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 Web 应用自动化测试系统报告生成服务的设计与实现

作 者 姓 名 张晨剑

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：MF1832232

论文答辩日期：2020 年 5 月 23 日

指 导 教 师：　　（签字）

The Design and Implementation of Report Generation Service in Web Application Automation Test System

By

Chenjian Zhang

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：Web 应用自动化测试系统报告生成服务的设计与实现

工程硕士（软件工程领域） 专业 2018 级硕士生姓名： 张晨剑

指导教师（姓名、职称）： 陈振宇 教授

摘 要

随着互联网技术的不断发展，不同用途的 Web 应用不断出现，Web 应用呈现出涉及领域广、复杂度高等特点，令 Web 应用测试相较于传统软件测试工作的难度进一步提升。使用现有的 Selenium 测试工具进行自动化测试，门槛高、成本大。为了降低测试成本，提升测试效率，研究 Web 应用测试自动化测试方法十分必要。

报告生成服务针对 Web 应用测试的特点，基于对测试执行的监控，实现 Web 应用自动化测试流程，包括测试结果的生成和测试报告、软件质量评估报告的展示，提供清晰、易用的测试报告。根据主要功能实现，服务可划分为执行引擎监控、自动化测试用例生成、测试报告以及软件质量评估报告四个模块。服务通过监控 Web 应用测试执行引擎的整个执行过程，获取记录相关参数。利用记录的日志、路径等信息，调用单页面的分析服务，通过整合这个测试流程中的分析服务结果，生成最终的测试报告以及软件质量评估报告。同时，利用记录的操作记录、执行路径等信息，生成基于 Selenium 的 Web 自动化测试用例，便于页面状态以及缺陷测试流程的复现。为了保证开发过程中效率和稳定性，在技术实现上采用主流框架，使用 Spring Boot 作为服务端框架、jQuery 作为前端框架。同时为了保证系统的稳定性和性能，采用 RabbitMQ 作为消息中间件、Redis 作为系统高速缓存。报告生成服务是 Web 应用自动化测试系统组成部分，该系统基于用户简单配置，实现了 Web 自动化测试。

本文针对 Web 应用涉及领域广、复杂度高的特点，从 9 个不同行业的选取了 50 个具有代表性的网站，对报告生成服务进行实验评估。实验表明，服务提供了可读性高、易用性强的测试报告。报告生成服务监控了执行服务 3 种浏览器的执行过程，生成包含缺陷复现方法的测试报告，通过报告生成服务生成的测试脚本成功率达到 99.8%，脚本复现缺陷的复现率达到 89.4%，缺陷分类准确率达到 97.5%。实验还发现，利用脚本复现缺陷比人工复现缺陷效率高将近 3 倍。目前报告生成服务已经集成并在慕测平台上线并且运行良好，为测试人员提供良好的 Web 应用自动化报告生成服务。

关键词： Web 应用测试，自动化，测试报告，测试用例

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 Report Generation Service in Web Application Automation Test System

SPECIALIZATION: Software Engineering

POSTGRADUATE: Chenjian Zhang

MENTOR: Professor Zhenyu Chen

Abstract

With the continuous development of Internet technology, Web applications for distinct usages are constantly appearing. Web applications appear to be widely used in many industries and highly complex which brings various challenges to its test work and the innovation of test technology. It makes high threshold and high cost that using Selenium to implement automated testing. For the purpose of reducing the testing cost, it is relatively necessary to research the Web application automatic test method.

According to the characteristics of Web application testing, report generation service implements the Web application automation test process based on the monitoring of test execution, including the generation of test results and the display of test reports, software quality assessment reports. The service provides clear and easy-to-use test reports.The service is consist of four modules, divided by function, including execution engine monitoring, automatic test case generation, test report and software quality assessment report. The system obtains and records related parameters through monitoring the entire execution process of Web application test execution engine. The single-page analysis service is invoked utilizing the recorded logs, paths and other information, and then the final test report is generated, so is the software quality assessment report. Meanwhile, Web automatic testing cases based on Selenium are generated for the purpose of facilitating the reproduction of each single-page and vulnerability test process, using the recorded operating procedures, execution paths and other information.In order to ensure the efficiency and stability in the development process, the mainstream framework is adopted in terms of technical implementation. Spring Boot and jQuery are respectively adopted as the server-end framework and the front-end framework. In addition, for the purpose of ensuring the stability and performance of the system,

RabbitMQ is employed as the message middleware and Redis is utilized as the system cache. Report generation service is a part of Web application automatic test system. The system implents an easy-to-use automated testing process using users' simple configuration.

For the characteristics of widely use and high complexity of Web applications, this thesis selects 50 representative websites from nine different industries to experiment on and evaluate report generation service. The experiment shows that the service can provide distinct and easy-to-use test reports. Report generation service monitors the execution process of the three browsers of the execution service, and generates a test report that includes the method of vulnerabilities reproduction. The success rate of the test script generated by report generation service reaches 99.8%, the recurrence rate of existing vulnerabilities reached 89.4%, and the accuracy rate of vulnerabilities classification reached 97.5%. The experiment also finds that using scripts to reproduce vulnerabilities is nearly three times more efficient than manually reproducing vulnerabilities. Currently, the service has been integrated on the Mooctest platform and is running well, providing testers with good Web application automation report generation services.

Keywords: Web Application Testing, Automation, Test Report, Test Case

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景与意义

随着互联网技术的不断发展，无论是商业、教育，还是人们的日常生活，网络对社会的各个方面都产生了重大的影响。随着基于网络的应用开发不断推进，不同用途的 Web 应用不断出现。相比较于传统软件应用，基于网络的 Web 应用具有明显的优势。首先，Web 应用无需安装，用户只需要通过计算机上的浏览器就可以轻松获取应用服务。其次，Web 应用更新便捷，相比与传统软件应用需要下载安装等过程，对于 Web 应用的升级，用户几乎是没有感知的并且无需任何操作。再次，用户获取服务不会受到物理限制，可以通过任何计算机随时访问 Web 应用。最后，Web 应用独立于客户端操作系统，用户无需考虑操作系统兼容性限制 [1]。

用户对 Web 应用质量的要求不断提高，Web 应用测试作为其软件生命周期的一部分，其地位日趋重要 [2]。良好的 Web 应用测试能够有效地减少 Web 应用中的缺陷、提高 Web 应用的质量。由于 Web 应用使用服务器和浏览器技术，使得其相较于传统应用软件更加容易出错并且难以测试，从而导致应用可靠性严重降低。除了可能导致修复维护成本以为，严重的还可能造成业用途中的经济损失以及信誉损失 [3]。

Web 应用自身具备的特征为应用软件的测试工作及测试技术的革新带来了种种挑战。Web 应用具有异构性，Web 应用程序可能使用多种不同的编程语言进行开发，导致 Web 应用系统实现方式复杂。其次，Web 应用程序通一般采用客户端/服务器架构实现，会使用异步的 HTTP 请求/响应模式进行交互和同步，存在网络延迟导致的同步问题。除此之外，Web 应用还具有动态特征和不确定性。良好的 Web 应用测试能够有效地减少 Web 应用中的缺陷，从而提高 Web 应用的质量 [4]。

Selenium 的出现，使得用户可以通过编写 Selenium 脚本模拟用户操作 Web 应用进行功能测试 [5]。但是编写 Selenium 脚本需要测试人员熟悉 Selenium 编写方法和前端语言并且需要编写大量的测试脚本，门槛高、耗费大。为了降低测试成本，研究 Web 应用测试自动化测试方法十分必要。

本文面向 Web 应用自动测试业务，针对 Web 应用测试的特点，基于对测试执行引擎的监控，实现完整的 Web 应用自动测试流程，包括测试结果的生成和测试报告、软件质量评估报告的展示，构建 Web 应用自动化测试系统报告生成

服务。在测试流程的基础上，引入 Selenium 工具，实现自动化测试用例生成，从而使用户在查看测试报告过程中能够对应用的缺陷出现过程有更加直观的认识，用户能够通过系统提供的自动化测试用例更好的复现测试报告中的缺陷，同时便于用户构建针对待测 Web 应用系统的自动化测试用例库。最终，通过直观的测试报告和便捷的复现方法，帮助用户高效地对 web 应用进行测试并反馈测试结果，降低测试成本。

1.2　国内外研究现状

由于 Web 应用测试日趋重要以及其特殊性，现有的 Web 应用测试贯穿于软件的整个生命周期。针对 Web 应用异构性、分布性、并发性以及平台无关性等特征，科研人员提出了许多 Web 应用测试方法。Daniela 等人依据软件方法主题分析指南 [6]，将这些测试方法划分为 10 个不同的主题类别，包括白盒测试、黑盒测试、变异测试、AJAX 测试、基于对话的测试、跨浏览器兼容性测试、回归测试、原子测试、其他类型的测试以及测试支持 [1]。

针对 Web 应用的白盒测试方法，较于传统软件测试，主要用于测试 Web 应用系统内部结构信息。Alshahwan 等人基于三种相关的算法实现了 SWAT 的工具，使用基于搜索软件测试方法进行 Web 应用自动测试。该测试方法显著提高了利用静态和动态分析基于搜索的技术的效率和有效性 [7]。Ozkinaci 等人提出了一种使用调和测试技术的工具 Mamoste，该技术用于动态生成测试输入来测试 ASP.NET 应用 [8]。

黑盒测试方法是在不考虑待测 Web 应用的代码结构和实现的情况下，根据系统的功能来生成测试案例。黑盒测试主要关注于使用合适的模型来指定 Web 应用程序的测试行为 [9]。Andrew 等人提出了一种基于有限状态机的黑盒测试方法，用于从 Web 应用生成测试用例。该方法是将基于有限状态机的测试生成与约束相结合，使用约束来减少有限状态机方法存在的状态空间爆炸的问题 [10]。但是使用简单的约束，依旧无法解决状态空间爆炸的问题，Andrew 等人评估了一种解决方案，使用输入约束来减少操作过渡次数，从而有效地压缩有限状态机，经过实验发现效果显著 [11]。缪淮扣等人利用基于有限状态机的测试方法构建了完整测试工具，实现了建模、转换、规约 Web 应用有限状态机的生成方法，优化了各项测试覆盖准则 [12]。

变异测试方法是采用语法结构，创建修改后的软件版本，利用测试输入使得修改后的软件与原来软件出现不同表现。修改后的版本为变异版本，如果测试输入导致变异版软件与原始行为不同，则可以认为测试杀死变异 [13]。用于 Web 应用特有的变异算子可以帮助创建对发现 Web 应用缺陷有效的变异测试

[14]。Praphamontripong 等人定义了基于 JSP 和 Java Servlet 的变异算子，使用该方法创建的变异测试能够有效地分析 Web 应用。Mirshokraie 等人提出了基于 JavaScript 的变异工具 MUTANDIS，并在许多 Web 应用上进行测试，取得了良好的测试效果 [15]。

相较于以上提到的测试方法，AJAX 测试、基于对话的测试、跨浏览器兼容性测试是 Web 应用测试特有的测试方法。AJAX 测试是基于 Web 应用特有的异步 JavaScript 和 XML 进行针对性测试，更加适应 Web 应用的特性 [16]。Marchetto 等人提出了一种基于状态的测试方法，在测试阶段考虑使用 AJAX 改变 HTML 元素，使客户端组件产生新的状态。使用基于状态的测试方法，可以发现使用以往技术无法发现的 Web 应用故障。Mesbah 等人提出了一种基于不变式的 AJAX 自动测试方法，使用抓取工具获取 AJAX 应用程序，在用户界面上模拟实际用户事件，以此来确定用户界面状态下可能发生的 AJAX 特定故障 [17]。Mesbah 等人在 AJAX 自动测试中又提出了进一步的 AJAX 研究主题 [18]，指出在 Web 应用中最佳的路径传播是捕获和重播。贺涛等人提出了一种基于 AJAX 技术的测试用例生成方法，并结合 Web 应用建模，能够有效的生成针对性的测试用例进行 Web 应用测试 [19]。

基于会话的测试方法是一种利用真实用户数据增强初始化测试套件的自动化方法 [20]。Thummalapenta 等人提出了一种业务规则驱动的自动生成测试技术。业务规则是用于描述 Web 应用的业务逻辑、访问控制以及导航属性的通用机制。该技术在测试覆盖比无方向性技术更为有效，能够覆盖 92% 的规则 [21]。Dallmeier 等人提出了一种基于会话的测试技术，并实现了测试工具 WEBMATE，系统地探索和测试 Web 应用的所有功能，用于自动 Web 应用测试生成，测试覆盖能力强 [22]。

基于会话的测试方法已经证实能够提高 Web 应用的质量，但是实际用户会话数据量非常大，为了减少用于 Web 应用测试的测试用例集的规模，huyan Wang 等人提出了一种基于用户会话的测试用例的简化方法，使用 PageRank 算法和汉明距离减少使用基尼索引排序的测试用例，测试案例的减少率达到了 85.7%[23]。武晋南等人提出了一种基于用户行为和会话的测试方法，利用 Web 日志数据进行用户行为获取以及测试用例创建，并通过实验验证了该方法在功能测试覆盖和缺陷检测方面的有效性 [24]。

随着 Web 应用和不同浏览器数量的快速增长，不同浏览器所呈现的应用内容可能会不同 [25]，跨浏览器兼容性问题变得越来越重要。跨浏览器兼容性测试方法是考量 Web 应用在不同浏览器上行为的方法。Mesbah 等人提出了现代 Web 应用的跨浏览器兼容性测试的问题，作为跨不同浏览器的 Web 应用的功能一致

性检查，并提供了一种自动化检查方法 [26]。Shauvik 等人提出了一种跨浏览器兼容性检测工具 CROSSCHECK，该方法组合了 WEBDIFF[27] 和 CROSST[26] 两种互补的方法，能够有效地检测 Web 应用的兼容性 [28]。

回归测试方法是为了确保 Web 应用在迭代更新过程中的稳定性和可靠性，避免迭代更新中的修改或者缺陷修复对应用产生不利影响 [29]。Mirshokraie 等人提出了一种针对 JavaScript 的自动化回归测试技术，通过向 JavaScript 代码注入不变的断言，来发现 Web 应用后续版本的回归错误 [30]。Animesh 等人借助 WSDL 解析来完成功能性和非功能性的 Web 应用测试，并通过识别更改来进行回归测试，同时基于自动 Web 服务变更管理工具开发了基于上述方法的测试工具，在真实项目实验中验证了其适用性。[31]。

原子测试方法是将 Oracle 机制引入到 Web 应用测试中，通过对比预期输出和实际输出来测试系统的正确性 [32]。Ran 等人提出了针对 Web 应用后端数据库测试工具 AutoDBT，该工具可用于生成 Orcale 测试，验证被测 Web 应用数据库在测试过程中是否正确更新 [33]。

除上述种类的测试方法以为，Web 应用测试技术进展还包括其他种类的测试以及测试支持方法。Karthik 等人提出了一种基于文档对象模型 DOM 测试 Web 应用健壮性的测试系统 DoDOM。DoDOM 系统能够提取 DOM 结果中的不变量用于检测影响 DOM 的错误，具有高覆盖率、低误报率的特点 [34]。Artzi 等人基于 Tarantula 算法以及语句与输出之间的映射技术，实现了针对 PHP 应用程序的缺陷定位工具，提高了缺陷定位效率 [35]。Yunxiao Zou 等人针对 Wen 应用测试提出了一种 V-DOM 覆盖率，基于服务端脚本可能产生的所有 DOM 对象，构建 V-DOM 树，该技术比动态爬取技术更加全面 [36]。彭树深等人总结了 4 类 Web 应用测试用例生成方法，包括 HTML 分析法、User-Session 分析法、Capture/Replay 法、源代码分析法，并从实现难度、覆盖率等方面阐述了 4 类方法的优缺点 [37]。

现有的 Web 应用测试技术已经取得一定的进展，也利用 Web 测试方法提出了一些 Web 应用自动化测试方法 [38]，但是现有的 Web 应用自动化测试方法缺乏完整的自动化测试流程，未提供清晰完整的测试报告。与此同时，在测试报告中提供测试重访以验证复现缺陷，对于测试报告的易于使用也十分重要。因此，本文中的系统基于上述问题，针对自动化测试流程以及清晰的测试报告进行设计与实现。

