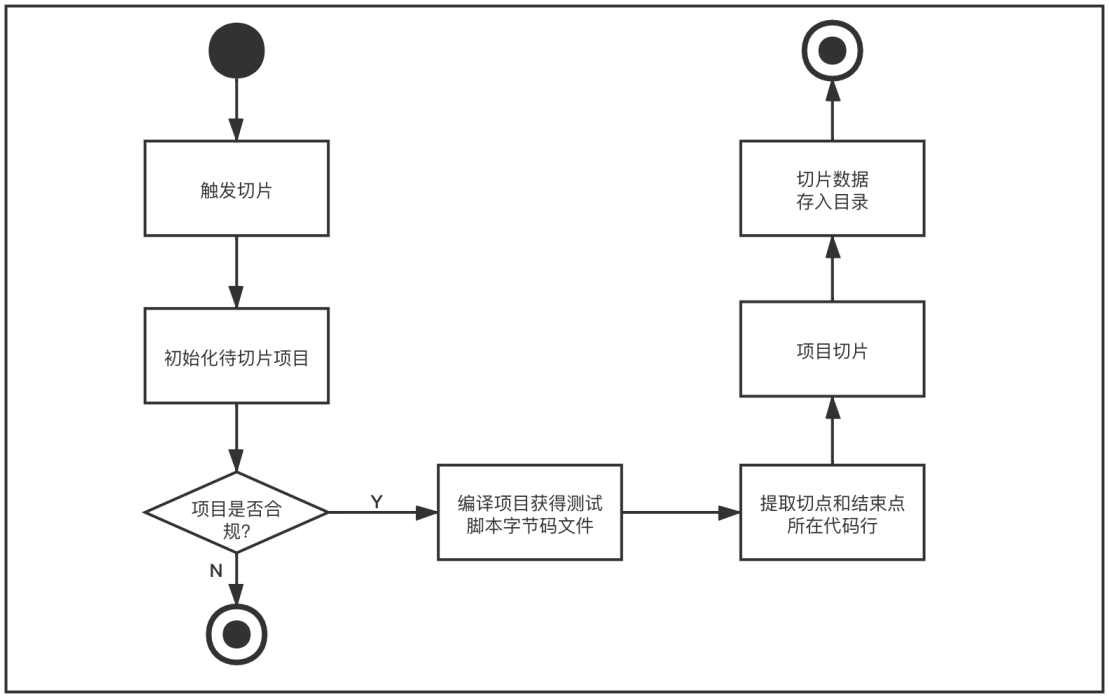

图 4.1: 测试代码切片流程图

测试代码切片子模块的流程如图 4.1所示，大致可分为三部分：切片前的数据处理、切片处理、切片后处理。在切片前处理阶段，首先进行对项目是否合规进行校验。由于切片工具分析需要测试脚本的字节码格式数据，所以系统首先需要编译目标代码以获取字节码文件。其次系统需要提取切片所用到的切点和

结束点所在代码行。在切片处理阶段，根据上一步得到的信息进行后向迭代式切片。在切片后处理阶段，保存每一个获得的程序切片文件到指定的文件夹中。

4.1.2　类图设计

测试代码切片子模块的类图如图 4.2所示, 图中包含测试代码切片过程中用到的核心类。为清晰的展示核心业务逻辑, 图中省略诸如 Controller 控制类, Wrapper 转化类等信息。SliceService 是切片核心类对外提供 slice 切片接口。SliceServiceImpl 为接口 SliceService 的实现类，调用 beforeSlice、computeSlice、afterSlice 三个方法。SinkCase 类用于抽象切片用到入口点信息，包含入口点类名、文件名和方法签名等属性。

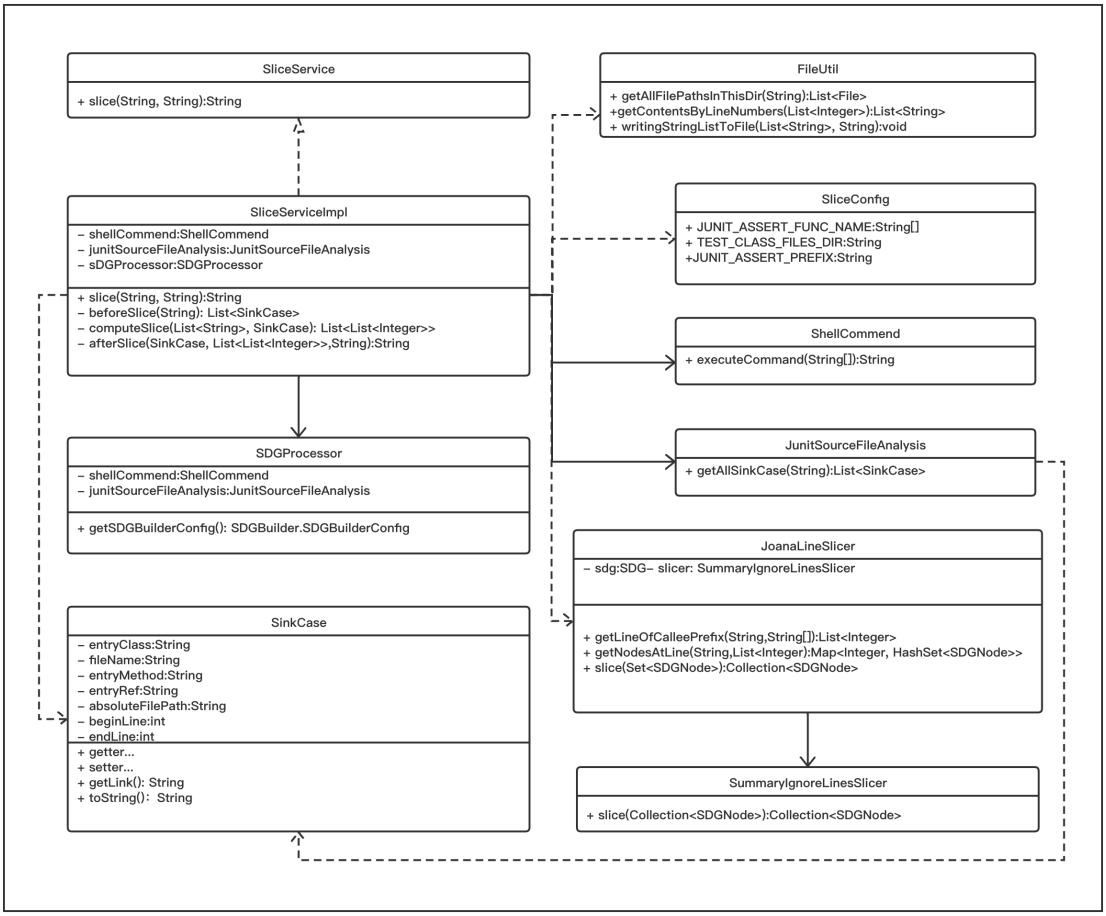

图 4.2: 测试代码切片类图

SliceServiceImpl 直接依赖 ShellCommend、JunitSourceFileAnalysis、SDGProcessor 和 JoanaLineSlice 类。ShellCommend 用于执行 shell 脚本，在本系统中用于执行 mvn test 命令得到测试脚本的字节码文件。JunitSourceFileAnalysis 类用

于分析测试代码脚本并把获取的方法信息封装成 SinkCase 对象。SDGProcessor 用于构建系统依赖图，JoanaLineSlice 用于调用 Wala 切片工具进行切片。FileUtil 用于处理切片工具得到的结果数据以及存储到文件中。

4.1.3　程序切片顺序图

测试代码切片子模块的切片过程的顺序图如图 4.3所示，系统首先触发程序切片调用 SliceServiceImpl 的 slice 方法。slice 方法通过完成三次自调用处理待切片项目包得到测试代码片段集合。在 beforeSlice 处理中，首先需要运行 mvn test 命令得到在切片过程中需要的测试脚本的字节码文件，其次需要使用 Java Paeser 工具分析测试脚本的抽象语法树得到所有待切片测试方法的 SinkCase，SinkCase 的内容在类图章节进行过详细介绍，该对象包含切片入口点和结束点等信息。

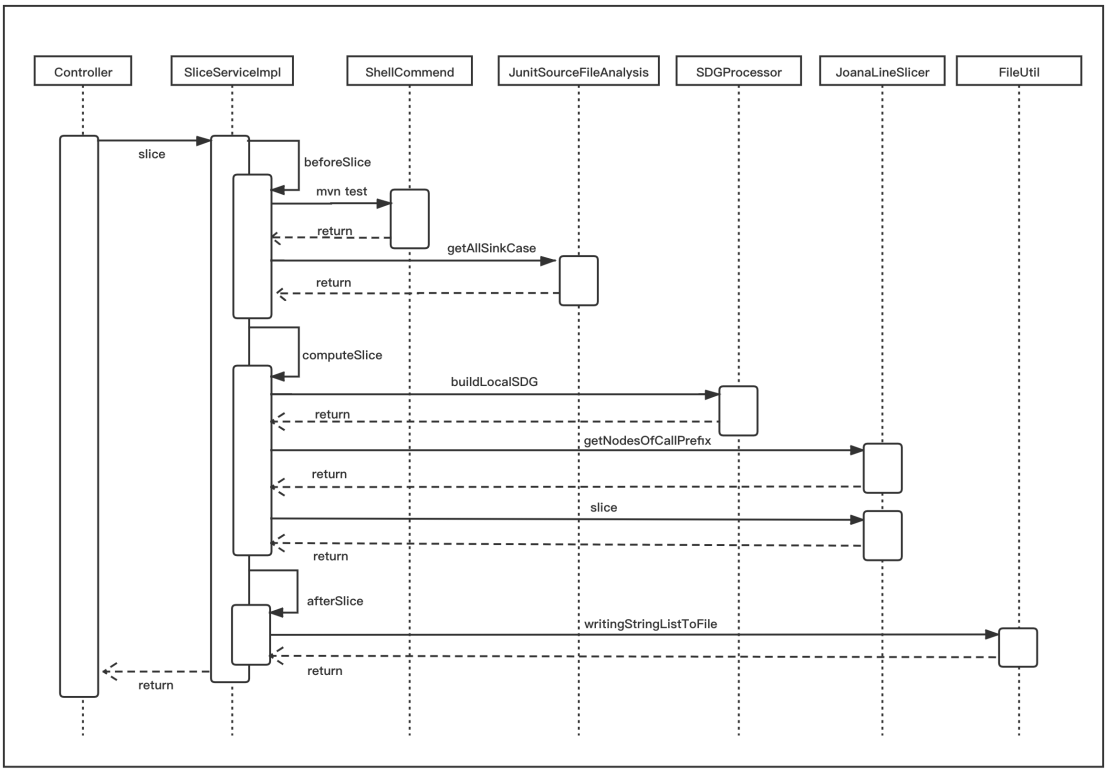

图 4.3: 测试代码切片时序图

上一步获得的所有 SinkCase 后，就可调用 computeSlice 方法进行程序切片。切片过程首先构建一个当前测试脚本的系统依赖图，之后将 SinkCase 转化为系统依赖图中的 Node 节点信息，最后遍历 Node 节点信息进行切片，得到测试片段在文件中的行号集合。每一个 SinkCase 都会得到一个行号集合，通过该集合可唯一对应到测试代码片段中代码行。

afterSlice 的处理过程为遍历每一个行号集合，将行号集合对应测试脚本中的代码片段提取出来并保存到文件中。到此，整个切片过程结束，输出为一个包含当前测试项目生成的所有测试代码片段集合，存储在磁盘的一个目录中。

4.1.4　程序切片关键代码

测试代码切片子模块切片过程的关键代码如图 4.4所示，涉及程序切片前、切片和切片后处理的业务逻辑。通过切片前处理得到所有切点 SinkCase 集合。

```
public String slice(String projectPath,String outputDir) {
        if(projectPath == null) { //校验
            return "";
        }
        String result = "";
        //切片前置处理，得到所有 SinkCase
        List<SinkCase> sinkCases = beforeSlice(projectPath);
        List<String> classFiles = FileUtil.getAllFilePathsInThisDir(projectPath +
SliceConfig.FILE_SEPARATOR + SliceConfig.TEST_CLASS_FILES_DIR);
        //遍历每一个 SinkCase，进行程序切片
        for (SinkCase sinkCase : sinkCases) {
            try {
                List<List<Integer>> codeSnippetsLines = computeSlice("",new
String[]{},
                        classFiles,null,sinkCase);
                //切片后处理，将结果保存到 outoutDir 中
                result = afterSlice(sinkCase, codeSnippetsLines, outputDir);
            } catch ( Exception e) {
                log.error("切片异常", e);
            }
        }
        //包含所有测试代码的
        return result;
}
```