1.3　本文的主要工作内容

本文针对以上研究现状以及提供清晰、易用的测试报告的必要性，开发了Web 应用自动化测试系统报告生成服务。报告生成服务的主要功能是完善整个Web 应用自动测试的流程以及测试报告的生成和展示。服务监控测试执行引擎收集相关测试数据，测试结束后利用单页面分析结果生成完成的测试报告，进行统计评估并展示，为便于测试复现，报告生成服务为用户生成相关测试流程的自动化测试用例。

报告生成服务基于 Web 应用自动化测试流程，主要从两个方面来帮助用户实现 Web 应用的自动测试，向用户提供缺陷分析结果、质量评估结果组成的测试报告以及相关页面、缺陷的自主复现方法。用户通过 Web 应用自动化测试系统输入待测应用相关信息开始测试流程，报告生成服务通过监控自动化测试执行引擎的运行情况，获取当页面的执行过程信息，然后通过整合和统计分析结果，生成并展示完整的可视化自动测试报告以及软件质量评估报告。用户可以通过服务获取页面以及缺陷的复现方法，服务通过监控执行引擎操作过程，生成相关页面、缺陷的自动化测试用例。用户通过下载用户提供的自动化测试用例可以对测试流程以及缺陷进行自主复现，使用更加便捷。

在系统开发过程中，为保证开发效率和稳定性，在技术实现上采用主流框架，使用 Spring Boot 作为服务端框架、jQuery 作为前端框架。同时为保证系统的稳定性和性能，采用 RabbitMQ 作为消息中间件、Redis 作为系统高速缓存。在测试阶段，为了保证报告生成服务的可用性和可靠性，本文对其进行了功能测试。同时，为保障 Web 应用自动化测试系统的测试能力，本文通过真实的使用场景对系统进行了实验评估。

报告生成服务保障执行服务高并发场景下后续流程的稳定性，以及大量日志处理的性能。报告生成服务结果向用户提供了清晰完整的测试报告和简单易用的复现方法，保障用户能够通过系统获取 Web 应用的测试结果，以及可以通过系统自主复现自动化测试检测出的缺陷，提高测试效率和测试可信度。

1.4　本文的组织结构

本文共有六个章节，具体组织结构如下：

第一章引言，介绍了 Web 应用测试的相关背景和研究现状，说明 Web 应用测试在其软件生命周期的重要性，以及减少人工干预能够节约大量测试成本，在阐明自动化进行 Web 应用测试的重要性。同时阐明了现有的测试工具在测试报告方面的不足，从而阐明本文报告生成服务的必要性。

第二章相关技术，介绍了项目中使用的 Spring Boot、Crawljax、消息中间件 RabbitMQ、缓存中间件 Redis、Selenium、jQuery 的概述和使用。

第三章 Web 应用自动测试系统的需求分析与设计，概述了项目整体情况，根据项目背景和预期目标，分析 Web 应用自动测试系统的功能性需求和非功能性需求，然后描述了报告生成服务系统的整体架构、视图以及模块划分。最后，根据划分执行引擎监控模块、测试报告模块、自动化测试用例生成模块以及软件质量评估报告模块，对每一个模块依次进行详细的模块设计。

第四章 Web 应用自动测试系统报告生成服务的实现，在第三章模块划分和详细设计的基础上，使用顺序图、关键代码和运行页面截图，对各个模块的实现细节进行阐述。

第五章 Web 应用自动测试系统报告生成服务的测试和实验评估，首先介绍了系统的测试环境以及系统待测的 Web 应用的情况，然后对报告生成服务的流程以及各个模块重要的功能点和接口进行测试，最后对系统测试结果进行分析和评估。

第六章总结与展望，首先先对本文的工作进行了总结，然后展望了 Web 应用自动测试系统报告生成服务未来的改进。

第二章　相关技术

2.1 Spring Boot

2.1.1 Spring Boot 简述

Spring Boot 是由 Pivotal 团队自 2013 起开发并迭代的轻量级的全新开源框架。其设计目的是为了简化 Spring 项目的搭建、开发、运行和部署的流程，更为了是帮助开发人员摆脱了原生 Spring 繁重的样板化配置 [39]。因此，近几年 Spring Boot 已经超越 Spring 成为工业界最受欢迎的 Java 开发框架。

Spring Boot 具备以下几个非常优秀的特性。首先，Spring Boot 采用内置的 Web 容器 Tomcat，相比于原生的 Spring MVC 我们部署时只需要项目编译打包生成的 jar 包，而无需配置 Web 容器的环境，这非常的方便于软件系统的迭代发布，大大减少部署的成本。其次，Spring Boot 为 Spring 平台和很多第三方依赖库提供了开箱即用的设置和依赖支持，我们可以使用 Spring Boot 提供的约定优先配置，也可以通过自定义配置文件或者在启动命令中自己设置参数值来达到想要的配置效果。同时 Spring Boot 系列的各种 starter 启动器也简化了很多第三方依赖的 maven 配置，如 Redis、RabbitMQ 等，减轻了开发者配置第三方依赖的工作。然后，对于开发者来讲，Spring Boot 提供了即时编译、热部署等工具包，更提供了安全监测、健康检查等内置的功能，使开发的效率得到提升。最后也是最关键的，近几年来微服务架构成为工业界非常火热的名词，越来越多的公司采用微服务架构来设计自己的项目，而 Spring Boot 支持 Spring Cloud 等微服务框架，其本身的特性也是针对于为微服务架构服务而设计的 [40]。

2.1.2 使用 Spring Boot 的优势

本文的所构建的系统使用微服务架构，其子系统需要使用轻量、对服务化架构支持度高的开发框架，Spring Boot 的特性非常适合各个子系统的独立开发。于此同时系统开发过程中需要用到诸如 Redis、RabbitMQ 等，使用 Spring Boot 系列的的各种 starter 启动器大大简化配置，减轻了减轻了开发者配置第三方依赖的工作。因此本文选择 Spring Boot 来构建后端微服务子系统。

2.2 Crawljax

2.2.1 Crawljax 简述

Crawljax 是一个开源的用于 Web 应用自动化抓取和测试的工具。Crawljax 使用 Java 语言开发，通过使用事件驱动的动态引擎来探索任何基于 JavaSripit 的 Ajax Web 应用程序 [41]。Crawljax 能够生成动态的 DOM 状态，以及 DOM 状态间的转化事件。在搜索 Web 应用程序的过程中，Crawljax 使用深度优先算法。

2.2.2 Crawljax Plugin

表 2.1: Crawljax Plugin

接口类	触发时间	应用场景
PreCrawlingPlugin	在加载 URL 之前	进入登录 URL 并登录，加载自定义代理配置以拦截请求或响应
OnNewStatePlugin	当一个新的状态被发现并爬取后	获取截图、网页 DOM 情况等
OnUrlLoadPlugin	当初始的 URL 被加载或者重新加载后	重置后端状态
OnRevisitStatePlugin	当一个状态被重新访问	分析测试流程
PreStateCrawlingPlugin	在一个新的状态被爬取前	记录候选元素
PostCrawlingPlugin	在爬去流程结束后	生成测试用例或者测试流程图

Crawljax Plugin 是 Crawljax 的功能扩展，使用 Crawljax Plugin 可以丰富 Crajlax 测试流程。表2.1列举了 Crawljax Plugin 的接口类，主要包括 PreCrawlingPlugin、OnNewStatePlugin、PostCrawlingPlugin、OnUrlLoadPlugin、OnRevisitStatePlugin、PreStateCrawlingPlugin。基于这些 Plugin，系统可以对测试流程进行开发。

在本文的系统中，基于系统业务流程，引入 Crawljax Plugin 中 PreCrawlingPlugin、OnNewStatePlugin 以及 PostCrawlingPlugin 三个接口对系统测试执行过程进行监控，分别实现初始化、运行过程、测试完成三个状态的监控。

2.3 RabbitMQ

2.3.1 RabbitMQ 简述

RabbitMQ 是有 Rabbit 科技有限公司开发，并在 2013 年正式成为 GoPivotal 的一部分。RabbitMQ 本质上是基于 AMQP 协议实现的一个开源面向消息中间件。AMQP 全称为 Advanced Message Queuing Protocol，即高级消息队列协议，

是针对面向消息中间件设计的一个应用层开发标准协议 [42]。其主要是面向消息、路由、队列、安全、可靠性。面向消息中间件的主要作用是组件、系统之间的解耦，消息的发送者和消息使用者之间的解耦 [43]。RabbitMQ 就是 AMQP 的一个开源实现，服务端有 Erlang 语言编写，支持 Python、Java、PHP、Ruby、ActionScript、XMPP、.Net、STOMP 等多种客户端，可用于分布式系统中存储转发消息。RabbitMQ 也支持多种操作系统，如 Linux、Windows、MacOS、FreeBSD、TRU64 等。

在使用 RabbitMQ 的过程中，主要使用的 API 对象包括 ConnectionFactory、Connection、Channel，其中前两个用于建立链接，而后面一个用于业务逻辑操作。ConnectionFactory 是创建 Connection 的工厂，主要包含一些配置信息。Connection 是 RabbitMQ 的 Socket 链接主体，封装了 socket 协议的部分逻辑。Channel 是主要的业务操作对象，主要作用包括定义 Queue 和 Exchange、绑定消息队列、发布信息等，其中的 Queue 和 Exchange 是 RabbitMQ 中的主要内部对象。Queue 是消息队列主体，用于存储消息。RabbitMQ 中所有的消息都存储在 Queue 中，消息生产者会将消息投递到 Queue 中，而消息使用者会从 Queue 中获取消息，然后对消息进行消费使用。

多个消息使用者可以同时订阅某个特定的 Queue，在这种情况下，消息会根据订阅顺序发布给这些使用者。Exchange 的存在则大大丰富了 RabbitMQ 的功能和使用场景，Exchange 主要定义了 Queue 之间的消息共享方式。常用的 Exchange 类型主要有 fanout、direct、topic、header 这四种。fanout 类型的 Exchange，会拷贝消息并将消息转发到所有绑定在 fanout Exchange 上的 Queue，主要用于快速转发消息。direct 类型的 Exchange，会把消息转发到那些 routing key 完全匹配的 Queue 中，是 RabbitMQ Broker 中的默认 Exchange。topic 类型的 Exchange 是基于 direct 类型的扩展，但是具有不同 routing key 的匹配规则。headers 类型的 Exchange 不依赖 routing key 的匹配规则，而是根据发送消息内容中的 Header 属性进行匹配，匹配规则更加强大。

2.3.2　使用 RabbitMQ 的优势

在面向消息中间件的选型中，RabbitMQ 相较于其他面向消息中间件，具有四个方面的优势。其一，RabbitMQ 支持消息持久化，能够保证系统容灾和稳定性 [44]，RabbitMQ 可以在系统收到一些因素影响而崩溃的情况下，保证信息不丢失。其二、RabbitMQ 具有良好的高并发支持，RabbitMQ 基于 Erlang 语言实现，而该语言具备高并发、高可用的特性 [45]。其三，在技术实现层面，RabbitMQ 能提供良好的支持，它具有灵活的路由、高可用的队列、可视化管理工具等。其

四，RabbitMQ 拥有较高的社区活跃度。

在本文的系统中，RabbitMQ 主要适用于组件间解耦和支持高并发的场景。系统中的执行引擎在执行 Web 应用自动化测试的过程中，会实时产生大量的分析请求，而分析服务模块和执行引擎模块是功能独立的两个系统组件，通过引入面向消息的中间件 RabbitMQ，可以使这两个组件间完成解耦。同时，为了提高测试效率，执行引擎采用并发的模型对 Web 应用进行测试，使用 RabbitMQ 能在并发的场景下使得系统能够稳定运行。

2.4 Redis

2.4.1 Redis 简述

Redis 是一个由 Salvatore Sanfilippo 开发，基于 key-value 键值对的缓存数据库。Redis 全称为是 Remote Dictionary Server ，是一个作用于内存的数据存储服务，它是由 ANSI C 语言编写，支持网络，遵守 BSD 协议，并提供多种语言的 API。Redis 能支持多种类型的数据结构，key-value 键值对中的值（value），可以是字符串（String）、列表（List）、哈希（Hash）、集合 (sets) 和排序集合（Zset）多种数据类型 [46] 。作为一个高性能的缓存数据库，Redis 具有很多的特点，首先和一般数据库相比，Redis 也支持数据的持久化，可以将运行过程中产生的缓存数据保存在磁盘中，重启的时候可以通过加载的方式继续使用。其次，Redis 存储功能强大，不仅仅局限于 key-value 类型的数据。最后，Redis 具有容灾能力，支持 master-value 模式的数据备份。

Redis 存储使用了两种不同的文件格式，全量数据和增量请求。全量数据这种文件格式是把数据写入磁盘；增量请求文件则是把数据序列转换为操作请求，操作请求可用于读取文件进行重放得到数据，支持的序列化操作包括 SET、RPUSH、SADD、ZADD 四类。

由于 Redis 上述特点，使其拥有了一些显著优点。作为一个高性能数据库，Redis 在读写性能上的表现非常出色，读取速度可以达到每秒 1100000 次，写入速度可以达到每秒 81000 次。支持丰富的数据类型，使其能够应用的场景十分广泛。相比其他的 key-value 型存储结构，Redis 中所有操作都是原子型的，便于监控操作结果以及回滚。

2.4.2 使用 Redis 的优势

在本文的系统中，使用 Redis 来作为缓存中间件，主要原因是 Redis 的高性能以及能够支持多种数据结构的特性和优点 [47]。在 Web 应用测试引擎执行的过程中，会产生大量需要缓存的信息，如日志、路径等，于此同时由于引擎支持

并发执行，测试结果分析也同时进行，缓存数据库读写压力大。Redis 高性能的优势，能够系统在高负荷的读写中高效地运行。Web 应用执行引擎在执行过程中产生的数据类型多样，包含字符串、列表等多种数据类型，使用 Redis 能够良好地满足系统对存储数据结构的要求。此为，由于系统存在重启的可能性，Redis 数据持久化的特性，也为系统能够长期稳定运行提供了帮助。

2.5 Selenium

2.5.1 Selenium 简介

Selenium 是目前在 Web 应用测试方面使用广泛的开源自动化测试套件。Selenium 是由杰森·哈金斯（Jason Huggins）在 2004 年开发第一版，开发初期 Selenium 被用为 ThourgtWorks 的内部工具。Selenium 具有良好的兼容性。Selenium 支持支持不同的操作系统、不同的浏览器、支持不同的编程语言。操作系统方面，Selenium 能够轻松的运行部署在 MacOS、Windows、Linux 等主流的操作系统，并且能够兼容这些操作系统的大部分发行版本；浏览器方面，Selenium 能够支持运行的浏览器包括 Mozilla FireFox、Google Chrome、Internet Explore 等 [48]；编程语言方面，Selenium 可以通过每种不同的编程语言的驱动程序来来操作，能够支持语言包括 C#，Java，Perl，PHP，Python 和 Ruby，基于这些语言，通过相应的 Selenium 驱动程序，可以直接操作浏览器执行各种操作 [49]。

Selenium 除了上文提到的具有良好的兼容性以外，还具有以下一些功能特性。Selenium 作为一个开源的测试框架，可以同 Maven、Ant 等框架集成，减轻开发者配置工作。与其他自动化测试工具，如 HP QTP 相比，Selenium 的资源消耗较低。WebDriver 的引入，使得 Selenium 的使用更加便捷，Selenium 测试脚本可以直接与被操作浏览器进行交互。Web Element 类的设计，使得 Selenium 命令使用更加清晰，更易于这些命令的理解和实现。

2.5.2 Selenium 使用

本文在进行自动化测试用例生成的过程中，基于 Selenium 测试脚本的规格，对自动化测试用例进行编写，使得生成的 Web 应用自动化测试用例能够通过 Selenium 操作浏览器从而复现测试流程。生成的自动化测试脚本的过程中，页面元素 Web Element 的定位，使用的 Selenium 中 xpath、id、name 等，涉及的页面元素 Web Element 包括文本框、单选框、iframe、下拉选择框等。通过输入的 Web Driver 的不同，生成的自动化测试可以调用远程的、以及本地的浏览器来执行自动化测试用例。