图 4.4: 测试代码切片关键代码

切片过程为遍历每一个 SinkCase，进行切片获得测试代码切片的行号集合。afterSlice 负责分析测试脚本，提取对应行号的代码，保存到文件中。最后返回测试代码片段保存的目录路径。

4.2　测试模板构造模块详细设计与实现

4.2.1　流程设计

测试项目模板构造子模块是构建测试代码片段语料库的第二个步骤。该步骤的逻辑较为简单，流程如图4.5所示。在上一步得到测试代码片段后，系统会触发测试项目模板构造业务逻辑。该业务逻辑的第一步是使用切片用到的待切片项目包，替换项目包中的 pom.xml 文件替为包含 OpenClover Maven 插件的 pom.xml 文件。之后通过分析测试脚本删除原测试脚本中的所有测试方法得到测试模板。接着从文件中读取测试代码片段，将读到的测试代码片段融入到测试模板中并进行保存。

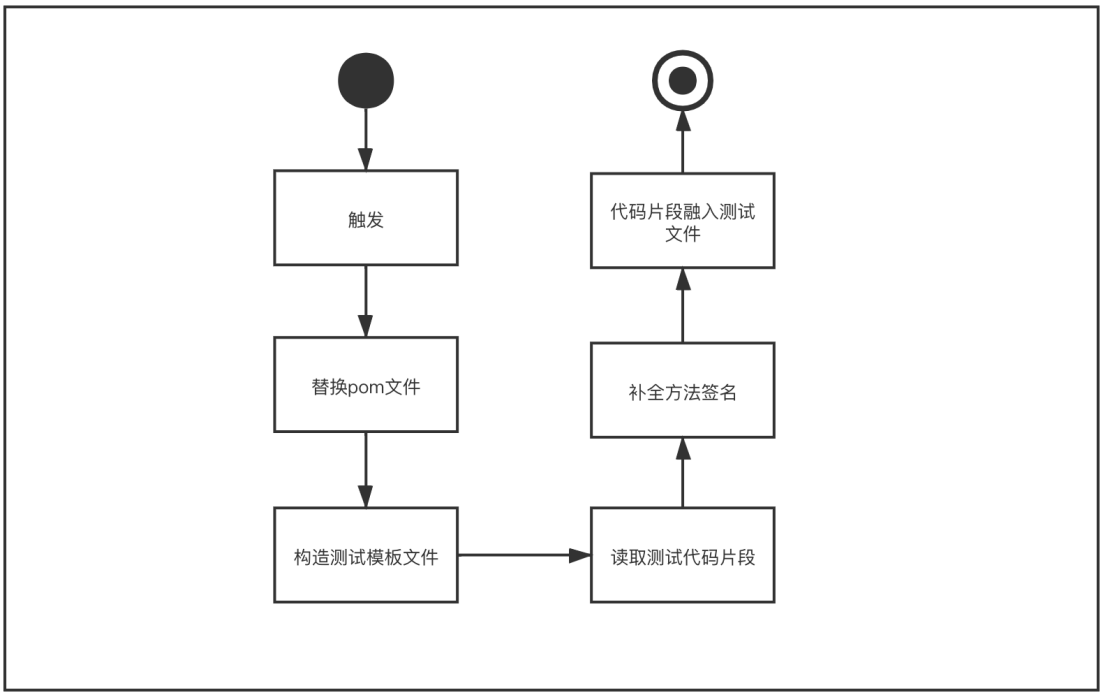

图 4.5: 测试项目模板构造流程图

4.2.2　类图设计

测试项目模板构造子模块的类图如图 4.6所示，ConvergenceService 暴露 converged 接口，它接受测试代码片段、项目路径、输出融合文件路径作为参数，输出

为最终融合好测试模板的文件路径。ConvergenceServiceImpl 为其实现类，提供替换 pom.xml 方法和删除测试模板已有测试用例两个内部方法。JunitSourceFileAnalysis 用于分析测试代码脚本，其内部的静态内部类 MethodVistorForCoverged 提供针对抽象语法树的遍历能力，使用 Java Parser 库提取每个方法的起始行和结束行封装成 MethodScope 对象。FileUtil 用于将融合好的测试模板保存到磁盘的文件夹中。

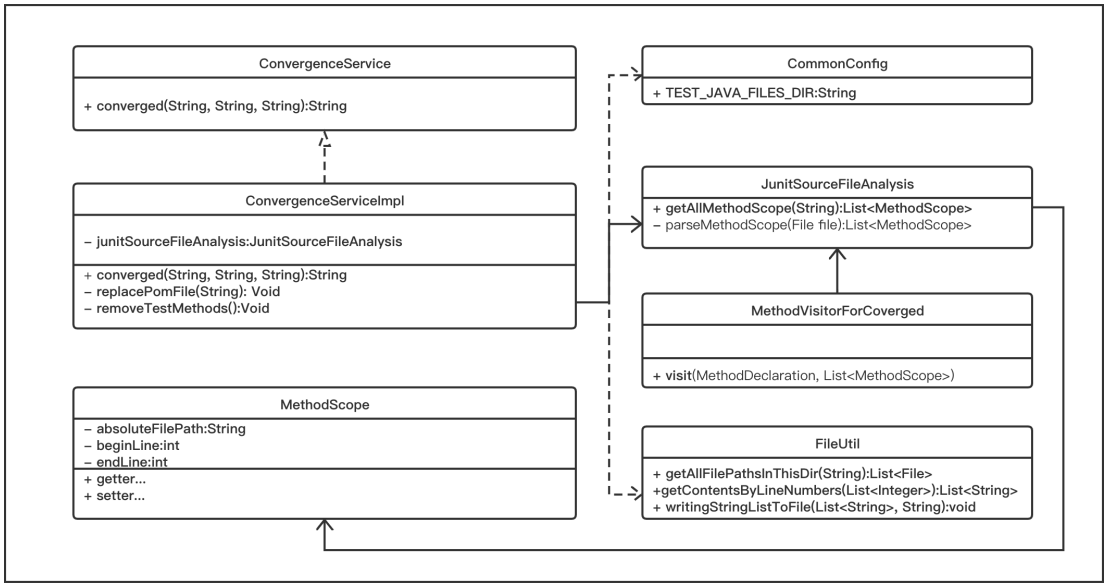

图 4.6: 测试项目模板构造类图

4.2.3　顺序图

测试项目模板构造子模块的顺序图如图4.7所示。Controller 调用 converged 触发测试项目模板构造，分为三个步骤。步骤一调用 replacePomFile 方法将待切片项目中的 pom 文件进行替换，替换为内嵌 OpenClover 覆盖分析插件的 pom.xml 文件。步骤二调用 removeTestMethodLines 方法分析测试脚本文件的抽象语法树，得到所有单元测试方法的起始行和结束行，通过 FileUtils 的 getContentsByLineNumbersNotIn 方法获取所有测试方法以外的代码行构造空的测试脚本模板。之后通过 writeStringListToFile 方法将构造好的测试脚本模板进行保存。步骤三为读取每一个测试代码片段融入到空测试模板中，将融合后的完整测试脚本保存。

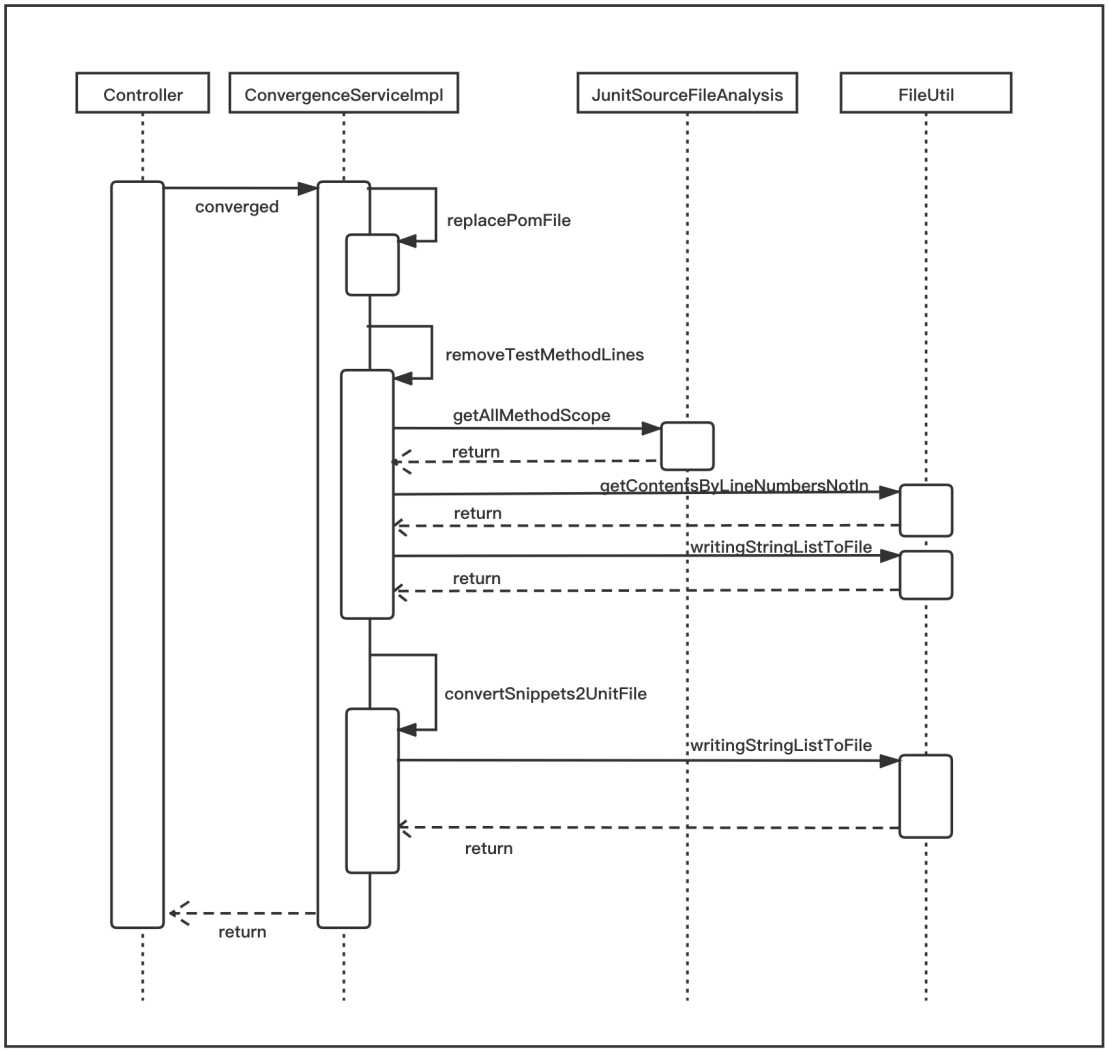

图 4.7: 测试项目模板构造顺序图

4.2.4　关键代码

测试项目模板构造子模块的关键代码如图 4.8所示，在保证不影响阅读的情况下, 本文删除了一些冗长的代码并以注释的形式进行代替。测试项目模板构造部分的核心方法为 converged 方法，其参数 projectPath 为测试模板项目。步骤一为替换该测试模板项目的 pom.xml 文件，步骤二中识别所有的测试方法并记录其行号范围。步骤三删掉行号范围内的所有内容，将代码片段嵌入其中得到测试模板文件并进行保存，其完整的代码在私有方法 convertSniippets2UnitFile 中进行实现，方法内的 merge 方法为合并模板文件和测试代码片段文件。

```
public String converged(String projectPath, String codeSnippetsPath, String
baseOutputDir) {
        //step.1 替换 pom 文件
        replacePomFile(projectPath);
        //step.2 分析 Test 文件，删掉@Test 的方法
        removeTestMethodLines(projectPath);
        List<File> files = FileUtil.getAllFileInThisDir(projectPath);
        List<File> codeSnippets = FileUtil.getAllFileInThisDir(codeSnippetsPath);
        //step.3 将代码片段融入代码片段
        convertSnippets2UnitFile(baseOutputDir, files, codeSnippets);
        return baseOutputDir;
}
private void convertSnippets2UnitFile(String baseOutputDir, List<File> files, List<File>
codeSnippets) {
        files.forEach(file -> {
            codeSnippets.forEach(codeSnippet -> {
                List<String> f = merge(file, codeSnippet);
                FileUtil.writingStringListToFile(f,path);
            });
        });
}
```