2.6　jQuery

2.6.1　jQuery 简述

jQuery 是由 John Resig 等人于 2006 年发布的 JavaScript 框架，并在同年发布了第一个稳定版，支持 CSS、AJAX 和事件机制。jQuey 的设计理念是简洁、高效，它封装了 JavaScript 高频的功能代码，同时提供了一种简单的 JavaScripit 设计模式，优化 HTML 文档操作、AJAX 交互和事件处理机制等 [50]。jQuery 库的功能包括 HTML 元素选取和操作、HTML 事件函数、HTML DOM 遍历和修改、CSS 操作、JavaScript 特效和动画、AJAX。虽然目前网络上有大量开源的 JavaScript 框架, 但 jQuery 是目前使用最广泛、最流行的框架，很多大公司例如:Google,Microsoft，IBM，Netflix，都在使用 jQuery 进行 JavaScript 代码的开发。

2.6.2　使用 jQuery 的优势

基于 jQuery 简洁、高效的设计理念，jQuery 具有很多的优点。jQuery 十分轻量级，压缩文件大小不到 30k；jQuery 具有强大的 CSS 选择器，包括 CSS1 到 CSS3 几乎所有选择器, 以及 jQuery 自身的选择器; jQuery 封装了高频的 DOM 操作，是开发者编写便捷；jQuery 具有可靠的事件处理机制，它参考了 JavaScript 专家 Dean Edwards 的事件处理函数，事件绑定相当可靠；jQuery 具有完善的 AJAX，它将所有的 AJAX 操作封装到 $.ajax() 里，使开发者能够专心处理业务逻辑；jQuery 不污染顶级变量，该特性使 jQuery 和其他 JavaScript 库共存时不会产生冲突；jQuery 能够在 IE6.0+、Safari2.+、Opera9.0+ 和 FF 2+ 下正常运行，具有良好的浏览器兼容性；jQuery 支持链式操作，编写在同一个 jQuery 对象上的动作时无需重复获取对象；jQuery 具有隐式迭代的特性，方法都被设计成自动操作的对象集合，减少代码中的循环结构，大幅度的减少代码量；jQuery 实现了行为层与结构层的分离，开发者可以直接通过选择器选择元素并给其添加事件；jQuery 提供了大量的插件扩展以及丰富的文档支持，包括中文 API；jQuery 开源，社区活跃度高。

在本文的系统中，jQuery 主要用做 Web 应用测试报告的可视化的开发。使用 jQuery 进行可视化开发，可以很大程度上增加开发效率。基于 jQuery 强大的选择器，可以高效完成页面元素的选取。基于 jQuery 可靠的事件处理机制，完善和增强测试报告的交互性。基于 jQuery 良好的兼容性，可以保证报告能够在不同的浏览器环境中展示。基于封装的高频 DOM 操作、链式操作，使得开发便捷高效。

2.7　本章小结

本章节介绍了项目中所用到的相关工具、技术和框架。Spring Boot 用于 Web 应用自动化测试系统后台主要部分的构建和集成开发。Crawljax Plugin 用于执行引擎中分析所需的相关数据的采集。面向消息中间件 RabbitMQ 用于 Web 应用测试执行引擎和分析服务组件之间的解耦，提供消息传输服务。缓存数据库 Redis 用于系统组件间的数据共享和传输。Selenium 用于 Web 应用自动化测试用例的生成。jQuery 用于系统测试报告的可视化开发。

第三章　报告生成服务的需求分析与设计

3.1　系统整体概述

报告生成服务是 Web 应用自动化测试系统的组成部分，系统主要为用户提供 Web 应用的自动化测试服务。用户可以通过系统发布 Web 自动化测试任务，主要流程包括发布测试任务、查看任务情况、查看测试报告。系统用户可以通过系统向 Web 应用自动化测试服务端发布测试任务。测试服务端接收到测试任务后，进行 Web 应用的自动化测试。服务端完成测试后，对待测的 Web 应用会生成相应的测试测试报告。系统的主要目标是针对 Web 应用系统测试存在的问题和难点，通过输入待测 Web 应用的 URL，模拟人工测试的方法，完成 Web 应用自动化测试，并生成详细的测试报告，从而帮助用户评估 Web 应用。通过自动化测试的方法，系统可以大大减少 Web 应用测试的成本，并且大大提升 Web 应用测试的效率。

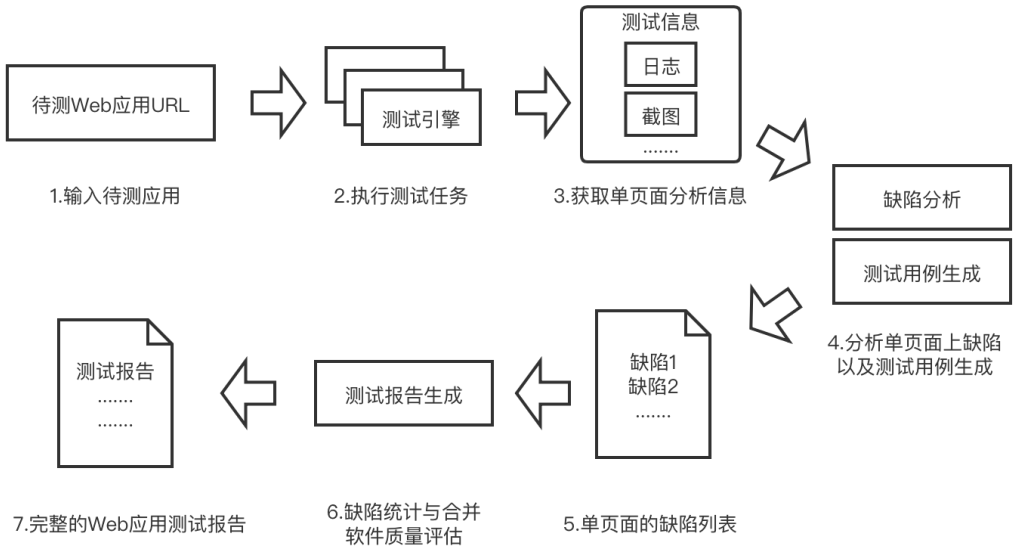

图 3.1: Web 应用自动化测试流程图

如图3.1所示是 Web 应用自动化测试系统进行测试的工作流程。用户可以通过登陆系统并获取进行 Web 应用自动化测试的权限。在获取测试权限后用户就可以进行 Web 应用自动化测试的流程。首先，用户通过系统上传待测 Web 应用的 URL，URL 信息是 Web 应用自动化测试服务端访问待测 Web 应用的重要信

息。用户也可以配置测试过程中的参数信息，具体配置信息包括环境配置信息、测试任务基本信等。在设置完上述测试任务的信息后，用户可以向 Web 应用自动化测试服务端发布测试任务。服务端在接收到测试任务后，会启动一些测试执行引擎对待测的 Web 应用进行自动化测试。在自动化测试过程中，测试执行引擎每对 Web 应用进行一次操作，就会产生一次新的页面状态。当新的页面状态产生后，测试执行引擎中的监控器就会记录当前页面状态的一些信息，包括浏览器日志、操作路径等。通过这些信息，Web 应用自动化测试服务端会调用分析服务，以及自动化测试用例生成服务，生成单页面测试分析结果。在所有测试执行引擎结束测试任务后，Web 应用自动化测试服务端，就会调用测试报告整合服务，合并并统计所有单页面的分析结果和自动化测试用例生成完整的测试报告，同时收集测试流程图以及软件质量多维评估结果进行软件质量评估展示，作为测试报告的补充。Web 应用自动化测试服务端生成完整的测试报告后，用户可以通过系统访问该测试报告。

表 3.1: 用例描述

需求编号	**需求名称**	**需求描述**	**优先级**
R1	上传待测应用	系统用户可以通过系统上传待测 Web 应用的 URL	高
R2	配置测试参数	系统用户可以通过系统配置待测 Web 应用在测试过程中的参数，如并发执行数量、测试运行环境信息等	高
R3	发布测试任务	系统用户可以通过系统发布一个待测 Web 应用的测试任务	高
R4	查看测试结果	系统用户可以通过系统查看是否成功发布一项测试任务，并且用户可以查看测试是否执行完成，即是否生成测试报告	中
R5	查看测试报告	系统用户可以通过系统提供的测试报告链接，访问发布的测试任务的测试报告情况，测试报告中包含测试任务执行的基本信息，缺陷列表，以及相应的一些统计信息	高
R6	查看缺陷详情	系统用户可以在系统测试报告中的缺陷列表访问一个缺陷的详细情况，包括该缺陷的描述、所属的网页、缺陷出现的位置等	高
R7	查看缺陷复现路径	在展示缺陷详情的同时，系统用户可以通过系统获取该缺陷的执行路径，查看该缺陷的复现方法	高
R7	查看软件质量多维评估	系统用户可以通过系统查看被测 Web 应用的多维质量评估结果	高

3.2　系统需求分析

3.2.1　功能性需求

本系统的主要的功能是通过待测 Web 应用的 URL 输入，以及测试相关的一些参数配置完成待测 Web 应用的自动化测试任务。系统的功能需求主要包括上传待测 Web 应用 URL、配置测试参数、发布测试任务、查看测试结果、查看测试报告、查看缺陷详情、查看缺陷复现路径、查看软件质量多维评估。详细的功能需求描述如表3.1所示。

3.2.2　非功能性需求

本系统的非功能需求主要包括以可用性、易用性、性能、可扩展性这四个方面，具体的非功能需求描述如表3.2所示。系统需要具有一定容灾能力，具有高可用性，减少系统出现崩溃的情况，在出现崩溃后也能够快速恢复。用户使用系统能够高效、便捷，无需大量的指导和用户手册即可快速使用系统进行 Web 应用自动化测试。性能方面，系统能承受一定的压力，同时响应速度快。系统做好接口抽象和服务之间的解耦，能够快速应对变化以及迭代更新，并具备服务端水平扩展能力。

表 3.2: 用例描述

非功能需求	需求描述
可用性	系统各个服务之间实现解耦以及异常情况处理，保障系统具有高可用性，当一个 Web 服务宕机时不会导致整个系统崩溃；系统崩溃后，能够迅速恢复
易用性	系统能够向用户提供便捷的使用方式，包括提供快捷的系统交互方式，以及简洁明了、解释性强的前端展示页面
性能	系统前端具有高相应速度，界面等待时间短，提供良好的用户体验。同时，系统支持并发使用，服务端接口能够承受一定的压力
可扩展性	系统面向迭代以及修改，能够高效的应对自身业务以及环境升级带来的变化。系统内部报告生成服务各个模块、功能做好解耦以及抽象接口，方便功能改进、运行环境升级

3.2.3　用例描述

根据上文所示的 Web 应用自动化测试系统的功能需求，得到如图3.2所示的系统用例图。系统用户通过系统上传待测 Web 应用的 URL，以及配置相关的执行参数，创建一项测试任务。用户在创建完测试任务后，通过系统发布 Web 应用自动化测试任务。在发布测试任务后，用户可以查看是否成功的发布测试任务，以及测试任务的完成情况，即待测 Web 应用的测试报告是否生成。在 Web

应用自动化测试系统生成待测 Web 应用的测试报告后，用户可以通过系统提供的报告链接访问待测 Web 应用的测试报告。在测试报告中，用户可以查看待测 Web 应用的测试报告总览、测试中发现的网页缺陷列表以及一些相关测试指标的统计。用户还可以通过系统展示的缺陷列表查看某一项缺陷的详细信息，详细信息包括缺陷描述、缺陷资源以及复现缺陷的测试路径等。生成的测试报告还包括软件质量评估报告，用户可以通过测试报告中的链接访问通过测试生成的软件质量多维评估结果，评估结果包含多维评估雷达图，以及功能、性能、健壮性、兼容性、稳定性五个评估维度的详细评估报告。

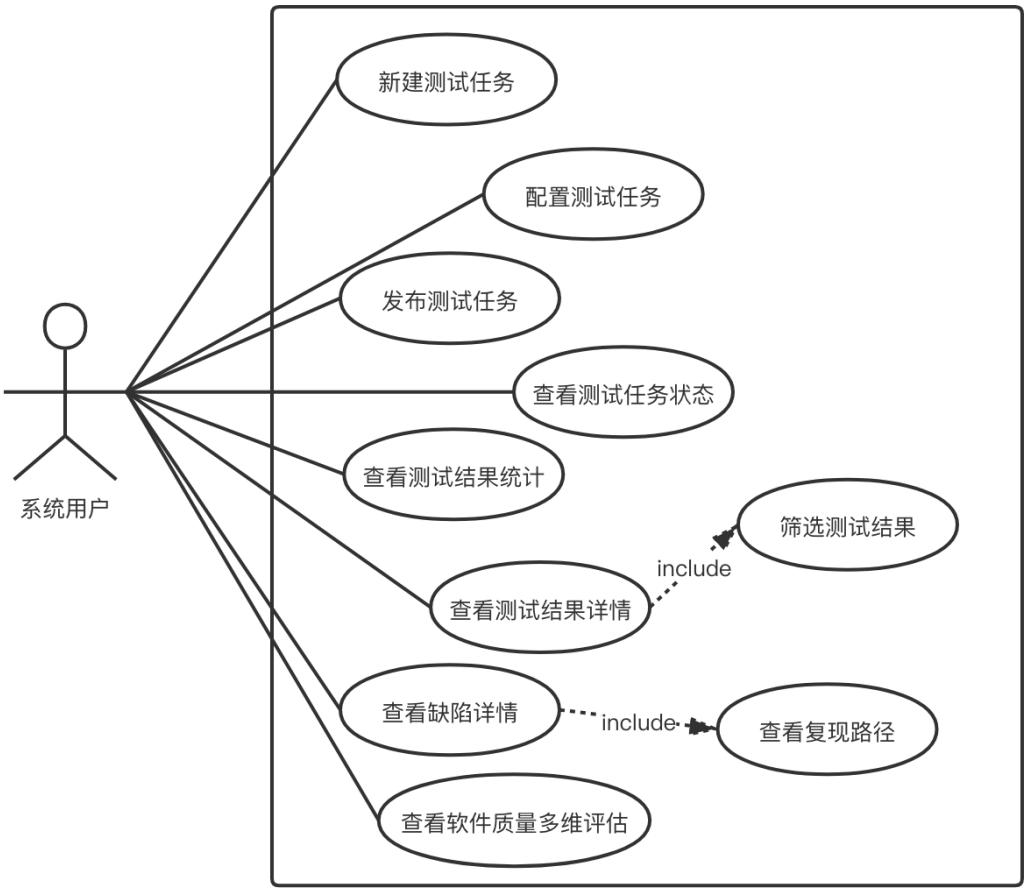

图 3.2: Web 应用自动化测试系统用例图

表3.3为 Web 应用自动化测试系统用例和系统功能需求的对应关系表，用例编号和名称与下文用例描述一致，对应需求编号可以参考表3.1所示。以下按照表3.3中的顺序，对系统中的用例详细描述。

表 3.3: 系统用例表

用例编号	用例名称	对应需求编号
C1	新建测试任务	R1
C2	配置测试任务	R2
C3	发布测试任务	R3
C4	查看测试任务结果	R4
C5	查看测试结果统计	R5
C6	查看测试结果详情	R5
C7	查看缺陷详情	R6、R7
C8	查看软件质量多维评估	R8

表3.4是新建测试任务的用例描述。系统用户通过登陆系统并获取创建测试任务的权限，可以向系统发起新建测试任务的请求，系统响应用户的请求并返回新建测试任务的详细配置表。如系统用户未登陆或者系统用户不具备新建测试任务的权限，用户无法从系统获取新建测试用例的服务。

表 3.4: 新建测试任务用例描述

描述项	说明
用例名称	新建测试任务
参与者	系统用户
用例描述	系统用户通过系统选择新建测试任务
优先级	高
前置条件	系统用户登陆系统并具有创建测试任务的权限
交互流程	1. 用户登陆 Web 应用自动化测试系统 2. 用户进入 Web 应用自动化测试服务界面 3. 用户选择新建测试任务 4. 系统对用户的新建请求作出反馈，展示新建测试任务的信息表，待用户填写
后置条件	无

配置测试任务是本系统用户完成 Web 应用测试流程中的主要部分，完成了在执行测试过程中的参数配置。通过配置测试任务，用户可以上传待测 Web 应用 URL 信息，系统在测试待测 Web 应用的过程中会使用这个信息访问待测 Web 应用，并进行 Web 应用自动化测试。同时，在配置 Web 应用测试任务的过程中，用户也可以选择运行自动化测试的配置信息，包括测试过程中使用的环境配置信息、测试过程中使用的执行引擎数量、浏览器配置信息、最大网页页面状态数、最长任务执行时间、最大探索深度等。配置测试任务描述的具体用例描述，如表3.5所示。