图 4.8: 测试项目模板构造关键代码

4.3　测试覆盖分析模块详细设计与实现

4.3.1　流程设计

覆盖率分析子模块是构建测试代码语料库的最后一个步骤。该步骤的流程如图 4.9所示。首先进行准备模板项目，删除内部的所有单元测试文件，之后填充上一步中准备好的测试模板脚本。最后运行 OpenClover 工具进行覆盖率分析。运行顺利后产生覆盖报告。最后阶段将得到的数据和测试代码片段进行整合，共同构建成一个可用的测试语料并将其持久化到存储介质中。

OpenClover 工具提供 Maven 插件，该插件可以在 Maven 的生命周期中触发和运行。OpenClover 生成的覆盖率报告为 XML 格式的文本数据。但其并非直接可用的数据结构，因此需要进行额外的转换，本系统在这一步骤中为减少开发的工作量以及为更好的复用，使用 慕测 平台提供的覆盖率分析转化工具。该工具可对 XML 格式的数据进行处理，精炼数据结构后输出一个 JSON 格式的文本。分析覆盖报告阶段的过程就是在对该 JSON 文本提取方法覆盖情况、分支覆盖情况和语句覆盖情况的解析。

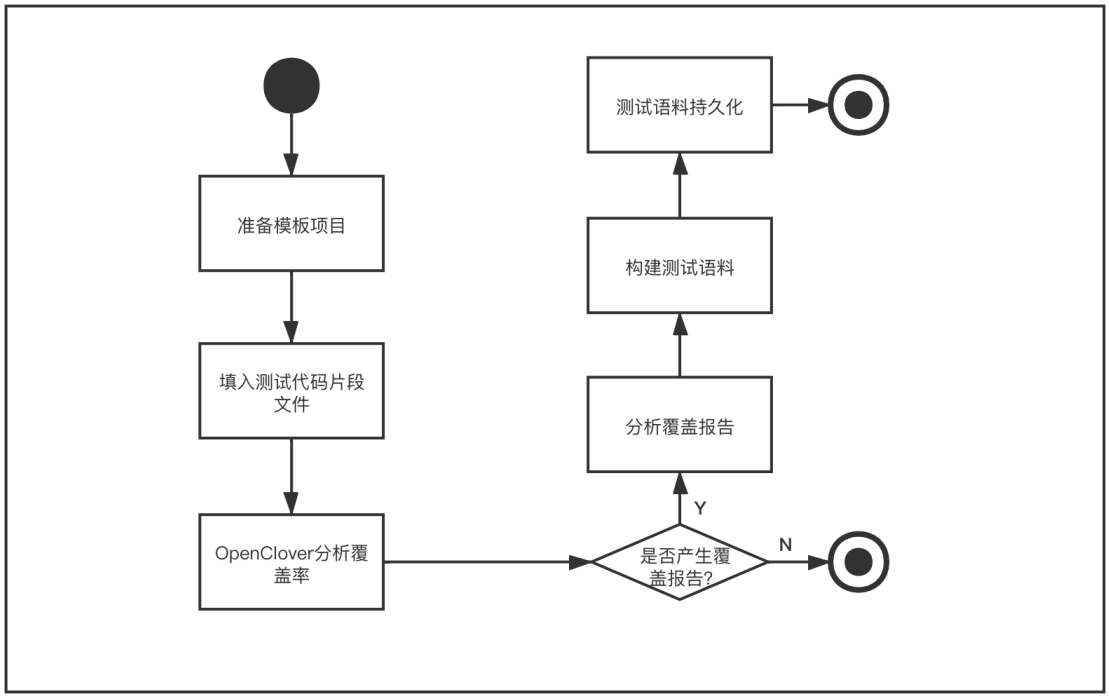

图 4.9: 覆盖率分析流程图

4.3.2　类图设计

覆盖率分析子模块的类图如图 4.10所示，该部分共分为两个处理流程，首先进行覆盖率报告生成，由 NodeAnalysisService 接口进行处理。上文介绍过 OpenColver 被集成到 Maven 的生命周期中，因此为得到覆盖率报告，NodeAnalysisService 接口的实现类 Impl 会调用 ShellCommend 执行 OpenColver 提供的 mvn 执行命令得到覆盖率报告，这个过程同样会设计到文件读取写入等操作，需要用到 FileUtil 进行处理。

接着，系统需要进行覆盖率报告的解析和测试代码语料的持久化，由 StorageService 接口提供。StorageServiceImpl 为 StorageService 的实现类，getFunctions 方法负责提取覆盖报告中覆盖到的方法列表。buildVectors 方法负责提取覆盖向

量，包括方法向量、分支向量和语句向量。CaughtNode 类为覆盖报告的抽象，其中 name 表示该覆盖的名称、ifCatch 代表是否成功覆盖、category 代表覆盖的类型。EliteCode 为持久化结构的抽象表现，将在动态推荐模块使用。

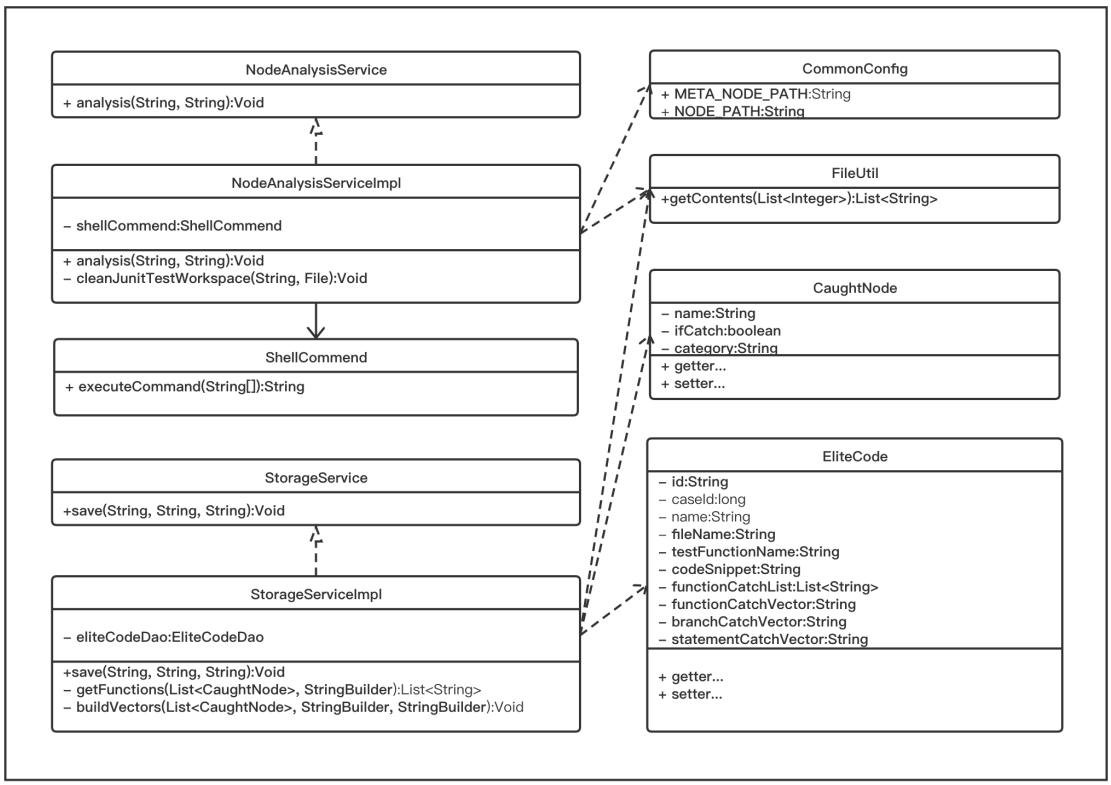

图 4.10: 覆盖率分析类图

4.3.3　顺序图

覆盖分析子模块的顺序图如图 4.11所示，省略掉校验等流程后，首先对测试代码片段构成的项目进行覆盖分析。第一步是清理掉测试模板项目的 junit 测试文件空间并将我们的测试代码片段文件放入该项目中，第二步调用 ShellCommend 执行 mvn 命令，调用 OpenClover 工具进行源码插桩，并运行覆盖分析得到覆盖分析报告。之后进行测试代码语料库构建，StorageServiceImpl 会调用 FileUtil 获取到刚才得到的覆盖报告。getFunctions 解析该覆盖报告得到该代码片段覆盖到的方法列表，buildVectors 得到方法、分支和语句维度的覆盖向量。将向量、覆盖方法列表和代码片段共同构建成测试语料 EliteCode。最后通过 EliteCodeDao 将测试语料持久化到 ElasticSearch 搜索引擎中。

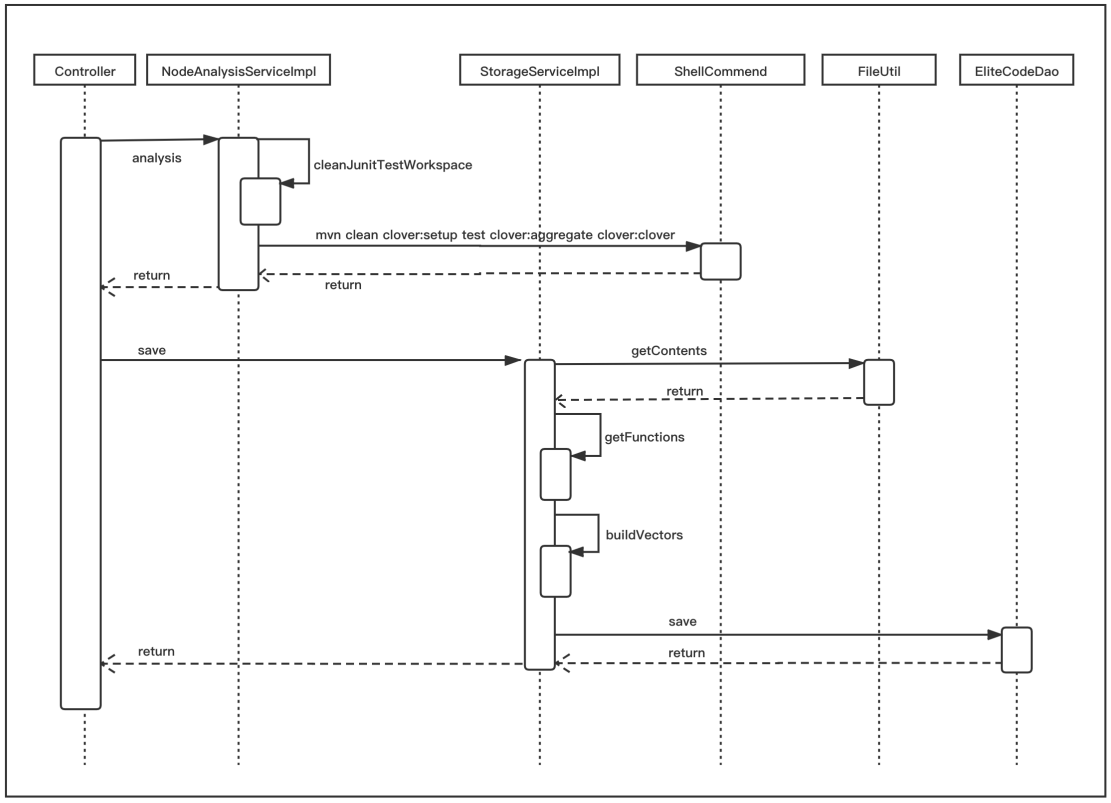

图 4.11: 覆盖率分析顺序图

4.3.4　关键代码

覆盖分析子模块的整体过程代码如图 4.12所示。process 方法为覆盖分析的整体处理过程，首先通过 FileUtil 读取在融入模板模块中得到的测试代码片段完整脚本文件。每个测试脚本文件均包含一个程序切片得到的测试代码片段。nodeAnalysisService 通过 analysis 方法调用 OpenClover 工具得到覆盖报告。storageService 的 save 方法对 OpenClover 输出的覆盖报告进行解析和存储。

```
void process() {
    List<File> fileList = FileUtil.getAllFileInThisDir(basePath);
    for (int i = 0; i < fileList.size() ; i++) {
        nodeAnalysisService.analysis(project,fileList.get(i));
        storageService.save(project,codeSnippets,fileList.get(i).getAbsolutePath());
}}
```

图 4.12: 覆盖分析代码片段

对于每一份覆盖报告最终都是以 Json 的格式进行展示的，且按照插桩的顺序进行排列，例如，方法 1 覆盖信息和方法 2 覆盖信息之间一定为方法 1 内的所有分支覆盖信息和语句覆盖覆盖信息。提取各类覆盖信息为覆盖向量的过程如图4.13所示。

```
private StringBuilder buildVector(List<CaughtNode> caughtNodes) {
    //省略 i，j 等控制变量的定义、自增和判断过程语句
    while (i < caughtNodes.size()) {
        if(caughtNodes.get(i).getCategory().equals("method") &&
caughtNodes.get(i).isIfCatch()) {
            while (!caughtNodes.get(j).getCategory().equals("method")) {
                if (caughtNodes.get(j).getCategory().equals("branch")) {
                    if (caughtNodes.get(j).isIfCatch()) {
                        branchV.append(1);
                    } else {
                        branchV.append(0);
                    }
                }
            }
            branchV.append(",");
        }}
    return branchV;
}
```

图 4.13: 转化覆盖报告提取覆盖向量

4.4　动态代码推荐模块详细设计与实现

4.4.1　流程设计

动态代码推荐模块流程图如图 4.14所示，用户在使用 慕测 WebIDE 在线智能编程平台进行开发者测试的学习过程中，会提交已编写测试代码运行单元测试。系统会在用户运行代码成功后提取用户产生的测试覆盖报告，计算当前次提交所覆盖到的各个方法的覆盖率，以百分制分数的形式存储于数据库中。

接下来进行的是动态推荐过程，首先判断学生提交次数是否大于设定的指定次数，以及各个方法的覆盖率是否处于指定推荐范围内。如果不存在潜在推荐方法则直接返回“当前无需进行推荐”。筛选出待推荐方法后，使用方法签名和题目相关信息检索相关代码片段。由于检索的数据仅仅是通过方法签名过滤得到的相关搜索记录，需要进一步过滤。因此使用学生提交数据的分支覆盖向量和语句覆盖向量和语料库库中测试代码片段的覆盖向量计算 Jaccrad 相似度，最后选择 TopK 数据返回给用户，完成一次推荐过程。

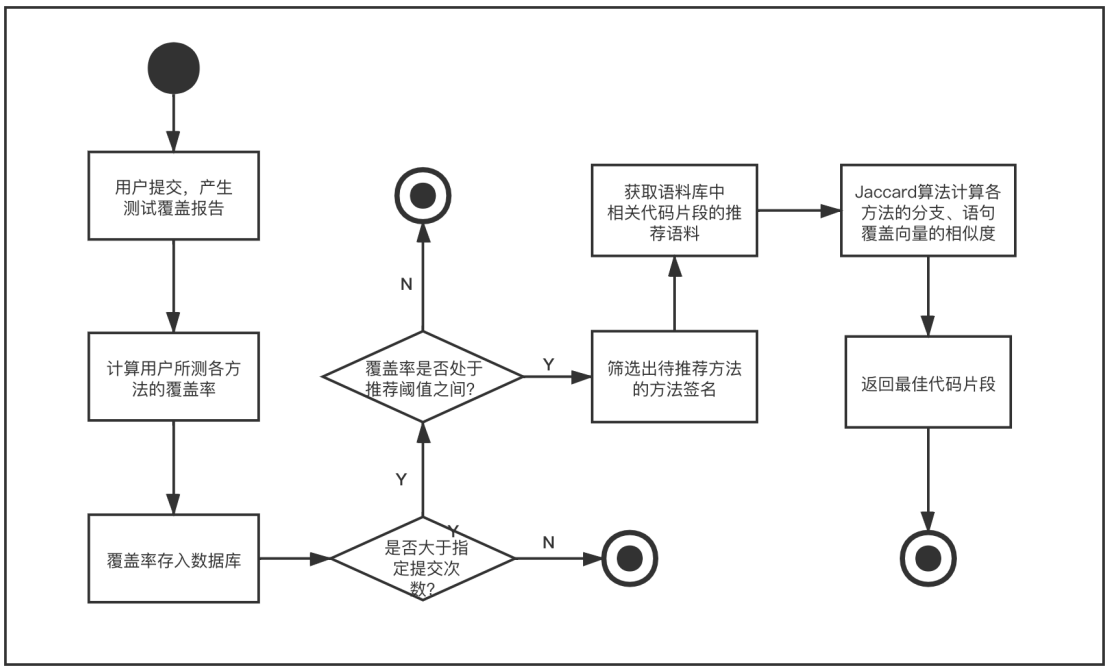

图 4.14: 动态代码推荐流程图

4.4.2　类图设计

动态代码推荐模块类图如图4.15所示，ConverageService 用与存储学生提交的覆盖信息，类图中省略了 methodCoverageDao 相关类。saveNodeCoverage 方法通过解析 List<CaughtNode> 列表中的内容，计算每个方法的覆盖率以百分制分数的形式存储。

TestMethodTargetSelector 用于从用户存储的覆盖信息中挑选出需要进行推荐的方法列表，DefaultTestMethodTargetSelector 是默认实现方式，其中 ifNeedSelector 方法通过获取最近几次学生提交的方法覆盖率得分，筛选出成绩多次没有发生变化且分数处于推荐阈值的被测方法。

TestMethodContentSelector 用于筛选出可推荐的测试语料。其默认实现类

DefaultTestMethodContentSelector 提供 jaccardSimilar 方法，该方法完成用户的覆盖向量和语料库代码片段覆盖向量 Jaccard 相似度的计算过程。recommendCount 表示应推荐的测试方法片段数量，可动态进行配置。ElasticSearchService 封装针对测试代码语料库的基本查询功能，在本模块会用到其提供的方法 findByFunctionName 搜索测试代码语料。

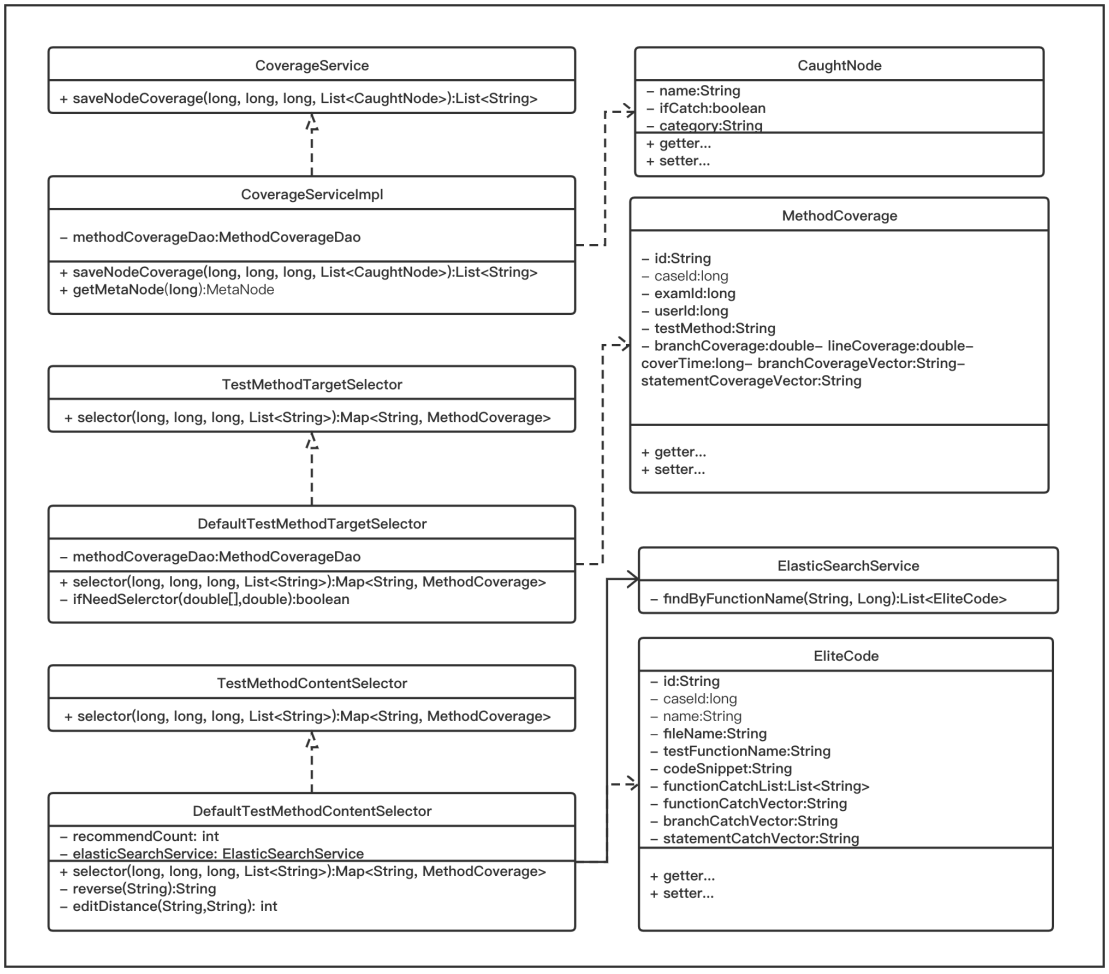

图 4.15: 动态代码推荐类图

4.4.3　顺序图

动态代码推荐模块流程图如图 4.16所示，用户通过方法 recommendCode 触发动态推荐过程，CoverageServiceImpl 分析用户提交的覆盖报告计算每一个方法的覆盖得分和生成覆盖向量，调用 CoverageDao 将结果持久化到数据库中。

接下来 Controller 调用 TestMethodTargetSelector 进行待推荐方法的筛选，首先查询最近几次学生提交的覆盖数据，通过 ifNeedSelect 方法判断每个方式在最

近几次提交中是否始终无法提高覆盖得分，最后将筛选得到的待推荐方法签名列表返回给 Controller 控制层。

Controller 控制层调用 TestMethodContentSelector 方法针对每一个待推荐的方法签名，查询语料库中是否有可推荐的代码片段。选择出的代码片段通过 filter 方法计算每一个方法覆盖向量的相似度，选出最能互补到学生所写测试代码覆盖情况的 TopK 个代码片段。Controller 在得到针对每一个待推荐方法的代码片段列表后返回给用户。