表 3.5: 配置测试任务用例描述

描述项	**说明**
用例名称	配置测试任务
参与者	系统用户
用例描述	系统用户通过系统提供的配置参数列表，配置测试任务的参数
优先级	高
前置条件	系统用户登陆系统并具有创建测试任务的权限，用户进入新建测试任务的流程
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择新建 Web 应用自动化测试任务 3. 用户根据页面显示填写 Web 应用 URL 4. 用户可以填写 Web 应用自动化测试过程中的配置信息 5. 系统前端页面显示 Web 应用自动化测试配置信息
后置条件	无

表 3.6: 发布测试任务用例描述

描述项	**说明**
用例名称	发布测试任务
参与者	系统用户
用例描述	系统用户通过系统向 Web 应用自动化测试系统服务端发布测试任务
优先级	高
前置条件	系统用户登陆系统并且具有创建测试任务的权限，按照要求创建和配置测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择新建并配置 Web 应用自动化测试任务 3. 用户选择发布 Web 应用自动化测试任务 4. 系统前端向 Web 应用自动化测试服务端发布 Web 应用自动化测试任务
后置条件	无

发布测试任务是本系统用户与 Web 应用自动化测试服务服务端进行交互的重要组成部分。用户完成 Web 应用自动化测试任务的新建和配置后，通过系统前端向系统服务端发起 Web 应用自动化测试任务。发布测试任务的具体用例描述如表3.6所示。

查看测试任务状态的用例描述如表3.7所示，本系统用户在系统中成功发布 Web 应用自动化测试任务后，可以在系统中查看其发布的任务是否运行结束。用户通过进入本系统的测试报告页面，可以查看发布的 Web 应用自动化测试任务的测试报告是否生成。测试报告的生成表示发布的测试任务已经成功运行完成。查看测试任务状态是用户在发布 Web 应用自动化测试任务和查看任务结果两个

表 3.7: 查看测试任务状态用例描述

描述项	**说明**
用例名称	查看测试任务状态
参与者	系统用户
用例描述	系统用户通过系统查看其发布的 Web 应用自动化测试是否完成
优先级	中
前置条件	系统用户登陆系统并成功发布了 Web 应用自动化测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择查看 Web 应用自动化测试任务报告 3. 用户查看系统是否生成 Web 应用自动化测试报告
后置条件	无

本系统的主要流程之间的中间流程，用户通过完成此用例后，可以进入查看测试任务结果的流程。

表 3.8: 查看测试结果统计用例描述

描述项	**说明**
用例名称	查看测试结果统计
参与者	系统用户
用例描述	系统用户通过系统访问 Web 应用自动化测试报告
优先级	高
前置条件	系统用户登陆系统并成功完成 Web 应用自动化测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择查看测试报告 3. 通过系统访问生成的 Web 应用自动化测试报告 4. 进入测试报告首页 5. 系统前端页面显示 Web 应用自动化测试报告统计信息
后置条件	无

查看测试结果统计是用户查看测试结果流程中的重要部分，用户通过系统查看已运行完成的 Web 应用自动化测试报告，报告模块会向用户展示 Web 应用自动化测试报告的一些统计信息，包括待测的 Web 应用基本信息、任务执行的时间、测试的页面数量、测试发现的缺陷数量以及基于缺陷类型、缺陷级别、环境信息的统计图表。该用例是用户通过系统对生成的 Web 应用自动化测试报告的总览，为用户提供测试报告的概要信息。查看测试结果统计具体用例描述如表3.8所示。

查看测试结果详情用例是向用户展示在 Web 应用自动化测试过程中，本系统获取的所有缺陷列表。系统除了展示测试过程中获取的所有缺陷列表以外，系统还向用户提供缺陷的筛选功能，包括缺陷种类、缺陷级别和操作系统的筛选。

表 3.9: 查看测试结果详情用例描述

描述项	**说明**
用例名称	查看测试结果详情
参与者	系统用户
用例描述	系统用户通过系统生成的 Web 应用自动化测试报告，查看详细的测试发现缺陷列表
优先级	高
前置条件	系统用户登陆系统并成功完成 Web 应用自动化测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择查看测试报告 3. 通过系统访问生成的 Web 应用自动化测试报告 4. 进入测试报告缺陷列表 5. 系统前端页面显示 Web 应用自动化测试报告详细的缺陷列表
后置条件	无

用户可以通过系统从所有的缺陷列表中获取感兴趣的缺陷，从而使操作流程变得更加便捷。查看测试结果详情的用例描述如表3.9所示。

表 3.10: 查看缺陷详情用例描述

描述项	**说明**
用例名称	查看缺陷详情
参与者	系统用户
用例描述	系统用户通过系统生成的 Web 应用自动化测试报告，查看一个缺陷的详细描述
优先级	高
前置条件	系统用户登陆系统并成功完成 Web 应用自动化测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择查看测试报告 3. 通过系统访问生成的 Web 应用自动化测试报告 4. 进入测试报告缺陷列表并选择进入一个缺陷 5. 系统前端页面显示缺陷的详细信息
后置条件	无

查看缺陷详情用例是用户查看测试结果的重要组成部分，系统向用户展示在测试中发现的缺陷的详细信息。系统除了向用户展示缺陷发现的网页页面、缺陷定位的资源、缺陷描述等基本信息以外，还向用户展示缺陷出现时测试引擎的执行路径，方便用户能够自主地复现这个缺陷。系统还提供了缺陷复现测试路径相应的 Selenium 测试用例，使得系统用户更加便于复现缺陷。这个用例向用户展示了缺陷的详细信息，同时帮助用户自主复现缺陷。查看缺陷详情的用例描述如表3.10所示。

表 3.11: 查看软件质量多维评估用例描述

描述项	说明
用例名称	查看软件质量多维评估
参与者	系统用户
用例描述	系统用户通过系统查看被测 Web 应用的软件质量评估结果
优先级	高
前置条件	系统用户登陆系统并成功完成 Web 应用自动化测试任务
交互流程	1. 用户进入 Web 应用自动化测试服务界面 2. 用户在系统中选择查看测试报告 3. 通过系统访问生成的 Web 应用自动化测试报告 4. 进入测试报告首页 5. 进入软件质量评估报告
后置条件	无

查看软件质量多维评估具体用例描述如表3.11所示。查看软件质量多维评估是系统向用户展示软件质量评估报告的部分，该报告是上述测试报告的一个重要补充，系统除了向用户展示被测 Web 应用存在的缺陷之外，还根据测试过程获取的应用的其他信息对 Web 应用的功能、性能、健壮性、兼容性、稳定性进行评估，更全面地向用户展现被测 Web 应用的测试结果。

3.3　系统架构和模块设计

3.3.1　系统架构设计

图3.3是 Web 应用自动化测试系统的整体架构图。用户通过客户端首先访问慕测平台（以下简称 MT 平台），通过链接访问 Web 应用测试服务。MT 平台输入上线的待测网站网址，以及相关的测试需求，在平台上提交测试任务，对 Web 自动化测试服务发起任务。

Web 自动化测试服务服务端以微服务的形式单独部署，独立提供 Web 应用自动化服务。服务端的主要部分使用 Spring Boot 框架开发，该框架以其优越的特性提供了丰富的业务功能，同时提供集快速集成其他服务组件的能力。服务端主要分为执行服务、报告生成服务以及分析服务，报告生成服务报包括执行引擎监控模块、测试报告模块、自动化测试用例生成模块以及软件质量评估报告模块。其中，执行服务接收来自 MT 测试平台的请求，创建测试任务并启动测试执行引擎进行 Web 应用自动化测试。执行引擎监控模块通过引入执行引擎插件的形式，执行引擎插件使用 Crawljax Plugin 实现。执行引擎插件的信息一份存储到 MongoDB 中作为持久化部分，另一份存储到 Redis 高速缓存中，以备后续模

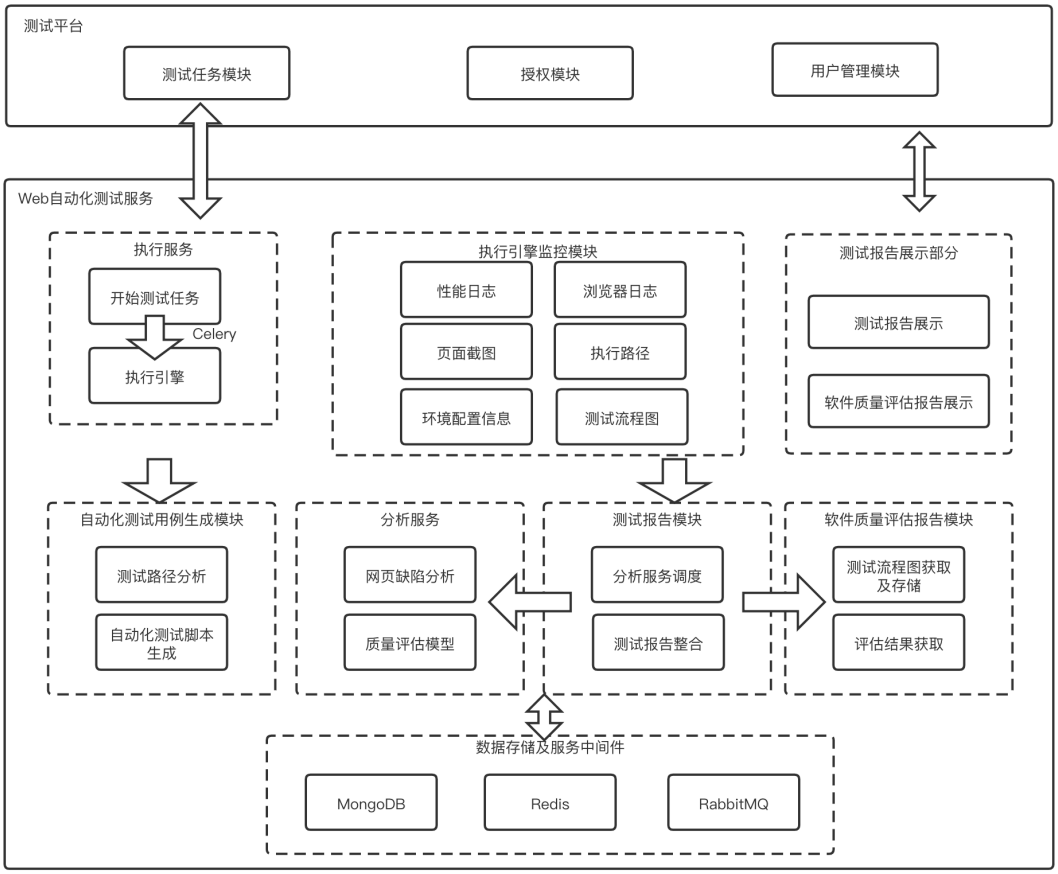

图 3.3: Web 应用自动化测试系统整体架构图

块快速使用。由于执行引擎是多例执行，插件依赖于执行引擎也是多例的，为保证系统稳定性和扩展性，系统使用 RabbitMQ 作为消息中间件。自动化测试用例生成模块，引入 Selenium 的语法，高效地生成自动化测试用例，即 Selenium 测试用例。测试报告模块也利用 Redis 保存和使用分析服务的结果。软件质量评估报告模块利用 Redis 从执行引擎中获取测试执行流程图并将其存储到 MongoDB 中，报告展示过程中，与 MongoDB 进行交互，获取展示所需的数据。

Web 应用自动化测试系统的前端为测试报告展示以及软件质量评估报告展示，作为独立的前端展示可以通过 MT 平台进行访问，页面风格和交互方式符合 MT 平台前端统一的标准。前端服务采用 jQuery 开发，由于 jQuery 简洁高效的特点，可以很大程度上增加开发效率。

3.3.2　系统模块划分

Web 应用自动化测试系统由执行服务、分析服务和报告生成服务组成。执行服务部分主要负责 Web 应用自动化测试系统自动化测试用例执行部分，测试执行引擎通过模拟人工测试的操作，完成 Web 应用自动化测试流程。分析服务部分主要负责单页面的缺陷分析，通过输入单页面的执行信息得到单页面的缺陷分析。报告生成服务负责整个自动化测试流程的实现，以及自动化测试用例的生成。报告生成服务通过监控自动化测试执行引擎的运行情况，获取当前页面的执行过程信息。通过输入缺陷分析系统所需的执行过程信息，得到持久化的单页面缺陷分析结果。同时，通过记录的执行路径信息调用自动化测试用例生成服务，得到持久化的自动化测试用例。在监控到执行引擎完成所有测试任务后，系统将所有的单页面分析结果以及自动化测试用例进行整合，生成完整的测试报告数据。系统将得到的测试报告数据，生成可视化的测试报告返回，供用户查看。

根据 Web 应用自动测试系统报告生成服务主要功能实现，系统可以划分为执行引擎监控、测试报告、自动化测试用例生成以及软件质量评估报告四个模块，各模块的主要功能描述如下：

执行引擎监控模块：监控执行引擎的执行过程，在执行引擎产生新的页面状态后中，获取新的页面状态的一些信息，如性能日志、运行环境配置、执行路径等，后续的模块会使用到这些信息。

测试报告模块：负责缺陷分析的调用、单页面分析结果的存储以及完成测试报告的生成以及展示。生成测试报告过程中，测试报告模块将所有的分析服务的结果进行整合，并基于执行环境、缺陷级别和缺陷类型进行统计。测试报告展示主要包括测试结果总览以及缺陷独立展示。测试结果总览包括测试报告的基本、缺陷列举、基于操作系统、缺陷类型、缺陷级别的统计。缺陷独立展示包括缺陷所在网页、相关页面资源、缺陷截图以及复现方法。

自动化测试用例生成模块：使用执行引擎监控模块获取的浏览器配置、执行路径等信息，基于 Selenium 的语法，生成 Selenium 自动化测试用例，便于系统用户可以快速、便捷地复现测试流程，验证缺陷。

软件质量评估报告模块：负责 Web 应用软件质量多维评估报告内容获取以及展示。软件质量评估包含五个维度的评估，涵盖功能、性能、健壮性、兼容性、稳定性。测试执行完成后，软件质量评估报告模块需要从执行引擎中获取测试流程图存储到数据库中，用于辅助展示系统的性能以及兼容性结果。软件质量评估报告内容包括评估结果雷达图以及各个维度的详细报告。

3.3.3　4+1 视图

在软件架构领域工程中应用最广泛的一种方法是由 Philippe Kruchten 教授提出的“4+1”视图法。本章中的图3.2是 UML 中的场景视图实例，常用于描述用户场景，该图代表“4+1”视图中的“1”。其余的“4”个视图依次为逻辑视图，用于设计过程中描述对象模型；开发视图，用于描述在开发环境下的静态组织；进程视图，用于描述系统并发、同步设计；物理视图，用于描述软件的硬件设计。本节中将依次给出系统的“4”个视图，对系统进行描述。

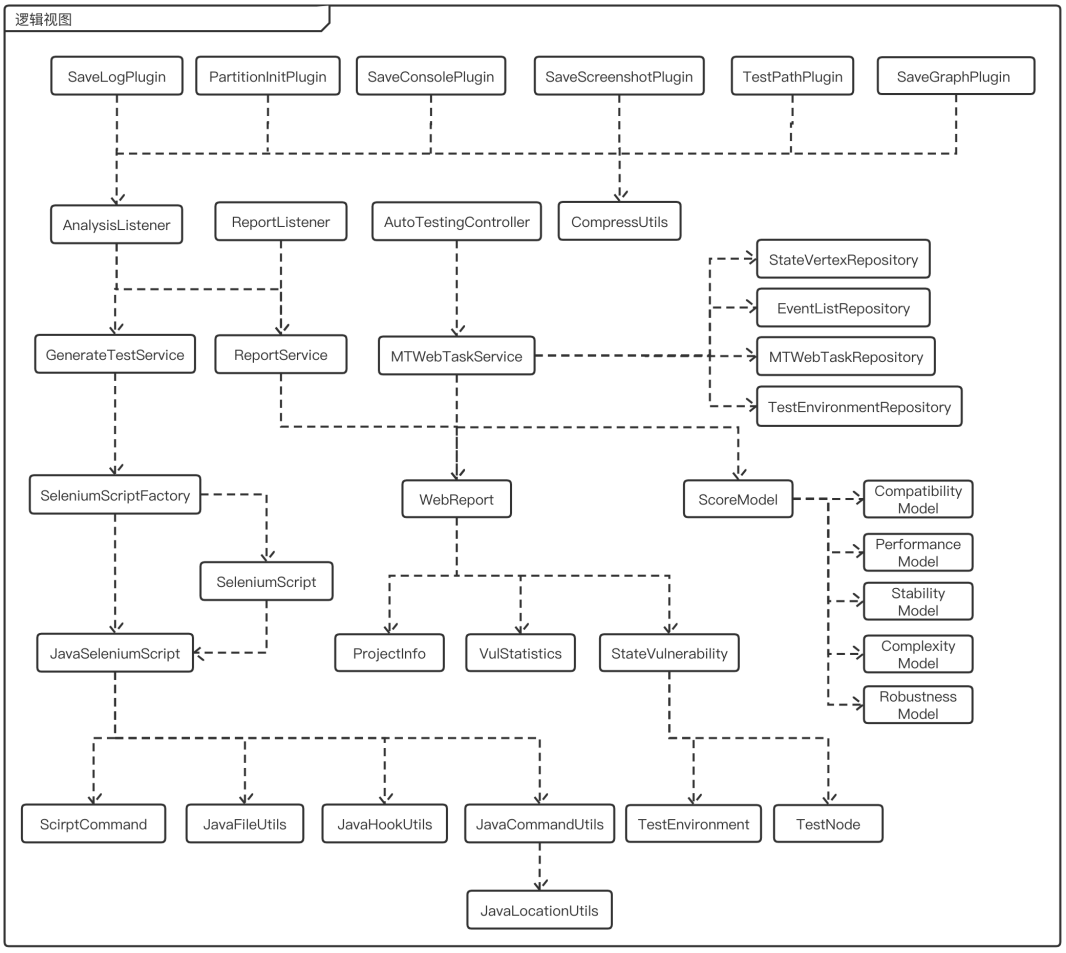

图 3.4: 逻辑视图

系统的逻辑视图如图3.4所示。逻辑视图主要是通过面向对象设计方法对系统的功能进行对象模型描述，包括用户可见功能以及用户不可见的辅助功能。SaveLogPlugin、PartitionInitPlugin、SaveConsolePlugin、SaveScreenshotPlugin、TestPathPlugin、SaveGraphPlugin 主要用于监控执行引擎的运行，收集相关过程信息，

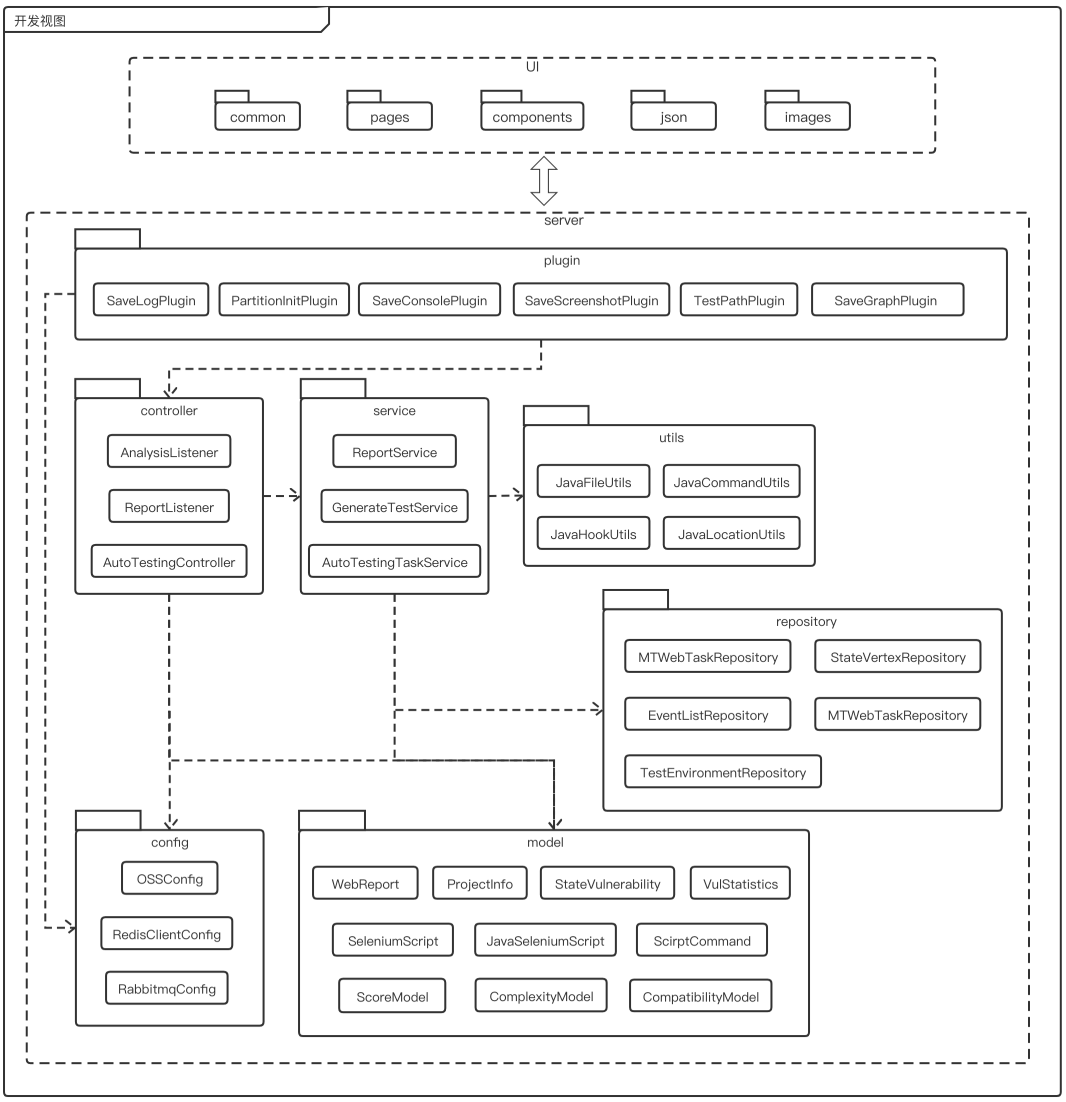

图 3.5: 开发视图

以供后续分析流程使用。CompressUtils 用于在 Redis 传输数据过程中对数据进行压缩。AnalysisListener 主要用于整个单页面状态分析的调用。ReportListener 负责完整的测试报告生成的调用。AutoTestingController 负责响应系统前端获取测试报告的需求。ReportService 负责测试报告生成的逻辑。GenerateTestService、SeleniumScriptFactory、SeleniumScript 主要用于生成自动化测试用例。JavaHookUtils、JavaFileUtils、JavaCommandUtils、JavaLocationUtils 主要用于提供 Java 自动化测试用例的生成方法。JavaHookUtils 负责自动化测试用例中静态语句的生成；JavaFileUtils 负责自动化测试用例文件名的生成；JavaCommandUtils 负责自动化测试用例中 Selenium 命名语句的生成；JavaLocationUtils 负责自动化测

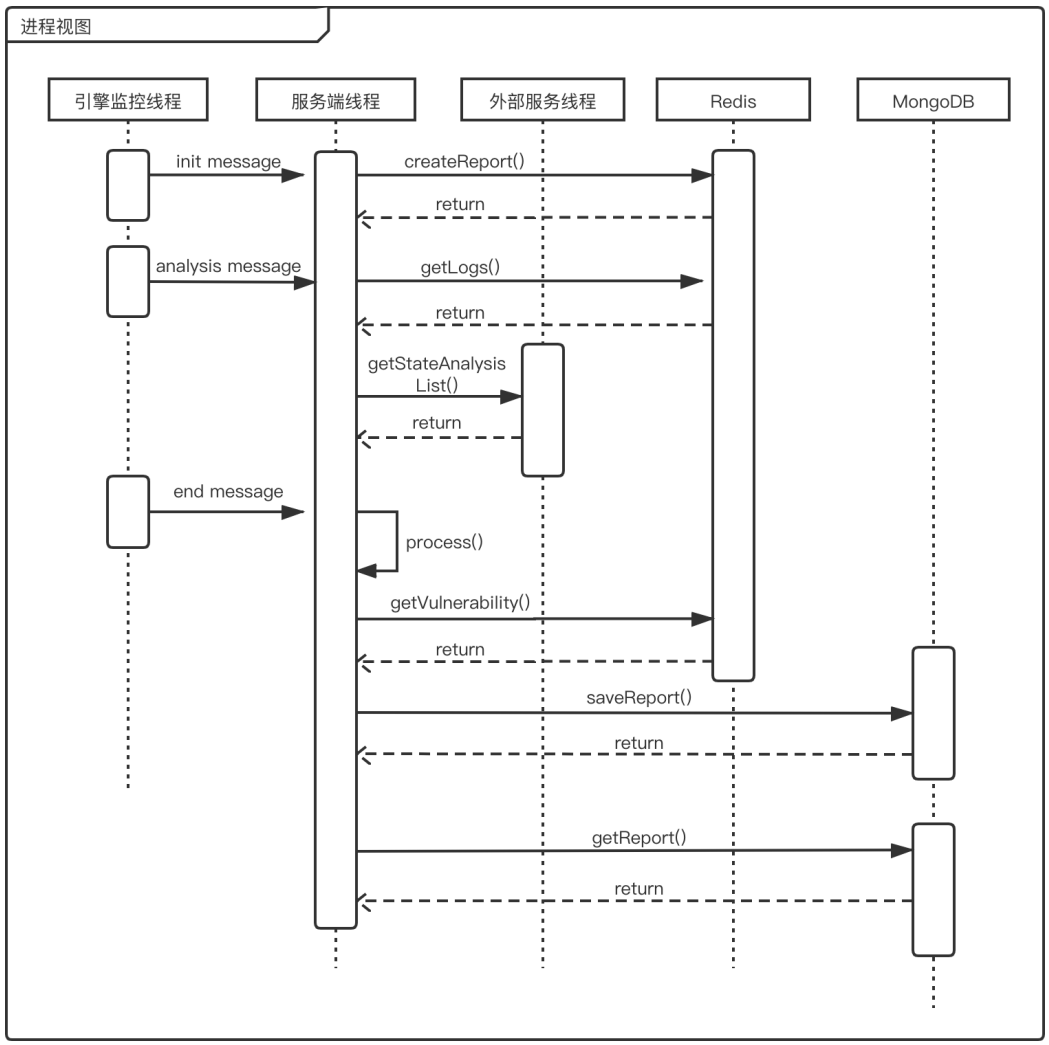

图 3.6: 进程视图

试用例中网页元素定位方法。MTWebTaskService、MTWebTaskRepository、StateVertexRepository、TestEnvironmentRepository、EventListRepository 主要用于从 MongoDB 数据库中获取测试报告。与自动化测试用例生成流程相关的实体类包括 JavaSeleniumScript 以及 ScirptCommand。JavaSeleniumScript 是 Java 自动化测试用例的实体类。ScirptCommand 是自动化测试用例中代码的实体类。与测试报告相关的实体类包括 WebReport、ProjectInfo、StateVulnerability、VulStatistics、TestEnvironment、TestNode。WebReport 是测试报告的实体类，包括其他三个类的信息。ProjectInfo 是待测应用和测试任务基本信息的实体类，包括待测应用 URL、状态数、缺陷数，测试任务开始时间、执行时间、发起用户。StateVulnerability 是单页面状态分析结果的实体类，包括单页面基本信息和缺陷列表。VulStatistics

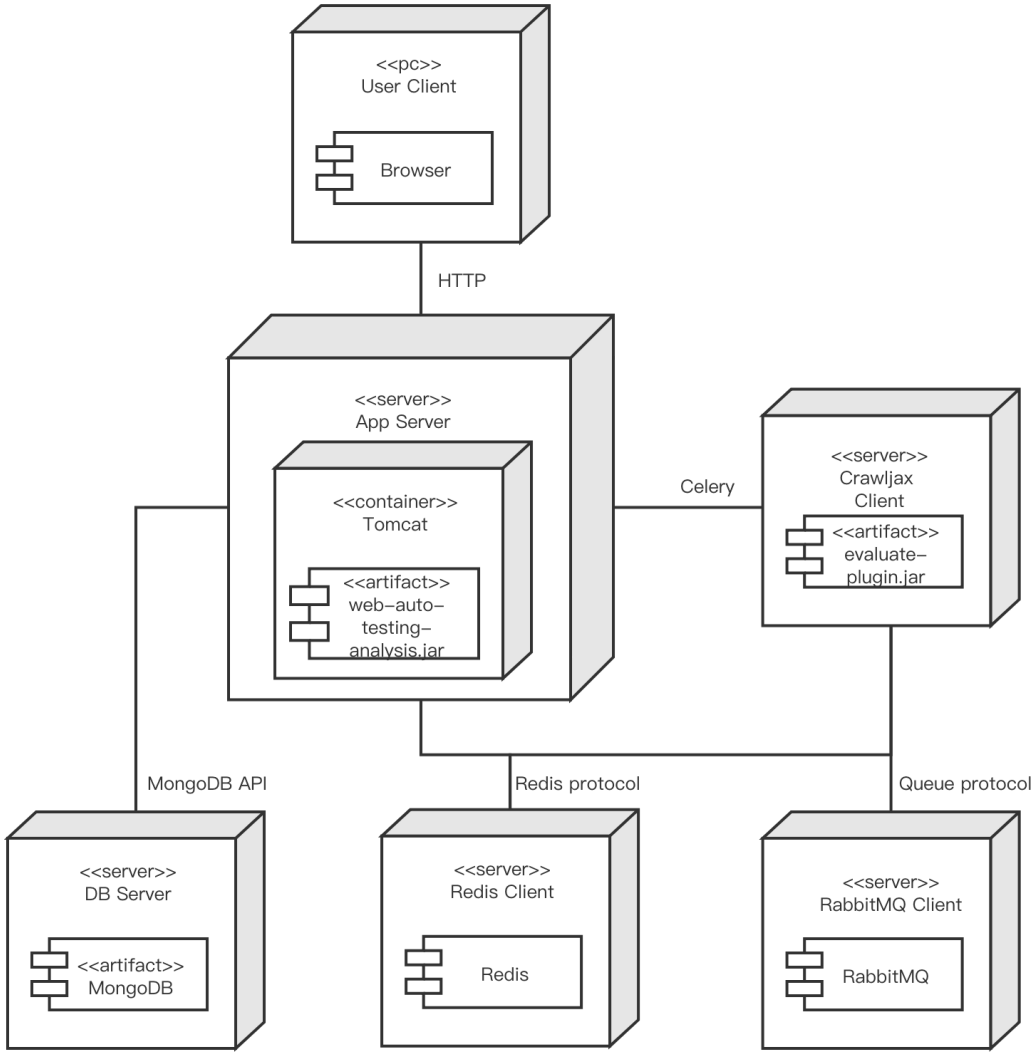

图 3.7: 物理视图

是测试缺陷统计结果的实体类；TestEnvironment 是环境配置实体类，包括操作系统和浏览器信息；TestNode 是测试路径中节点的实体类，包含当前状态信息和执行引擎操作浏览器信息。ScoreModel、ComplexityModel、PerformanceModel、RobustnessModel、CompatibilityModel、StabilityModel 是软件质量评估报告的实体类，其中 ScoreModel 包含整个报告的信息，其余五个类分别对应功能、性能、健壮性、兼容性、稳定性五个维度的详细报告。

如图3.5所示的是系统的开发视图。UI 部分由为页面、组件和静态资源组成。pages 包包含前端页面，components 包包含前端页面组件，静态资源如数据、图片保存在其他包中。服务端主要根据系统模块以及其实现的业务逻辑功能划分。

plugin 包负责实现执行引擎监控模块相关业务逻辑，实现测试执行过程数据的获取以及分析流程触发。controller 包负责接收外部请求以及来自消息队列的消息。Service 负责具体业务逻辑的实现，接收 controller 包业务逻辑控制。repository 负责数据库交互操作。config 负责初始化外部服务的连接配置，包括 redis 配置、RabbitMQ 配置、OSS 连接。utils 包负责提供生成自动化测试用例的方法。model 包负责实现运行时数据存储模型。