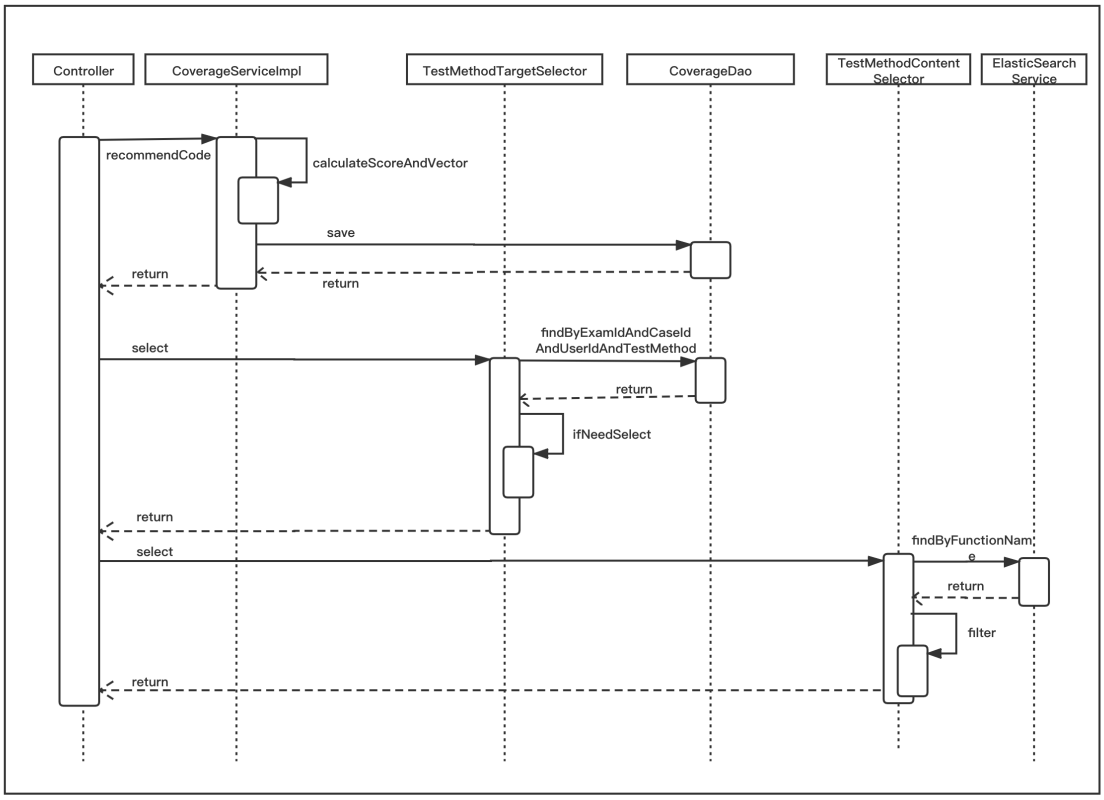

图 4.16: 动态代码推荐顺序图

4.4.4　获取待推荐方法列表关键代码

获取待推荐方法列表的关键代码如图4.17所示，首先通过调用 SpringBoot 内置的数据处理方法获取用户最近几次提交的方法覆盖得分数据 MethodCoverage，MethodCoverage 对象中包含该方法的签名、存储时间、覆盖得分等属性。然后提取 MethodCoverage 集合按照存储时间 coverTime 进行倒序排列并选择 windowSize 区间范围的分数数组。该分数数组可表示用户针对某一个方法在最近几次提交中的覆盖得分。接下来系统会调用类内私有方法 ifNeedSelect，通过传入得分数组来判断该方法是否需要进行推荐。

ifNeedSelector 方法会判断出该用户的此次提交数据的覆盖率情况是否处在待推荐范围内，并且和前几次提交相比分数没有显著提高，最后将筛选后的方法签名存储到 methodShouldChoose 的 Map 中并返回。获取待推荐方法核心代码为方便日后迭代过程中扩展，使用设计模式中的模板模式。通过扩展接口实现对应的 ifNeedSelect 方法即可优雅的实现其他算法形式的业务逻辑来获取待推荐方法列表。

```
public Map<String, MethodCoverage> selector(long examId, long userId, long caseId,
List<String> method2Choose) {
    Pageable pageable = PageRequest.of(0,windowSize+1,
            Sort.by(Sort.Direction.DESC,"coverTime"));
    for (String method:method2Choose){
        Page<MethodCoverage> methodCoveragePage = methodCoverageDao
                .findByExamIdAndCaseIdAndUserIdAndTestMethod(examId,
                        caseId,userId,method,pageable);
        double[] branchScoresDuringWindow = methodCoveragePage.stream()
                .mapToDouble(MethodCoverage::getBranchCoverage).toArray();
        boolean needSelect =
                ifNeedSelerctor(branchScoresDuringWindow,lowThreshold);
        if (needSelect) {
            methodShouldChoose.put(method,methodCoveragePage
                    .stream().findFirst().orElse(new MethodCoverage()));}
    }
    log.info("examId:[{}],userId:[{}],caseId:[{}],
            需要推荐的方法列表:{}",examId,userId,caseId,
            Arrays.toString(methodShouldChoose.keySet().toArray()));
    return methodShouldChoose;
}
```

图 4.17: 获取待推荐方法列表

4.4.5　获取测试代码片段列表关键代码

获取待推荐方法列表代码如图 4.18所示，method2Recommendate 为上一步中获取到的待推荐方法签名。针对每一个待推荐方法签名通过 ElasticSearch 检

索到相关的测试代码片段列表。本段代码选择展示的是只通过分支覆盖率的相似度计算每个测试代码片段的得分，首先将用户的覆盖向量进行 01 互补反转，之后和代码片段的分支覆盖向量计算 Jaccard 相似度暂存。

等到所有相似度计算成功后进行排序和过滤操作，Java8 的特性 Stream 流式处理可以很方便的帮助我们根据向量相似度进行从大到小排序以及对排序后的列表进行 TopK 截断，这里通过 Java8 的 Lambda 方法 sorted 和 limit 方法处理。最后返回处理后的 eliteCodeMap。

```
public Map<String, List<EliteCodeVO>> selector(Map<String, MethodCoverage>
method2Recommendate) {
    for (String x:method2Recommendate.keySet()) {
        List<EliteCodeFromES> eliteCodeFromESList = elasticSearchService
                .findByFunctionName(x,method2Recommendate.get(x).getCaseId());
        for (EliteCodeFromES eliteCodeFromES : eliteCodeFromESList) {
            int index = findFunctionIndex(x, eliteCodeFromES);
            String[] branchV = eliteCodeFromES.getBranchCatchVector().split(",");
            if (index != -1 && branchV.length > index) {
                eliteCodeFromES.setDistance(jaccardSimilar (
                                reverse(method2Recommendate.get(x)
                                        .getBranchCoverageVector()),
                                branchV[index]));
            }
        }
        List<EliteCodeVO> result = eliteCodeFromESList.stream()
                        .sorted(Comparator.comparingInt(EliteCodeFromES::getDistanc
                         e)).limit(count).map(ec -> eliteCodeVOWrapper.wrapper(ec))
                        .collect(Collectors.toList());
            eliteCodeMap.put(x,result);
    }
    return eliteCodeMap;
}
```

图 4.18: 获取测试代码片段列表关键代码

4.5　本章小结

本章详细介绍测试代码切片模块、测试代码片段融入模板模块、覆盖率分析模块和动态代码推荐模块的具体设计和实现。本章分别阐述各模块的流程设计、类图设计和顺序图设计，并针对每一个模块中的关键代码进行详细说明。

第五章　系统测试与案例分析

5.1　系统测试

为保证系统在投入使用时可以正常稳定的运行，需要对系统进行全方面的测试。本小节介绍在项目开发中和结束时针对测试代码推荐系统进行的相关测试工作。

5.1.1　测试目标与测试环境

测试用例代码推荐系统可分为两个主要模块，动态代码推荐模块和构建测试片段语料库的离线数据处理模块。为保证后续开发者在进行项目开发时，不会受到原有代码的影响，本系统应用自动化测试和持续集成的思想，为项目构建 Jenkins Pipline[1]进行持续集成和部署，该过程会对每一次提交的代码运行系统的单元测试，保证代码的正确性。为了解测试用例代码推荐系统的承载能力，我们在项目部署后对系统进行单机性能测试，该测试结果将作为日后服务压力增大后弹性扩容的标准参考数据。

表 5.1: 测试环境说明

服务器	配置说明	部署服务说明
阿里云 A	配置：4C16G 系统：Ubuntu 16.04 带宽：外网 4M 容器：Docker 17.09.1-ce	反向代理：Nginx 1.12.2 WebIDE 平台: 用户编程的工作空间 WebIDE 语言服务：用户编程所用基础代码提示 WebIDE 运行服务：用户提交代码运行编译服务
阿里云 B	配置：4C8G 系统：Ubuntu 16.04 带宽：外网 4M 容器：Docker 17.09.1-ce	测试用例推荐服务：处理用户推荐请求 ElasticSearch:7.5 MySQL 数据库：5.7
开发机	配置：8C16G 系统：MacBookPro 10.15.2 容器：Docker 19.03.5	离线数据处理：构建测试用例语料库

如表5.1所示，为本系统所部署测试环境的基础配置，测试环境用到两台阿里云 VPC 服务器，服务器通过阿里云内网相连。两台服务器均为 Linux 的 Ubuntu16.04 服务版本并部署 Docker17.09 版本的容器服务。阿里云服务器 A 部署用户进行开发者测试用到的 WebIDE 平台，Nginx 反向代理，用于处理编译运

[1]https://jenkins.io/doc/book/pipeline/

行的服务和提供基本的代码提示功能的语言服务。阿里云服务器 B 则部署推荐相关的语料库存储 ElasticSearch 引擎，关系型存储数据库 MySQL 和测试用例推荐服务。两台服务器通过内网相连，使用 HTTP/TCP 协议进行数据通信。开发机用于进行单元测试以及构建测试用例语料库。

5.1.2　单元测试

单元测试是软件开发周期重要的一环，其作用是验证软件的组成单元的正确性以及保证后续迭代过程的可靠性。系统开发过程中采用 MVC 的架构模式，分别包含用于处理业务逻辑的 Server 层，用于存储数据 Dao 层，以及用于外部请求的 Controller 层。由于 Controller 层提供的外部服务，表现为相关接口的形式，不包含具体的业务逻辑，因此无需进行单元测试。

本系统使用 Junit 框架和 Mockito 框架完成系统各个部分的单元测试。离线数据处理模块的单元测试用例如下表5.2所示，为本系统所包含的单元测试用例描述，由于篇幅的限制只展示离线数据处理模块所用到的核心单元测试用例。其余单元测试同表中类似，本文不再过多介绍。

其中 eliteCodeDaoTest 用于将测试代码片段持久化存储，sliceServiceTest 用于处理切片，包含正常的处理流程以及项目不存在或者参数为 Null 的异常处理流程。covergenvceServiceTest 测试用例用于测试生成测试脚本的正确性。node-AnalysisServiceTest 用于测试覆盖分析结果流程的正确性以及当传入项目编译失败是否可正常终止并返回报错信息。storageSericeTest 用于测试构造测试代码语料的正确性。shellCommend 测试用例用于测试执行 Shell 命令的正确性。

表 5.2: 离线数据处理模块测试用例表

单元测试编号	**单元测试签名**
UT-eliteCodeDao-1	test_eliteCodeDao_saveSuccess()
UT-eliteCodeDao-2	test_eliteCodeDao_saveFail()
UT-sliceService-1	test_sliceService_sliceProjectSuccess()
UT-sliceService-2	test_sliceService_sliceProjectWithNullProject()
UT-sliceService-3	test_sliceService_sliceProjectWithErrorProjectPath
UT-covergenceService-1	test_covergenceService_convergenceCodeSnippetSuccess()
UT-covergenceService-2	test_covergenceService_convergenceIllegalCodeSnippetPath()
UT-nodeAnalysisService-1	test_nodeAnalysisService_analysisSuccess()
UT-nodeAnalysisService-2	test_nodeAnalysisService_analysisFailProjectCompileFail()
UT-storageService-1	test_storageService_saveSuccess()
UT-storageService-2	test_storageService_getByIdSuccess()
UT-shellCommend-1	test_shellCommend_commendSuccess()
UT-shellCommend-2	test_shellCommend_commendIllegal()

离线数据处理模块的单元测试结果如图5.1所示，其中程序切片过程 Slice-Service 和覆盖率分析过程 NodeAnalysisService 需要调用第三方工具进行测试，且调用工具为同步阻塞模式，因此需要的处理时间较长。