如图3.6所示的是系统的进程视图。当执行引擎接收到测试需求并开始执行测试后，执行引擎中的监控模块开始对其状态进行监控。当执行引擎完成完成初始化后，执行引擎监控线程会根据执行引擎状态的变化，通过 RabbitMQ 消息向服务端主线程发送消息。服务端线程通过 Redis 请求从 Redis 数据库中获取数据。服务端线程通过 API 调用外部分析服务。在完整测试报告生成流程后，通过 MongoDB API 同 MongoDB 数据库进行交互。当测试任务完成后，用户在前端获取测试报告请求会通过 RESTful 接口请求调用服务端线程，获取测试任务结果。

系统的物理视图如图3.7所示。系统的用户主要通过浏览器与系统进行交互，获取 Web 应用自动化测试服务。用户的请求会通过 HTTP 到达应用服务器，应用服务会使用 Celery 调度测试执行引擎服务，执行引擎插件位于执行引擎服务之中。执行引擎监控插件以及应用服务，使用 Redis 协议进行缓存数据的读写。同时，两者使用 RabbitMQ 协议进行信息的通行。系统数据库单独部署，应用服务通过 Spring 集成的 MongoDB API 与数据服务器进行交互，进行数据增删该查操作。

3.4　执行引擎监控模块设计

3.4.1　监控功能分析

执行引擎监控模块主要负责执行引擎运行过程中的数据采集。根据系统其他模块的数据需求，执行引擎监控模块所需的获取的执行过程信息主要包括执行引擎环境配置信息、浏览器性能日志、浏览器日志、浏览器执行截图、执行测试路径信息以及测试流程图，测试流程图包含测试过程中所有的状态节点以及操作路径。其中，执行引擎配置信息需要在执行引擎初始化过程中获取到，浏览器性能日志、浏览器控制台日志、浏览器执行截图以及执行测试路径信息需要在执行引擎产生新的页面状态过程中获取，测试流程图信息需要在测试执行完后获取。考虑到不同的监控需求，执行引擎监控模块设计中引入 Crawljax Plugin 中的 PreCrawlingPlugin、OnNewStatePlugin 以及 PostCrawlingPlugin 三个监控模块接

口，PreCrawlingPlugin 用于在执行引擎初始化过程中进行记录，OnNewStatePlugin 在执行引擎产生新的页面状态时进行记录，PostCrawlingPlugin 用于在测试执行完成后进行记录。

表 3.12: 执行引擎监控模块信息 Redis Key 设计

描述项	说明
执行引擎环境配置信息	PatitionPrefix_config
浏览器性能日志	StatePrefix_performancelog
浏览器日志	StatePrefix_browserlog
浏览器执行截图	StatePrefix_screenshot
执行测试路径信息	StatePrefix_testpath
所有状态节点信息	StatePrefix_statelist
所有操作路径信息	StatePrefix_eventlist

由于其他模块的异步调用数据的需求，执行引擎监控模块需要将记录的执行过程信息存储到 Redis 中，Redis 中的 Key 的设计如表3.12所示，表中的 Redis Key 的前缀 PatitionPrefix 是执行引擎的标志符，前缀 StatePrefix 均为执行引擎过程中产生的页面状态的标志符。由于存储浏览器截图需要存储截图文件，所以引入了 OSS 作为文件存储服务。执行引擎监控模块使用 OSS 存储浏览器截图文件，使用 Redis 存储 OSS 文件链接，后续的模块通过访问 Redis 获取截图文件的 OSS 链接，进而获取浏览器执行截图。

当执行引擎监控模块出现新的页面状态时，执行引擎监控模块需要通知测试报告模块调用单页面分析服务。由于这是一种消息生成和消费的关系，同时执行引擎采用多例的模式，为了处理这样的消息调用，我们在执行引擎监控模块引入了 RabbitMQ 面向消息中间件服务。

3.4.2　架构设计

执行引擎监控模块架构图如图3.8所示。执行引擎监控模块主要与执行服务和测试报告模块进行交互，通过监控执行引擎模块获取相应的测试信息触发测试报告模块的分析流程。执行引擎监控模块主要对执行服务中的执行引擎的三个状态进行监控，包括执行引擎初始化、执行引擎中新的页面状态出现以及测试执行完成执行引擎关闭。在执行引擎初始化完成后，执行引擎监控模块会对其环境配置信息进行获取。在新的页面状态出现后，执行引擎监控模块会对单页面状态信息以及执行路径进行获取，状态信息包括浏览器日志、性能日志、执行截图。测试执行完成执行引擎关闭时，执行引擎监控模块会对测试过程中所有状态节点以及操作路径信息进行获取。在获取以上信息后，执行引擎监控模块会使用缓存中间件 Redis 对信息进行存储，以供测试报告模块使用；使用消息

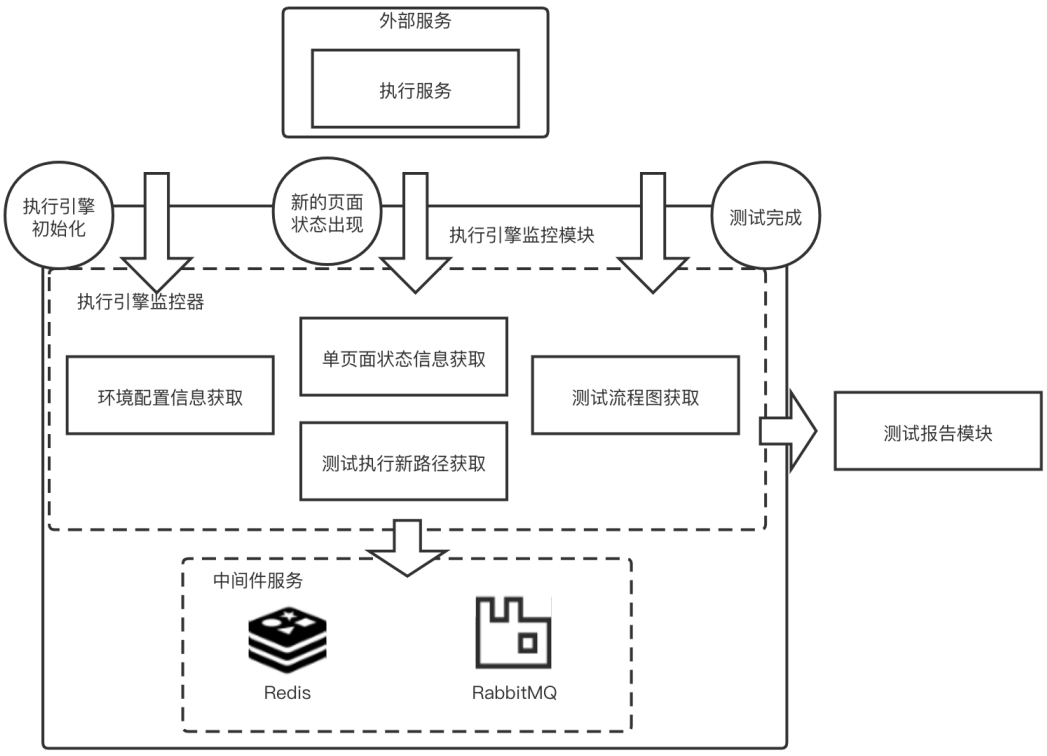

图 3.8: 执行引擎监控模块架构图

中间件 RabbitMQ 调用测试报告模块中相应的流程。

3.4.3　核心类图

执行引擎监控模块的类图如图3.9所示。执行引擎监控模块的主要类为执行引擎监控插件类，主要实现了 PreCrawlingPlugin 和 OnNewStatePlugin 两个插件接口。

PartitionInitPlugin 类实现了 PreCrawlingPlugin 接口，主要负责执行引擎环境配置信息的采集。当执行引擎打开测试任务为它配置的浏览器时，PartitionInitPlugin 监控器会被触发，通过初始化过程中输入的参数 MTWebClientConfigrationModel，获取执行引擎中的浏览器的配置信息，使用表3.12表中的规则存储到 Redis 中。

SaveLogPlugin 类实现了 OnNewStatePlugin 接口，主要负责浏览器性能日志的获取和存储。当执行引擎在测试过程中产生新的页面状态时，SaveLogPlugin 监控器会被触发，通过访问执行引擎中的 CrawlerContext 任务上下文对象，获取浏览器性能日志，使用表3.12中设计的规则存储到 Redis 中。

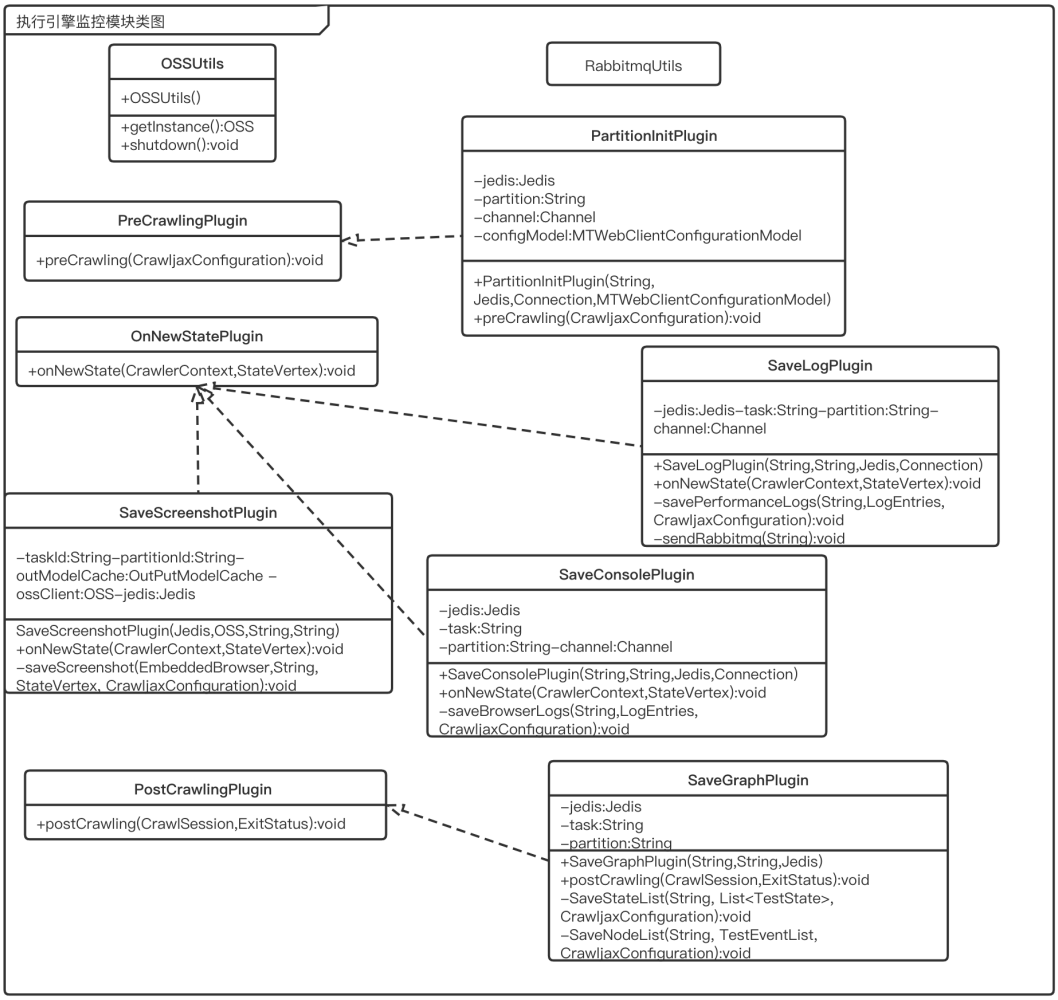

图 3.9: 执行引擎监控模块核心类图

SaveConsolePlugin 类实现了 OnNewStatePlugin 接口，主要负责浏览器日志的获取和存储。SaveConsolePlugin 类和 SaveLogPlugin 类的运行方式一致，监控执行引擎中新的页面状态，SaveConsolePlugin 监控器获取浏览器日志后，使用表3.12中设计的规则存储到 Redis 中。

TestPathPlugin 类实现了 OnNewStatePlugin 接口，主要负责执行引擎测试路径的收集。当执行引擎在测试过程中产生新的页面状态时，TestPathPlugin 监控器会被触发，通过访问执行引擎中的 CrawlerContext 任务上下文对象中的状态流程记录，获取并存储该页面状态的执行流程，使用表3.12中设计的规则存储到 Redis 中。

SaveScreenshotPlugin 类实现了 OnNewStatePlugin 接口，主要负责执行测试中页面截图的获取。当执行引擎在测试过程中产生新的页面状态时，SaveScreenshotPlugin 监控器会被触发，获取页面截图后，分别访问 OSS 和 Redis 存储相关信息，其中 OSS 存储截图文件、Redis 存储 OSS 路径。

在执行引擎监控模块中使用了多个不同的 Plugin 来实现 OnNewStatePlugin，主要目的是使得各个监控器任务独立，使得任务粒度小便于装配，符合设计原则中的单一职责原则，同时增加了执行引擎监控模块的拓展性。

SaveGraphPlugin 类实现了 PostCrawlingPlugin 接口，主要负责测试流程图中的状态节点以及操作路径信息的获取。当测试执行完成、执行引擎关闭后，SaveGraphPlugin 监控器被触发，通过访问 CrawlSession 获取执行过程中的测试流程图对象，由于测试流程图对象结构复杂，无法直接通过 Redis 直接存储，获取该对象中的所有状态节点和操作路径信息分别进行存储。

3.4.4　测试路径存储

由于后续模块的数据需求，执行引擎监控模块需要保存页面状态的测试执行路径。测试路径相比较于其他需要保存的数据结构，较为复杂，所以执行引擎监控模块定义了 TestNode 数据结构，来存储测试路径信息。TestNode 是测试执行过程中的一个操作，一个 TestNode 组成的列表组成了测试路径。TestNode 的属性的对应关系如3.13表所示。其中 relatedFormInputs 的属性为 FormInputModel 的列表，FormInputModel 是一个输入操作的数据结构，其中的属性包括输入元素对象的定位方式、输入元素的类型以及输入的值。测试流程图中的操作路径信息存储也基于上述方式。

表 3.13: TestNode 对象属性表

属性名	类型	含义
sourceStateName	String	源页面状态名称
sourceStateUrl	String	源页面状态 URL
targetStateName	String	目标页面状态名称
targetStateUrl	String	目标页面状态 URL
eventType	String	操作类型
how	String	元素定位方法名称
value	String	元素定位值
relatedFormInputs	List<FormInputModel>	输入操作列表
relatedFrame	String	切换 Frame 名称