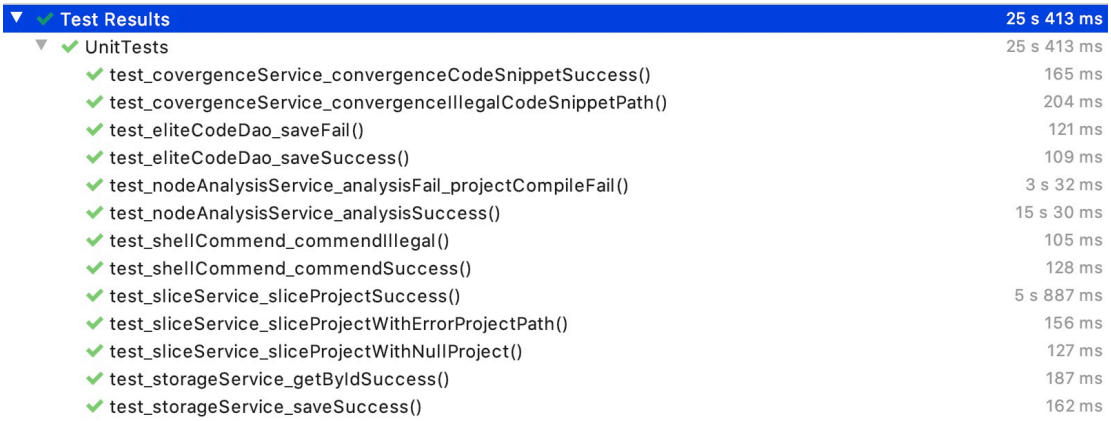

图 5.1: 离线数据处理核心方法单元测试结果说明

5.1.3　性能测试

本小节将介绍针对系统动态推荐接口的性能测试。该过程的目的是为探究单机部署下测试用例推荐系统的每秒查询率 QPS，该数据的作用是识别出系统可承载压力的上限，作为日后性能扩充的标准数据。受篇幅限制，本文只展示动态推荐代码片段接口的性能测试结果，该接口承担的业务逻辑包括持久化用户实时覆盖信息、筛选应推荐的测试方法、搜索推荐代码片段以及相似度算法过滤代码片段等功能。

在真实的用户使用场景中，推荐伴随用户运行代码而进行被动触发，用户运行一次的频率约为 20s。本次性能测试模拟并发量分别为 120，160，200，240 情况下系统的运行情况，可以看出在单机模式下系统吞吐量为 7 个/秒，错误率为 0，而系统的相应时间随着用户并发量的提升逐步增加。因此为得到秒级的响应速度，本文推荐部署该系统到 4C8G 服务器上承载量为 120QPS。在线上环境部署系统时需考虑用户实际并发量来调整推荐服务的副本数。

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request	120	124	122	151	170	262	81	583	0.000%	6.0344	1.76	10.82
HTTP Request	160	441	287	912	1524	1919	80	2783	0.000%	7.56394	2.21	13.56
HTTP Request	200	3006	3083	5495	6250	6981	80	7943	0.000%	7.46714	2.18	13.39
HTTP Request	240	5691	5735	10240	10907	13041	95	14730	0.000%	7.4606	2.18	13.37

图 5.2: 动态推荐接口性能测试结果图

5.2　系统案例分析

本小节将介绍作为一个用户如何在 慕测 平台进行开发者测试学习过程中如何使用测试代码推荐系统辅助学习过程。

5.2.1　题目描述

ALU 是一道 Java 白盒测试题目，代码实现的功能是 java 语言版的算数运算单元。使用 OpenClover 覆盖分析工具分析该题目，并将 xml 的测试覆盖报告可视化可得到如下图5.3所示的测试覆盖信息内容，该题目中包含了 18 个方法，其中 adder、fullAdder、claAdder 是加法器的具体实现；integerRepresentation、floatRepresentation 等进行数值的进制转化；orGate、andGate、xorGate 方法用于实现二进制与或和异或操作。用户需要通过对这些方法进行单元测试，学习如何构建巧妙的测试用例完成白盒测试任务，掌握各个方法的实现逻辑。

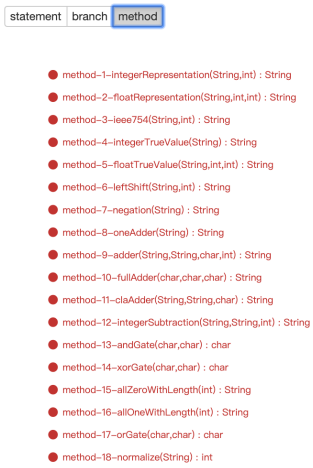

图 5.3: ALU 题目说明

integerRepresenttation 方法的代码截图如图5.4所示，integerRepresenttation 方法的具体实现内容是生成十进制的二进制表示，需要考虑传入的 number 为正数、负数、0 等数值以及相关边界值的问题。该方法包含 10 个被测分支，用于表示各个 if 和 while 判断为 true 和 false 时的分支情况，其相关信息如图5.5所示，考察初学者能否正确的通过边界值将该方法的全部分支进行完全覆盖。

```java
public String integerRepresentation(String number, int length) {
    StringBuilder result = new StringBuilder();
    result.append("");
    String tmpNum;
    boolean isMinus;
    if (number.charAt(0) == '-') {
        isMinus = true;
        tmpNum = number.substring(1);
    } else {
        isMinus = false;
        tmpNum = number;
    }
    int n = Integer.valueOf(tmpNum);
    while (n >= 1) {
        result.insert(0, String.valueOf(n % 2));
        n = (n - n % 2) / 2;
    }
    if (isMinus) {
        result = new StringBuilder(oneAdder(negation(result.toString())).substring(1, result.length() + 1));
    }
    while (result.length() < length) {
        if (isMinus) {
            result.insert(0, "1");
        } else {
            result.insert(0, "0");
        }
    }
    return result.toString();
}
```

图 5.4: integerRepresentation 实现代码截图

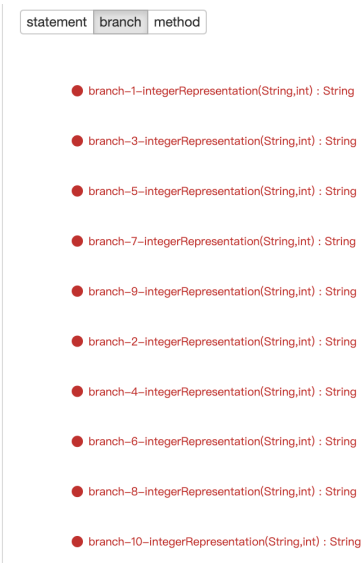

图 5.5: integerRepresentation 方法分支信息

5.2.2　推荐测试语料准备

为了证明测试用例推荐系统对于初学者存在帮助作用，需要准备针对 ALU 算数计算单元题目的测试代码语料。系统从 慕测 平台上获取了 10 位同学针对这道题目的测试脚本数据，他们的分支覆盖率集中在 90%-96% 之间。

使用本文描述的离线数据处理模块处理这 10 个测试脚本数据，经过程序切片后得到 338 个测试代码语料，经过覆盖分析后得到 322 个包含覆盖信息的测试代码语料。其处理结果如表5.3所示。从结果可看出使用程序切片和覆盖分析技术可正确处理得到 95.3% 的测试代码语料。

表 5.3: 对 ALU 进行程序切片得到测试语料结果表

Id	程序切片得到的切片数	可分析得到覆盖信息切片数
ALU_Student_1	27	25
ALU_Student_2	67	54
ALU_Student_3	34	34
ALU_Student_4	31	31
ALU_Student_5	26	26
ALU_Student_6	31	30
ALU_Student_7	31	31
ALU_Student_8	31	31
ALU_Student_9	31	31
ALU_Student_10	29	29

5.2.3　系统使用

用户登录到 慕测 WebIDE 后如下图5.6所示，通过打开右侧文件目录可展开如下页面。页面从左到右分为三部分，最左为当前工作空间的文件树，中间页面为用户进行学习时进行编辑的测试文件，用户需要在此文件中编写合适的白盒测试用例。右边的页面内容为被测文件，其中通过红绿黄三种颜色代码着色标注代码的覆盖情况，红色代表该行代码还未覆盖到；绿色代表该行代码被测试用例完全覆盖；黄色代码该行代码被测试用例部分覆盖到。该着色信息会在用户点击运行或者提交按钮时会触发。

同时触发的还有 慕测 平台测试评分系统，对项目进行编译运行和覆盖率计算，将语句覆盖、分支覆盖、方法覆盖等信息转化为百分制分数的形式，方便初学者了解自己的测试进度，以及方便初学者了解当前所编写测试用例的测试效果。当整体流程处理结束后，系统会通过 WebSocket 管道异步返回分支覆盖成绩、语句覆盖成绩和方法覆盖覆盖成绩。

用户如遇到多次运行且成绩无法提高的困境，可点击屏幕右侧的“测试用例推荐”选项卡，查看系统给出的推荐测试用例。页面如图5.7所示，页面所显示的推荐内容是系统通过分析用户每次提交时的覆盖报告得到的。该选项卡被设计成不会自动弹出的模式，目的是不主动干扰当前用户的学习过程。当用户需要代码推荐结果辅助学习时，可通过点击的方式进行主动触发。