3.5　测试报告模块设计

3.5.1　架构设计

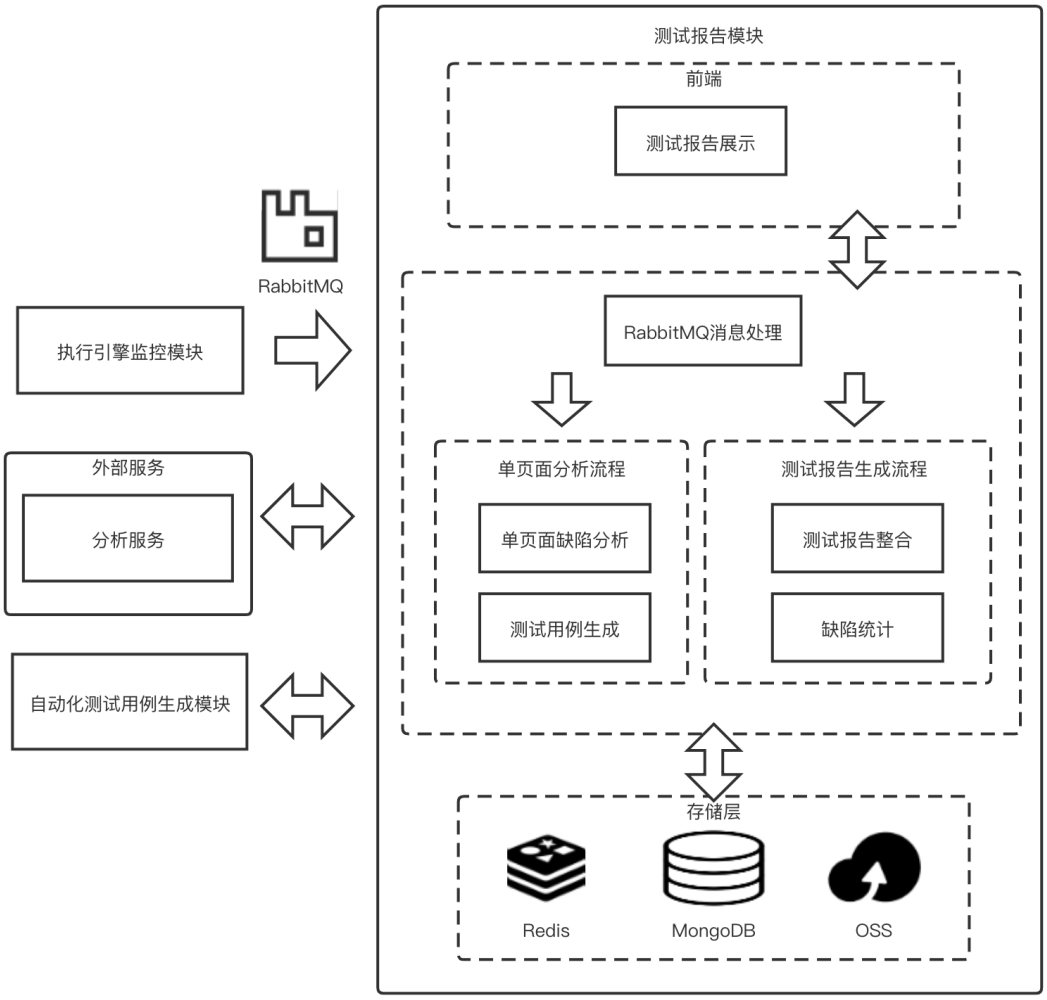

图 3.10: 测试报告模块架构图

测试报告模块的主要功能是单页面的缺陷分析以及测试报告的生成和展示。因此模块服务端有两个主要的业务流程，单页面分析流程以及测试报告生成流程。测试报告模块的架构图如图3.10所示。测试报告模块服务端主要使用 RabbitMQ 消息处理部分与系统中的执行监控模块进行交互，执行监控模块生成的消息被 RabbitMQ 消息处理部分消费，并启动两个业务流程。

单页面分析流程主要包括以下步骤：当执行引擎监控模块检测到执行引擎搜索并进入了一个新的页面状态后，执行引擎监控模块会将获取的执行过程信

息存储到 Redis 中，并在所有信息存储完成后向单页面分析流程绑定的 RabbitMQ 消息队列中发送一个单页面分析的消息信号并传递新页面状态在 Redis 中的标志信息。测试报告模块的 RabbitMQ 消息处理部分，会在信息信号出现后，对信号进行消费，启动单页面分析服务流程。单页面分析服务流程会分别依次调用外部分析服务模块进行单页面缺陷分析，并将缺陷分析结果的缺陷列表存储到 Redis 中；调用自动化测试用例生成模块进行测试用例生成，并将生成的测试用例文件存储到 OSS 中。

测试报告生成流程主要包括以下步骤：当执行引擎开始进行 Web 应用自动化测试后，会向测试报告生成流程绑定的 RabbitMQ 消息队列发送一个创建测试报告的消息信号，测试报告模块的 RabbitMQ 消息处理部分会启动测试报告生成流程，初始化新的测试报告并将初始化的测试报告存储到 Redis 中。在执行引擎结束配置的页面测试后，会向测试报告生成流程绑定的 RabbitMQ 消息队列发送一个生成测试报告的信息，测试报告生成流程会从 Redis 中获取当前测试任务的所有分析结果，进行整合。在整合缺陷列表的过程中，测试报告生成流程会对缺陷列表进行统计。统计包括执行环境、缺陷级别和缺陷类型。生成完整的测试报告后，测试报告生成流程会将生成的测试报告数据存储到 MongoDB 中，为测试报告前端提供数据。

前端为测试报告展示部分，主要通过调用测试报告模块的 Spring 接口获取前端展示数据。前端主要使用 jQuery 实现，主要展示内容包括测试结果总览、缺陷列表展示以及缺陷详情展示。测试结果总览包括测试报告的基本信息、基于操作系统、缺陷类型、缺陷级别的统计。缺陷列表展示，包括缺陷列表信息、基于操作系统、缺陷类型、缺陷级别的筛选。缺陷详情展示包括缺陷所在网页、相关页面资源、缺陷截图以及复现方法。

3.5.2　核心类图

测试报告模块服务端的类图如图3.11所示，主要包括服务接口类和数据类。AnalysisListener、ReportListener 类是测试报告模块中处理 RabbitMQ 消息的部分。AnalysisListener 绑定在单页面分析流程的 RabbitMQ 消息队列中，负责消费单页面分析消息的消息。AnalysisListener 类在接收到处理信息信号后会分别调用单页面缺陷分析和自动化测试用例生成服务。在获得返回结果后，AnalysisListener 会把数据存储到 Redis 中，以供后续的测试报告生成服务使用。ReportListener 绑定在测试报告生成流程的 RabbitMQ 消息队列中，负责消费测试报告相关服务的消息。ReportListener 在得到执行引擎发出的创建测试报告请求后，会针对当前执行引擎中的测试任务初始化一个新的测试报告对象，然后将其存储在 Redis

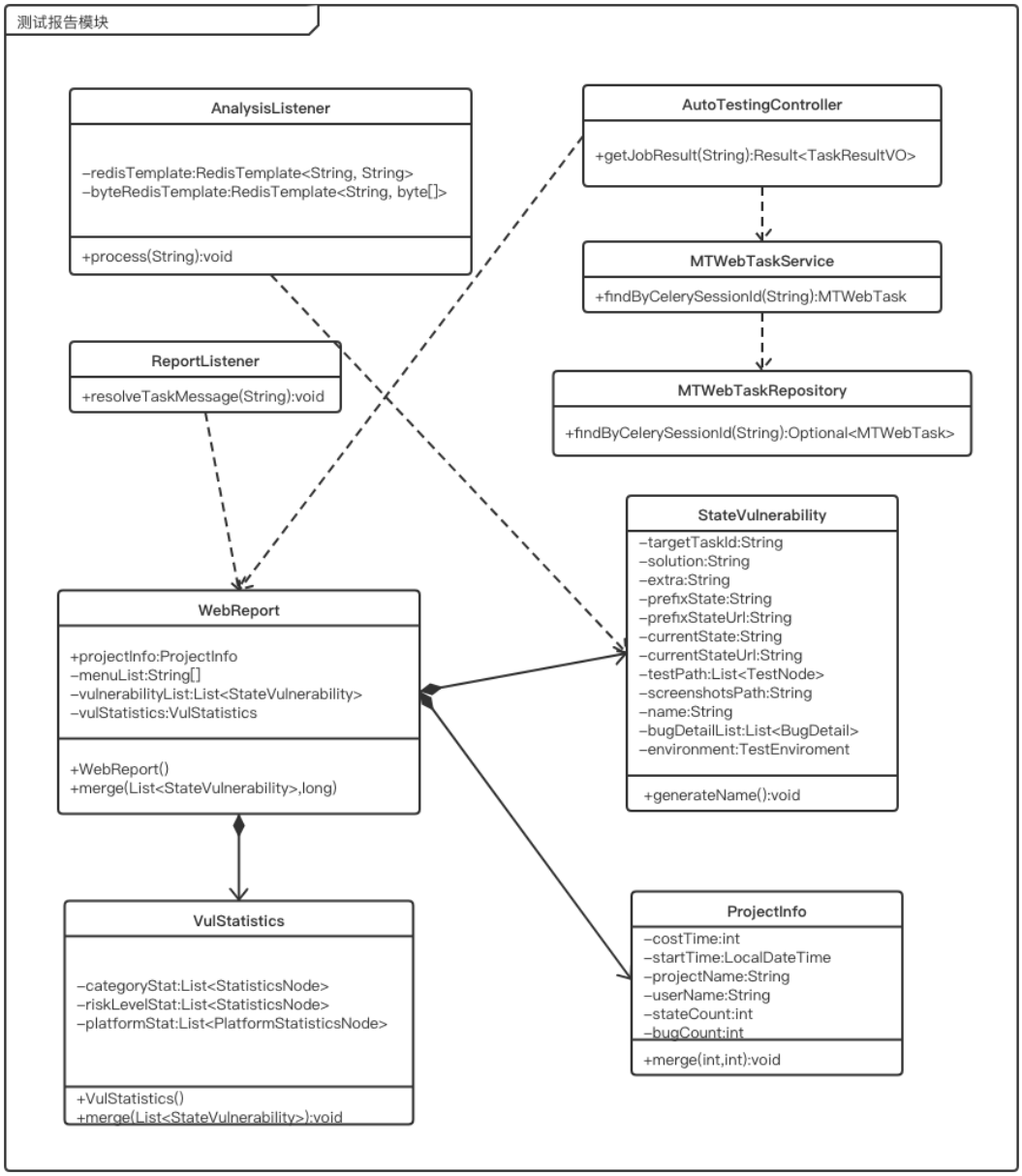

图 3.11: 测试报告模块服务端类图

中。在得到执行引擎发出的测试接收消息后，会针对结束的测试任务获取 Redis 中初始化的测试报告对象以及单页面分析结果，进行整合和统计，生成完整的测试报告，存储到 MongoDB 数据库中。

WebReport、ProjectInfo、StateVulnerability 以及 VulStatistics 类是测试报告模块的数据类，负责数据存储模型。

ProjectInfo 类是存储测试任务的总览信息数据类，包括测试任务名称、发布测试任务的用户名、测试任务开始时间、测试执行时间、测试页面状态数以及缺陷数量。在 ReportListener 初始化测试报告时，ProjectInfo 类会初始化测试任务名称、发布测试任务的用户名、测试任务开始时间。在 ReportListener 生成完整测试报告时，ProjectInfo 类会对测试任务结束时间、测试页面状态数、缺陷数量进行统计更新。

StateVulnerability 类是单页面分析结果数据类，包括页面状态基本信息，如当前状态名称、当前状态 URL、前缀状态名称以及前缀状态 URL 等，到达当前页面的测试路径，当前页面状态环境信息、当前页面状态缺陷列表、当前页面状态截图。

VulStatistics 类是存储了测试报告中的统计结果，把包括环境信息配置信息统计、缺陷类型统计以及缺陷级别统计。在 ReportListener 生成完整测试报告时，VulStatistics 类会根据获取的缺陷列表对环境配置信息、缺陷类型以及缺陷级别进行统计。

Webreport 类是测试报告的存储结构，包含了 ProjectInfo 类、StateVulnerability 类以 VulStatistics 类。在 ReportListener 生成完整测试报告时，会触发内部对象根据分析生成的缺陷列表进行更新。

AutoTestingController、MTWebTaskService 和 MTWebTaskRepository 类的部分方法为测试报告展示前端提供测试报告数据，负责通过测试任务编号从 MongoDB 数据库中获取生成的测试报告数据。

3.6　自动化测试用例生成模块设计

3.6.1　架构设计

自动化测试用例生成模块主要通过执行引擎监控模块获取的测试路径，来生成自动化测试用例。自动化测试用例的实现，使用 Selenium 测试用例的方法。Selenium 测试用例主要包括引用声明、类初始化、Selenium 命令以及类关闭四部分组成。引用声明是指在 Selenium 测试用例中需要引入的工具包的声明。类初始化是指执行 Selenium 测试所需的包括 Webdriver、JavascriptExecutor 的初始化。Selenium 命令是 Selenium 测试主要的组成部分，Webdriver 执行的操作集合。类关闭部分指 Webdriver 的释放。其中，引用声明、类初始化以及类关闭这三个部分的生成方式比较固定，而 Selenium 命令部分需要使用执行引擎监控模块获取的测试路径信息并且生成比较复杂，所以使用两个工具类来生成这两类的 Selenium 测试用例部分。

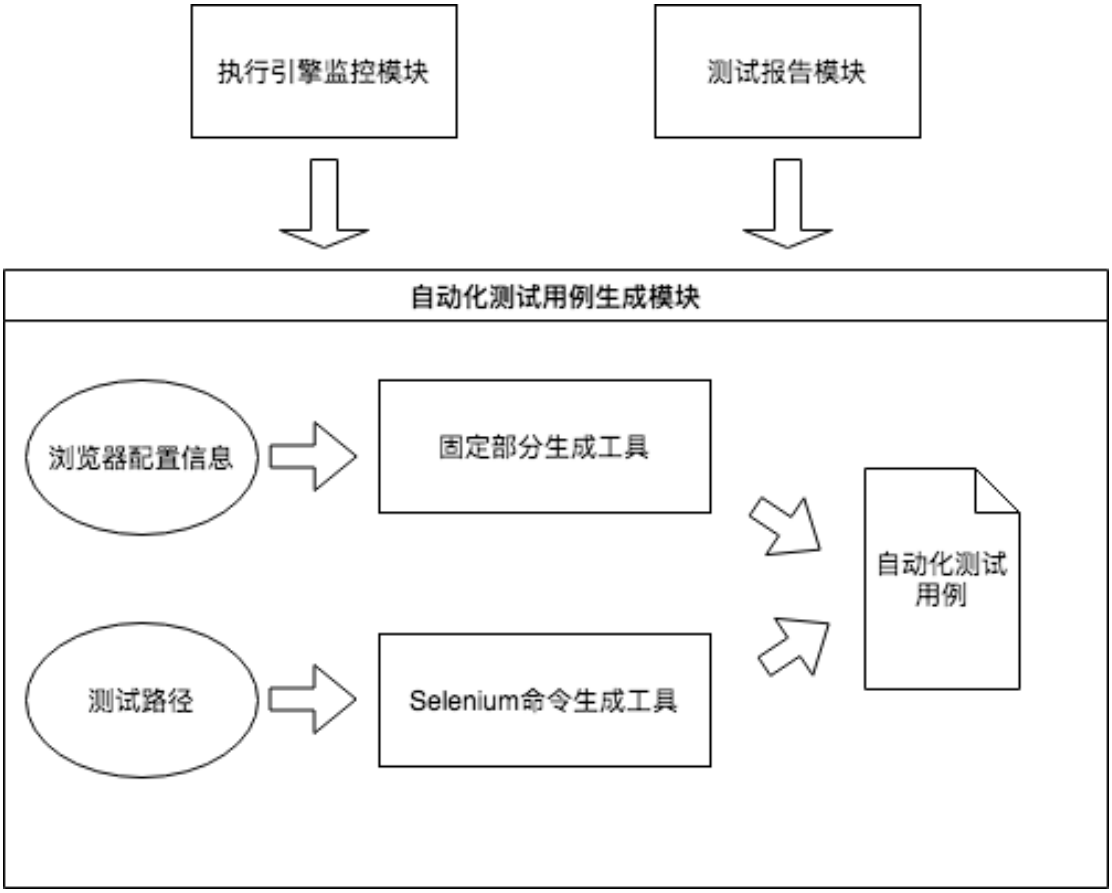

图 3.12: 自动化测试用例生成模块架构图

自动化测试用例生成模块架构设计如图3.12所示。自动化测试用例生成模块与执行引擎监控模块和测试报告模块进行交互。测试报告模块在新的页面状态分析开始是调用自动化测试用例生成模块，在生成自动化测试用例过程中国会使用到执行引擎监控模块获取的浏览器信息以及测试路径信息。固定部分生成工具负责生成自动化测试用例中的引用声明、类初始化以及类关闭部分。在生成类初始化部分的过程中，会使用执行引擎监控模块记录的浏览器信息以初始化相应的 WebDriver。Selenium 命令生成工具会根据执行引擎监控模块记录的测试路径信息，生成相应的 Selenium 命令集合。固定部分生成工具和 Selenium 命令生成工具相互协作，生成完整的自动化测试用例。

3.6.2　核心类图

图3.13中所示的是自动化测试用例生成模块的类图。自动化测试用例生成模块的主要类包括服务类 GnerateTestService、Selenium 数据结构类和测试用例生成工具类。