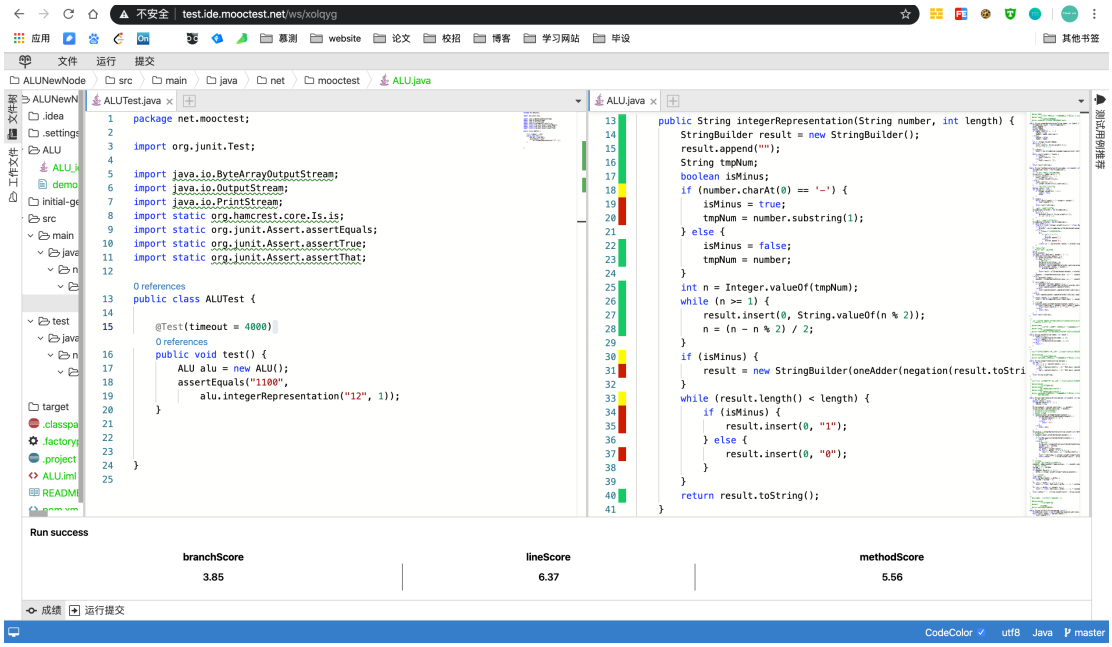

图 5.6: 使用 WebIDE 进行开发者测试学习

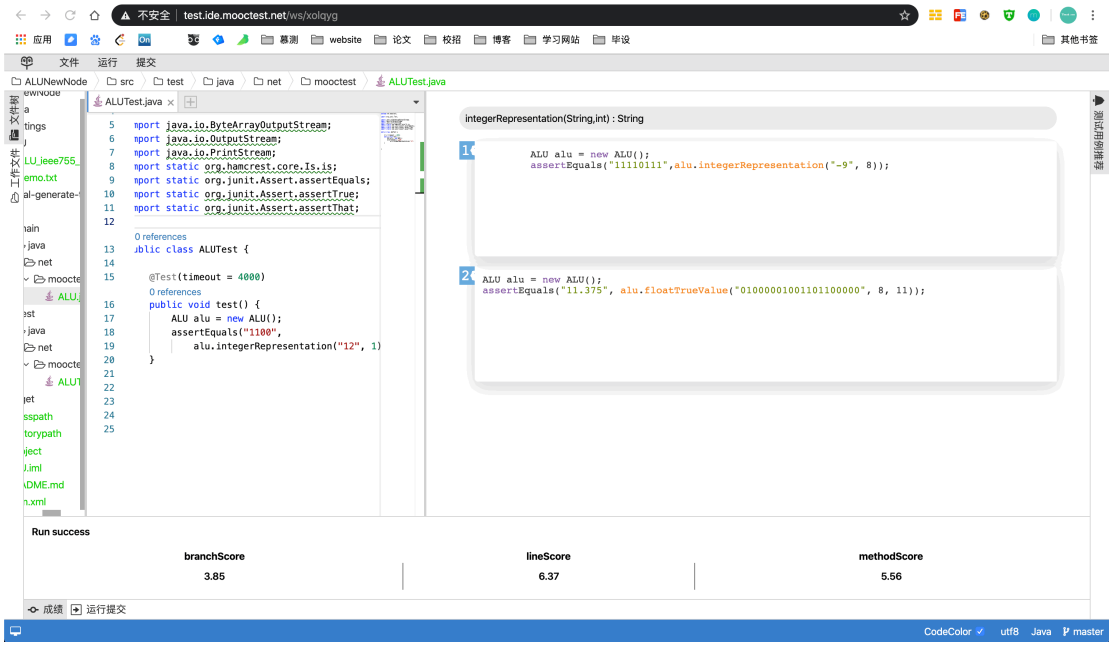

图 5.7: 点击测试用例推荐

以图 5.6中内容为例，当前用户所测试方法为 integerRepresentation，该方法的用途是生成十进制整数的二进制补码表示，其右侧的测试覆盖着色显示对于十进制数为负数的情况，该用户所写单元测试没有完成 100% 覆盖。此时用户点击“测试用例推荐”选项卡弹出图 5.7的界面。测试代码推荐系统通过分析当前

用户的覆盖信息得出该用户可能需要如下几个测试用例。可以直观的看到推荐测试用例的第一个为直接考虑输入参数负数的形式，符合用户的需要。第二个测试用例推荐结果并非直接测试的 integerRepresentation 方法，而是 floatTrueValue 方法的测试代码片段，该方法用于计算二进制原码表示的浮点数的真值，间接调用 integerRepresentation 方法，通过间接的方式测试到了 integerRepresentation 方法，且与用户所写代码覆盖互补性较高，因此系统也进行了推荐。

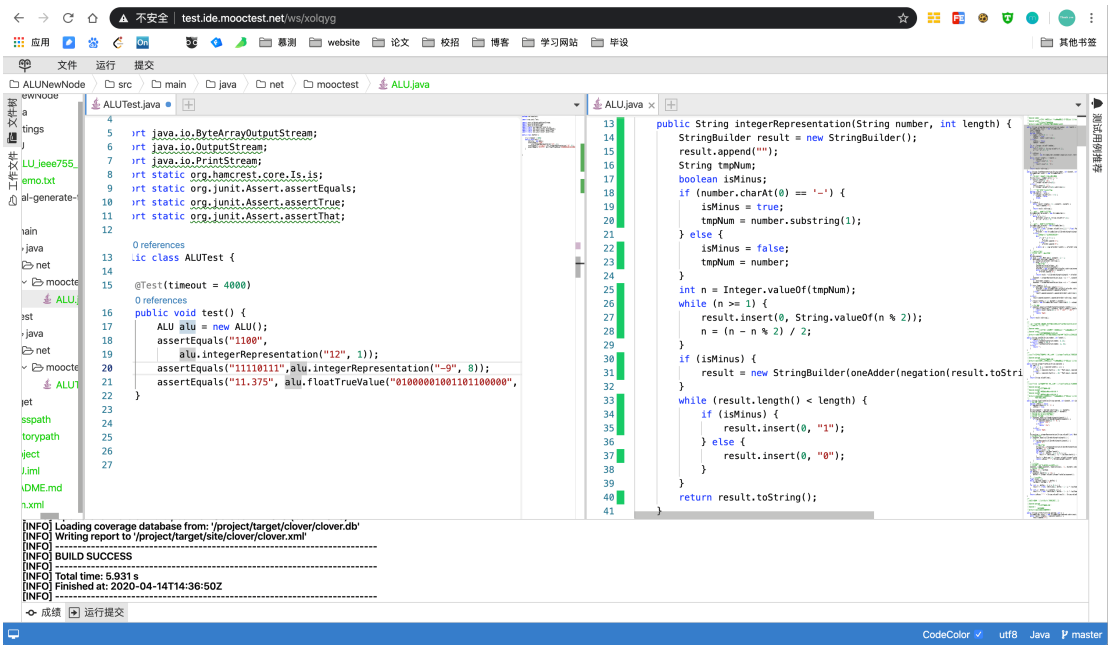

图 5.8: 使用推荐测试用例覆盖情况

用户再次触发运行时可看到运行结果如图 5.8所示，可以明显看到用户的测试覆盖得分有所提高且右侧覆盖代码着色中原来没有被覆盖到的分支颜色变成“绿色”，证明该推荐代码片段有效的帮助用户提高分支覆盖率，推荐系统统推荐的测试用例存在一定意义，对用户的学习过程存在帮助作用。

为验证测试用例系统通过程序切片构造的测试代码片段语料库是否可以覆盖到该题目的所有分支，我们将系统的推荐个数设置为 2，推荐频度设置为 1，含义是每次运行系统均进行推荐。其最终的推荐结果如图 5.9所示，可以看到测试代码推荐系统推荐的测试用例成功的覆盖该题目 96.15% 的分支、99.36% 的语句、以及 100% 的方法。接近 100% 的分支和语句覆盖表明系统可做到对任意分支进行推荐。学生遇到无法成功测试的分支和代码行，系统均可提供易理解、针对性强的测试代码片段。

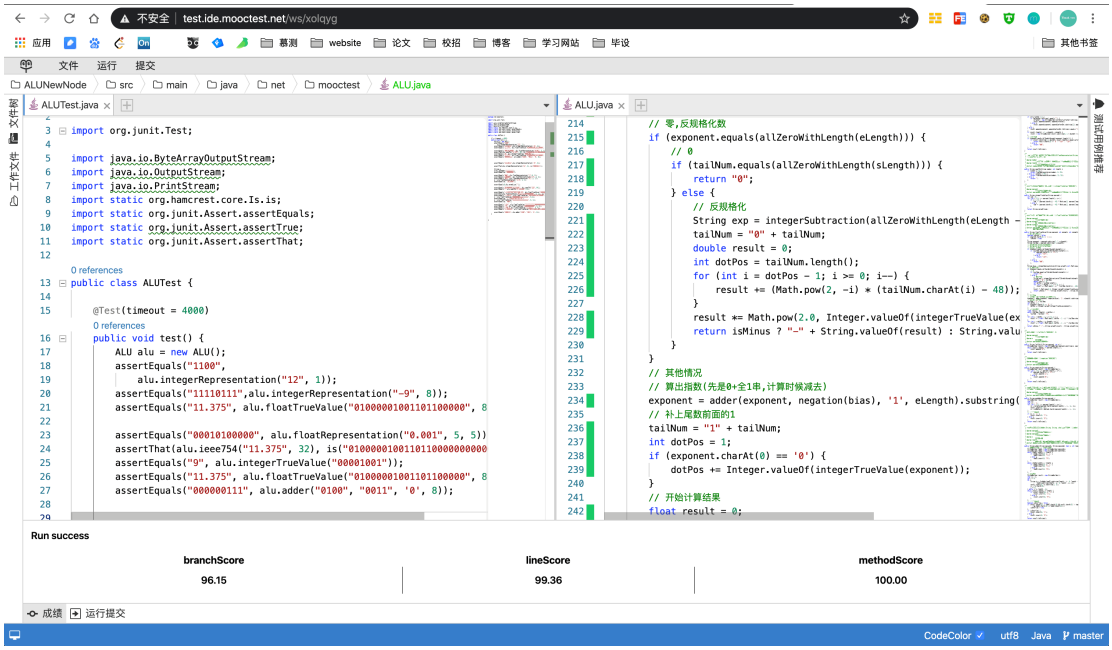

图 5.9: 推荐测试用例覆盖情况总览

5.3　系统实验

5.3.1　实验设计

为了验证本系统针对初学者存在一定的帮助作用，并且可提高用户白盒测试学习效率，本文采用了实验调研的形式进行验证。通过邀请 20 位初入职场或者步入大学的无测试经验的初学者试用本系统，并回答一系列问题：

（1）本文采用的程序切片技术得到的代码片段相比未处理的代码片段的可读性相比，是否可读性更好？

（2）本系统是否可帮助打开测试思路？

（3）本系统是否有助于提高提高测试学习效率？

针对问题 1，我们分别用程序切片得到 3 组复杂程度不同的代码片段，同时保证不丢失覆盖信息。让用户给出处理前后每个代码片段的可读性评分。评分标准为 1-5 的整数数字，可用性越高，数字越大。

针对问题 2 和问题 3，我们分别让 20 位实验人员在有测试代码推荐和无测试代码推荐的环境中试用系统，最后通过问卷调查的方式收集用户的反馈。

5.3.2　实验结论

实验结果如图5.10所示，对于问题 1，结果表明“程序切片”处理后的代码片段为 253 分，处理前为 180 分，程序切片产生的测试代码片段更被初学者所接受。对于问题 2，20 位实验人员中，有 18 位认为系统可有效的帮助自己打开测试思路，而认为系统毫无作用的用户为 0。对于问题 3，同样有 18 位认可系统可提高测试学习效率，认为系统无法帮助自己提高效率的用户为 0。综上所述，本文阐述的基于切片覆盖过滤的测试代码片段推荐系统对于初学者的测试学习过程存在帮助作用，达到了本文最初设定的目标。

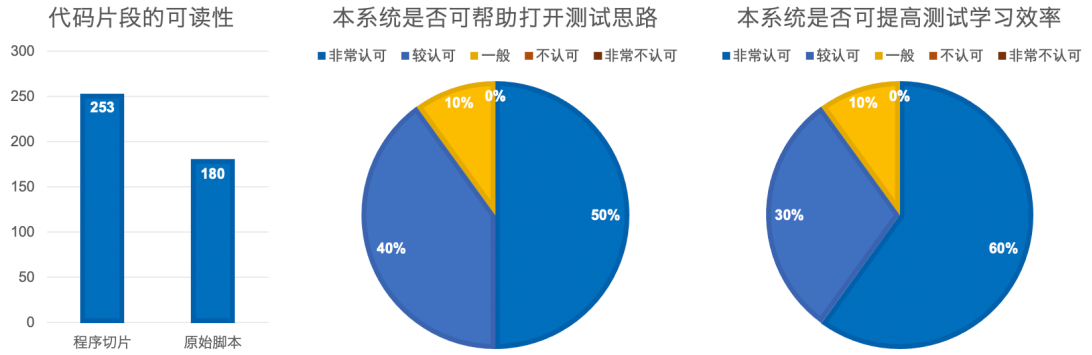

图 5.10: 实验结果

5.4　本章小结

本章从功能可用性的角度以及系统性能可靠性的角度对测试代码片段推荐系统进行多方面的测试，最后通过实际案例的形式展示动态代码推荐系统在使用过程中对初学者的帮助作用，证明系统在白盒测试教学场景下存在一定的实用性。

第六章　总结与展望

6.1　总结

随着软件应用场景多样化与复杂化，软件的架构和功能也变得越来越复杂。软件测试是软件生命周期中至关重要的一部分，越来越多无测试经验的软件开发人员和走进大学的学生投入到测试学习中。在测试领域，白盒测试一种非常重要的软件测试方法，是很多初入职场的无经验开发者要求掌握的必备技能。

许多初学者通过慕课、书籍、网络视频等途径学习白盒测试理论知识，并且通过测试用例自动化生成工具、覆盖率可视化工具等测试工具辅助编程学习。整体的学习过程不够连贯，并且相关测试工具对于初学者不够友好。通过自动化生成的测试用例有着随机性过高、生成代码可读性差的问题，而覆盖率可视化工具产生的结果太过简约、不够直观，不足以引导初学者进一步改进测试，提高测试质量。一种可行方案是在用户进行测试学习过程中推荐可读性强的测试用例，因此本文着手于现有测试辅助学习工具的不足开发基于切片覆盖过滤的测试代码推荐系统。

本系统使用程序切片技术处理 慕测 平台的学生在开发者测试考试和练习中提交的测试项目数据，得到一系列易于理解、可读性强的测试代码片段。本系统使用覆盖分析工具分析测试代码片段的覆盖信息并赋予每个测试代码片段覆盖语义，之后和所测方法进行关联。本系统使用有覆盖语义、易于理解的测试代码片段构建一个测试代码片段语料库用于推荐。最后再与 慕测 平台的智能化在线测试开发学习平台 WebIDE 进行有效的整合，提供一站式学习方式，降低无经验测试人员的学习门槛，达到提高初学者的测试学习效率的目的。

本文通过前两章的介绍详细阐述论文的选题背景和开发系统过程涉及的关键技术。第三章介绍测试用例推荐系统的需求分析和概要设计，完成系统的整体设计以及模块划分，并从 4+1 视图的角度介绍整体项目架构以及离线数据处理模块的架构设计。第四章从流程图、类图、顺序图设计的三个不同角度介绍项目具体实现，并通过给出关键代码的方式展示系统实现细节。最后通过性能测试和集成测试等多方面对系统进行全方位评估。在技术方面本系统使用 SpringBoot 作为开发框架；使用 Wala 程序切片工具进行测试代码切片处理；使用 OpenClover 分析代码片段的覆盖率；使用 ElasticSearch 作为测试代码片段的存储语料库；使用 Docker 做为程序部署容器。最后通过案例分析的方式展示系统具体运行方式以及证明系统对于无经验开发者存在帮助作用。

6.2　展望

本系统目前还存在一些问题，在后面的迭代开发中可从以下几个方面着手改进本系统：

（1）程序切片工具 Wala 不够稳定，学生提交的程序代码通过 Wala 切片后会出现丢失变量，缺少关键语句等问题，目前采用的解决方案是半自动化的方式，在切片完成后通过人工审阅修复代码片段丢失变量的问题。之后可通过完善切片流程以及集成更加完善的切片工具解决此问题。

（2）动态推荐过程仅考虑到分支覆盖相似度和语句覆盖相似度等两个因素。可通过添加更多推荐因素达到更好的推荐效果，例如推荐过程中考虑代码片段的语法结构和语义结构。

（3）系统的离线数据处理构建测试代码语料库的过程仅通过 Restful 方式进行调用，没有提供一个易用性更好的图形化界面进行管理。

（4）可以考虑将离线数据处理过程做成流式实时处理的形式，实时分析用户提交的测试脚本数据。通过此方式不断的扩充测试代码片段语料库，支持更多的学习题目，提供更加完善的学习环境。

参考文献

[1] P. C. Jorgensen, Software Testing: a Craftsman's Approach, 4th Edition, Auerbach Publications, 2013.

[2] 陈火旺, 王戟, 董威, 高可信软件工程技术, 电子学报 31 (b12) (2004) 1933–1938.

[3] 聂长海, 关于软件测试的几点思考, 计算机科学 (02) (2011) 7–9+33.

[4] 封亮, 严少清, 软件白盒测试的方法与实践, 计算机工程 26 (12) (2000) 87–90.

[5] Y. K. Malaiya, M. Li, J. Bieman, R. Karcich, Software reliability growth with test coverage, IEEE Transactions on Reliability 51 (4) (2002) 420–426.

[6] G. Fraser, M. Staats, P. McMinn, A. Arcuri, F. Padberg, Does automated white-box test generation really help software testers?, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA 2013, Association for Computing Machinery, 2013, p. 291–301.

[7] C. Gaffney, C. Trefftz, P. Jorgensen, Tools for coverage testing: Necessary but not sufficient, J. Comput. Sci. Coll. 20 (1) (2004) 27–33.

[8] R. Holmes, G. C. Murphy, Using structural context to recommend source code examples, in: 27th International Conference on Software Engineering, ICSE 2005, ACM, 2005, pp. 117–125.

[9] M. M. Rahman, C. K. Roy, On the use of context in recommending exception handling code examples, in: 14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, Victoria, BC, Canada, IEEE Computer Society, 2014, pp. 285–294.

[10] B. Antunes, B. Furtado, P. Gomes, Context-Based Search, Recommendation and Browsing in Software Development, Springer New York, 2014.

[11] L. Ai, Z. Huang, W. Li, Y. Zhou, Y. Yu, SENSORY: leveraging code statement sequence information for code snippets recommendation, in: V. Getov, J. Gaudiot,

N. Yamai, S. Cimato, J. M. Chang, Y. Teranishi, J. Yang, H. V. Leong, H. Shahriar, M. Takemoto, D. Towey, H. Takakura, A. Elçi, S. Takeuchi, S. Puri (Eds.), 43rd IEEE Annual Computer Software and Applications Conference, COMPSAC 2019, Milwaukee, WI, USA, IEEE, 2019, pp. 27–36.

[12] M. Burrows, D. Wheeler, A block-sorting lossless data compression algorithm, Digital Systems Research Center Research Reports 1.

[13] M. Ghafari, C. Ghezzi, A. Mocci, G. Tamburrelli, Mining unit tests for code recommendation, Proceedings of the 22nd International Conference on Program Comprehension (2014) 142–145.

[14] S. M. Nasehi, F. Maurer, Unit tests as API usage examples, in: 26th IEEE International Conference on Software Maintenance (ICSM 2010), Timisoara, Romania, IEEE Computer Society, 2010, pp. 1–10.

[15] W. Janjic, C. Atkinson, Utilizing software reuse experience for automated test recommendation, in: H. Zhu, H. Muccini, Z. Chen (Eds.), 8th International Workshop on Automation of Software Test, AST 2013, San Francisco, CA, USA, IEEE Computer Society, 2013, pp. 100–106.

[16] R. Pham, Y. Stoliar, K. Schneider, Automatically recommending test code examples to inexperienced developers, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, p. 890–893.

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

[18] J. Brandt, M. Dontcheva, M. Weskamp, S. R. Klemmer, Example-centric programming: integrating web search into the development environment, in: E. D. Mynatt, D. Schoner, G. Fitzpatrick, S. E. Hudson, W. K. Edwards, T. Rodden (Eds.), Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, ACM, 2010, pp. 513–522.

[19] 孙继荣, 李志蜀, 王莉, 殷锋, 金虎, 程序切片技术在软件测试中的应用, 计算机应用研究 (5) (2007) 216–219+223.

[20] M. Weiser, Program slicing, in: S. Jeffrey, L. G. Stucki (Eds.), Proceedings of the 5th International Conference on Software Engineering, San Diego, California, USA, IEEE Computer Society, 1981, pp. 439–449.

[21] M. Weiser, Program slicing, IEEE Transactions on Software Engineering 10 (4) (1984) 352–357.

[22] S. Horwitz, T. Reps, D. Binkley, Interprocedural slicing using dependence graphs, ACM Trans. Program. Lang. Syst. 12 (1) (1990) 26–60.

[23] D. Liang, M. J. Harrold, Slicing objects using system dependence graphs, in: 1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, IEEE Computer Society, 1998, pp. 358–367.

[24] S. Horwitz, T. W. Reps, The use of program dependence graphs in software engineering, in: T. Montgomery, L. A. Clarke, C. Ghezzi (Eds.), Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, ACM Press, 1992, pp. 392–411.

[25] P. E. Livadas, S. Croll, System dependence graphs based on parse trees and their use in software maintenance, Inf. Sci. 76 (3-4) (1994) 197–232.

[26] 朱三元, 刘霄, 国内外软件逆向工程综述, 计算机应用与软件 (4) (1990) 3–13.

[27] 王雪莲，赵瑞莲，李立健, 一种用于测试数据生成的动态程序切片算法, 计算机应用 25 (06) (2005) 1445.

[28] 曹鹤玲, 姜淑娟, 鞠小林, 软件错误定位研究综述, 计算机科学 (2) (2014) 7–12+20.

[29] 曹鹤玲, 姜淑娟, 鞠小林, 王兴亚, 基于动态切片和关联分析的错误定位方法, 计算机学报 v.38;No.395 (11) (2015) 66–80.

[30] R. Giacobazzi, F. Logozzo, F. Ranzato, Analyzing program analyses, in: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’ 15, Association for Computing Machinery, New York, NY, USA, 2015, p. 261–273.

[31] J. Zhu, Symbolic pointer analysis, in: Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design, ICCAD ’ 02, Association for Computing Machinery, New York, NY, USA, 2002, p. 150–157.

[32] 姜文, 刘立康, Code coverage test of c/c++ software based on continuous integration 计算机技术与发展 028 (003) (2018) 37–41,46.

[33] W. B. Frakes, R. Baeza-Yates (Eds.), Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Inc., USA, 1992.

[34] M. McCandless, E. Hatcher, O. Gospodnetic, Lucene in Action, Second Edition: Covers Apache Lucene 3.0, Manning Publications Co., USA, 2010.

[35] C. Gormley, Z. Tong, Elasticsearch: The Definitive Guide, 1st Edition, O’ Reilly Media, Inc, 2015.

[36] J. Nickoloff, Docker in Action, Manning Publications, 2016.

[37] C. Boettiger, An introduction to docker for reproducible research, with examples from the r environment, Acm Sigops Operating Systems Review 49 (1) (2014) 71–79.

[38] P. Riti, Introduction to Continuous Integration and Delivery: With Docker, Jenkins, and Kubernetes, 2018.

[39] T. Elrad, R. E. Filman, A. Bader, Aspect-oriented programming: Introduction, Communications of the Acm 44 (10) 29–32.

[40] H. Suryotrisongko, D. P. Jayanto, A. Tjahyanto, Design and development of backend application for public complaint systems using microservice spring boot, Procedia Computer Science 124 (2017) 736–743.

[41] Kruchten, P.B., The 4+1 view model of architecture, IEEE Software 12 (6) 42–50.

简历与科研成果

基本情况 门铎，男，汉族，1996 年 7 月出生，天津市武清区人。

教育背景

2018.9 ~ 2020.6 南京大学软件学院 硕士

2014.9 ~ 2018.7 大连海事大学信息科学与技术学院 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

致　　谢

时光荏苒，岁月如梭，难忘的研究生阶段也进入了尾声。我首先要感谢我的导师陈振宇教授和房春荣老师以及黄勇老师，感谢老师们在我研究生阶段给予我的帮助。从项目的开发到论文的写作，都离不开三位老师的指导。感谢老师们多次的会议讨论让我详细了解了论文的目标和方向，感谢三位老师在项目开发阶段给予我的建设性意见和项目设计方面的帮助。感谢老师们细心教导，帮助我完成了整体论文的写作。

在 iSE 实验室学习和奋斗的经历是我这一生最难忘的时刻，从实验室同学们的身上，我看到了积极、乐观、靠谱的特性。在他们身上，我知道没有一件难事是做不成的。很高兴可以和他们一起学习、一起奋斗。

感谢我们的团队慕测 WebIDE 小组的韩奇、袁阳阳同学，我们拥有着一致的目标，感谢他们和我一起完成了慕测 WebIDE 相关项目。一起拼搏的时光总是让人难以忘记，一起通宵改 Bug、一起熬夜写论文、一起做演示视频等等的画面历历在目，希望大家在毕业后的日子里越来越好。同时还要感谢实验室的赵源学长、周赛同学、徐文远同学，感谢大家的帮助，在我项目遇到困难时总会伸出援助之手，帮助我克服困难、解决问题。

最后要由衷的感谢我的父母，受疫情的影响，在家闭关写毕设的日子中，他们是我的精神依靠。感谢父母的支持与理解，感谢父母尊重我的每一次选择与决定。是他们让我有着继续努力前行的动力，感谢你们！

《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：门铎

2020 年 5 月 24 日

论文题名	基于切片覆盖过滤的测试代码推荐系统的设计与实现				
研究生学号	MF1832126	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	1648145124@qq.com				
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）