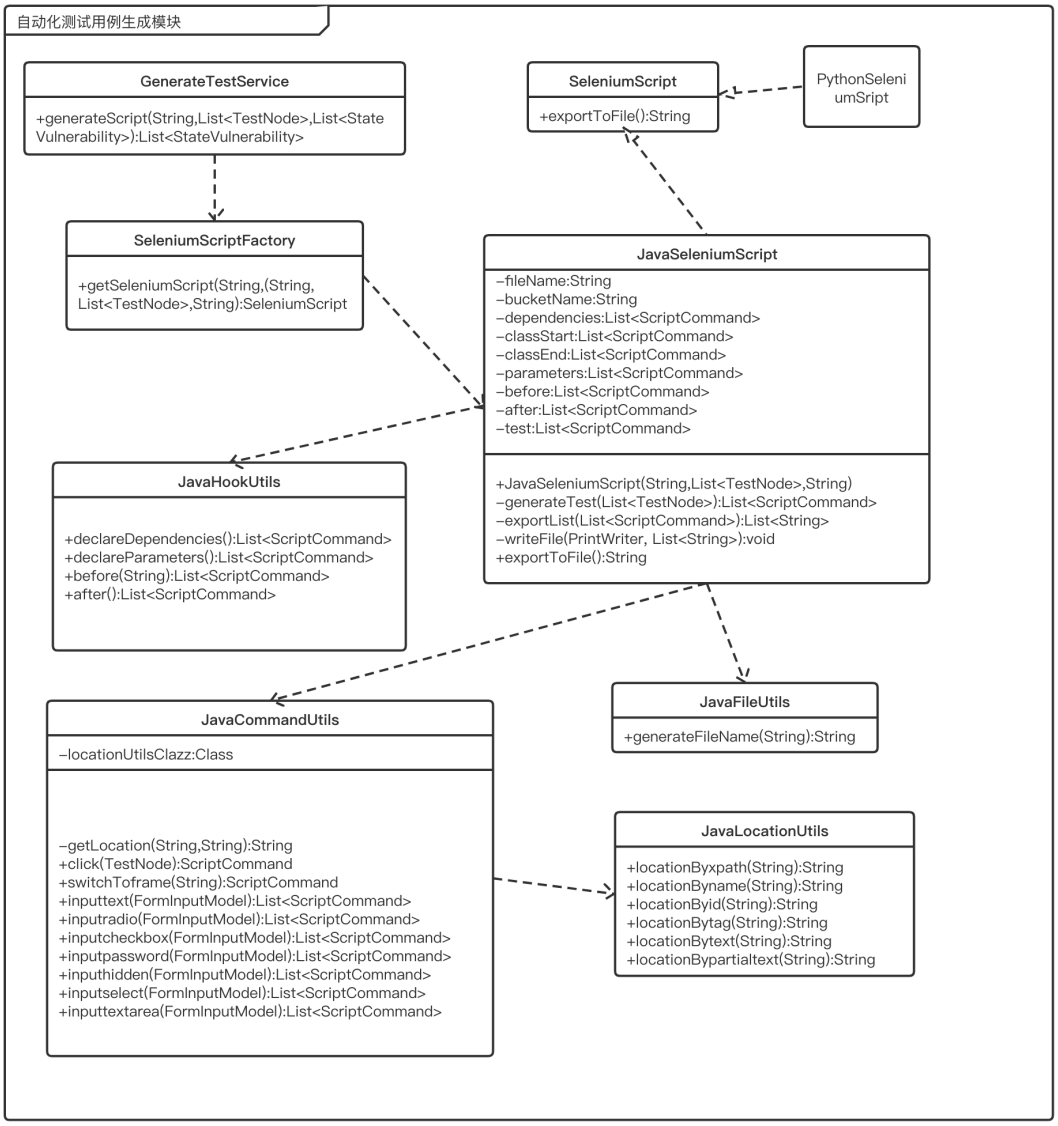

图 3.13: 自动化测试用例生成模块类图

GenerateTestService 类是自动化测试用例生成模块向外提供服务的接口类，测试报告模块在需要触发自动化测试用例生成的时候可以调用 GenerateTestService 接口，获取自动化测试用例生成的服务。

自动化测试用例生成模块使用工厂模式实现了不同编程语言下的自动化测试用例生成。工厂模式相关的类包括 SeleniumScriptFactory、SeleniumScript、JavaSeleniumScrpit、PythonSeleniumSript 等，在图3.13中仅展示了 Java Selenium 测试用例生成的相关类。

SeleniumScrpit 类是不同编程语言测试用例类的抽象接口，定义了 exportToFile 方法，该方法负责把生成的 Selenium 语句输出到指定的文件中，形成自动户测试用例。SeleniumScriptFactory 负责不同语言的 SeleniumScript 的创建。GenerateTestService 调用工厂类的 getSeleniumScrpit 方法，通过输入语言类型，生成不同的 SeleniumScript 对象。

JavaSelniumScript 类是 Java 语言的 Selenium 测试用例数据类，使用迪米特法则设计，把 Selenium 测试用例各部分的生成过程封装在 JavaSeleniumScript 类内部实现。JavaSeleniumScript 类的成员变量表示了 Selenium 测试用例的各个组成部分。fileName 定义了输出的文件名，根据编程语言不同文件会产生不同的命名方式。dependencies 是 Selenium 测试用例的引用声明部分，paramenters 和 before 组成了类初始化部分，test 是 Selenium 命令部分，after 是类关闭部分，classStart 和 classEnd 组成了 Java 类的类名和结束符。

JavaFileUtils、JavaHookUtils、JavaCommandUtils 和 JavaLocationUtils 类是生成 Java Selenium 测试用例的工具类，工具类设计过程中使用单一职责原则，各工具类负责不同部分的生成。JavaFileUtils 负责 Java Selenium 测试用例的文件名生成，根据 Java 命名的规则生成测试用例文件名。JavaHooksUtils 负责 Selenium 测试用例中的引用声明、类初始化和类关闭部分的代码生成，类初始化部分通过接收 Webdriver 标志作为输入，声明类初始化过程中定义不用的 Selenium WebDriver。JavaCommandUtils 负责根据输入的测试路径，生成不同的 Selenium 命令，包括 click、sendkeys 等操作。JavaLocationUtils 提供 Selenium 命令中的元素定位方法，可以采用 xpath、name、id、tag、text、partialText 进行网页元素的定位。

3.7　软件质量评估报告模块设计

3.7.1　架构设计

软件质量评估报告模块主要是通过测试报告模块的测试结果以及测试过程数据，调用分析服务被测 Web 应用的软件质量进行评估，对评估结果进行展示。质量评估的维度包括功能、性能、健壮性、兼容性、稳定性，其中功能、兼容性评估的详细报告的展示依赖测试流程图，软件质量评估模块需要从执行引擎中获取测试流程图。由于获取测试流程图的过程属于测试任务执行的一部分，而报告展示需要异步展示测试流程图，因此使用数据库对测试流程图进行存储。

软件质量评估报告模块架构图如图3.14所示。软件质量评估报告模块与测试报告模块进行交互，当测试报告模块接收到来自 RabbitMQ 消息队列的测试执行

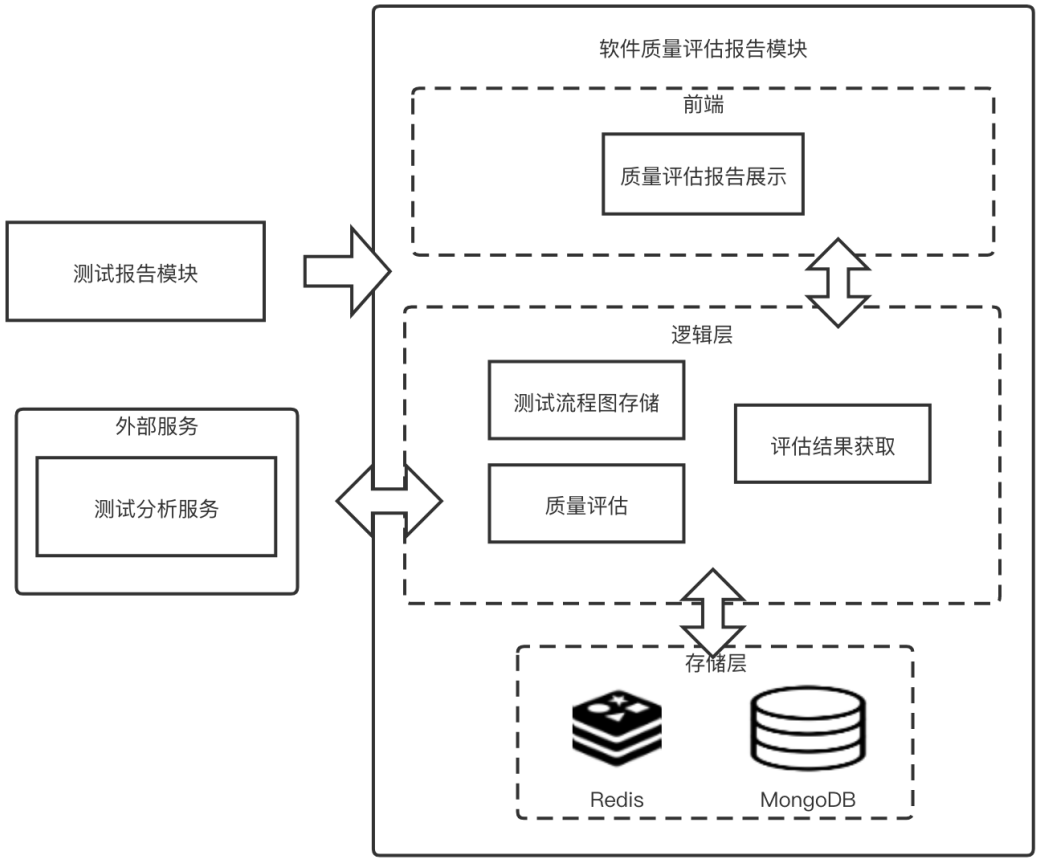

图 3.14: 软件质量评估报告模块架构图

结束消息后，进行测试结果的整合，通过测试结果调用软件质量评估报告模块进行评估结果以及测试流程图的获取和存储。软件质量评估报告模块调用分析服务的质量评估模型获取评估结果并存储到 MongoDB 数据库中，同时，从 Redis 中获取执行引擎监控模块获取的测试流程图，存储到 Redis 中。当前端展示部分需要获取质量评估结果时，软件质量评估报告模块访问 MongoDB 数据库，获取测试流程图生成功能、兼容性的详细报告，以及性能、健壮性以及稳定性的详细报告。

软件质量评估报告模块的前端为软件质量评估报告展示部分，主要包括质量评估结果雷达图展示以及各个维度的详细报告展示。质量评估结果雷达图展示了被测 Web 应用在质量评估中的综合结果。功能报告包括应用在测试过程中检测到的网页数、可交互操作数以及应用功能导航图；性能报告包括可能存在性能问题的请求列表；健壮性报告包含影响应用健壮性的缺陷列表；兼容性报告包括被测 Web 应用在不同测试环境下的执行路径对比；稳定性报告包括被测

Web 应用中存在断链情况的页面。

3.7.2　核心类图

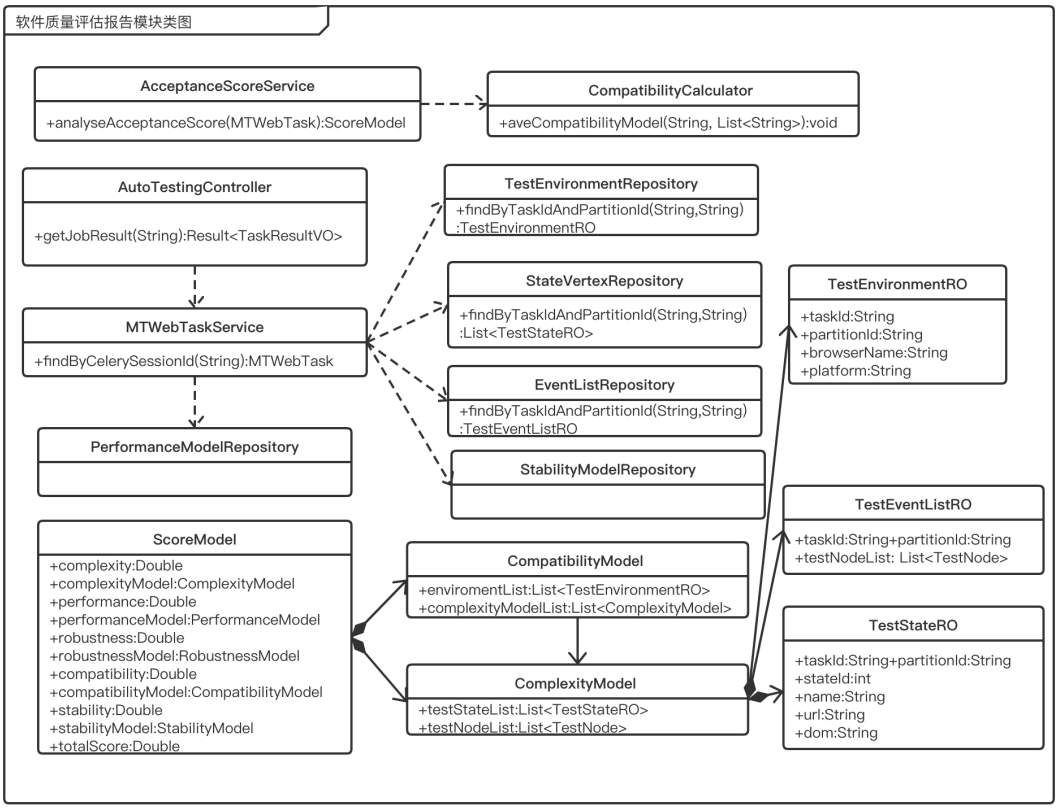

图 3.15: 软件质量评估报告模块类图

软件质量评估报告模块的核心类图如图3.15所示，主要包括获取存储测试流程图以及获取展示数据的服务接口和数据类。

AcceptanceScoreService、CompatibilityCalculator 类用于获取和存储测试流程图的业务逻辑。AcceptanceScoreService 类负责接收来自测试报告模块的调用请求，当测试报告模块检测到执行引擎完成测试后开始生成测试报告，同时调用 AcceptanceScoreService 提供的接口获取测试流程图和并调用分析服务获取软件质量评估结果进行存储。CompatibilityCalculator 类负责从 Redis 数据库中获取执执行引擎监控模块获取的测试流程图信息并将其存储到 Redis 中。

AutoTestingController、MTWebTaskService 类负责实现获取软件质量评估报告数据的服务接口。AutoTestingController 类向前端提供调用接口，获取用于展示的报告数据。MTWebTaskService 类实现业务逻辑层服务接口，通过调用数据层服务，整合各个分项报告。

TestEnvironmentRepository、StateVertexRepository、EventListRepository、StabilityModelRepository、PerformanceModelRepository 类为数据层接口，继承了 MongoRepository 类实现与 MongoDB 数据库的交互。TestEnvironmentRepository、StateVertexRepository、EventListRepository 类用于获取性能与兼容性详细报告展示信息。StabilityModelRepository、PerformanceModelRepository 类分别用于获取稳定性和性能详细报告。

ScoreModel、CompatibilityModel、ComplexityModel 类以及类图中未包含的 PerformanceModel、RobustnessModel、StabilityModel 类实现了软件质量评估报告的数据存储，后三者由于主要在分析服务中实现因此不作阐述。ScoreModel 类是完整的软件质量评估报告数据集，其中包括了功能、性能、健壮性、兼容性、稳定性五个维度的详细报告、五个维度的评分以及质量评估总分。CompatibilityModel、ComplexityModel 类主要包含了测试流程图信息，用于前端功能报告部分展示功能流程图和兼容性报告部分展示不同浏览器环境下的不同测试执行流程。

TestEnvironmentRO、TestEventListRO、TestStateRO 类是存储不同环境下测试流程图的数据类。TestEnvironmentRO 类负责存储执行过中配置使用或者能够运行被测 Web 应用的浏览器环境，同时也包括测试所用的操作系统环境。TestEventListRO、TestStateRO 类分别存储了测试流程图中的状态节点和操作路径信息。

3.7.3　数据库设计

软件质量评估报告模块使用 mongoDB[1]作为数据存储，用于存储测试执行过程中获取的测试流程图，以供软件质量评估报告展示过程中使用。系统实现过程中通过 spring-data-mongodb 集成 MongoDB 服务。

模块将测试流程图拆分为状态节点列表和包含起初节点和目标节点的操作路径列表进行存储。状态节点以及操作路径细信息的存储结构如表3.14所示，上半表为状态节点存储结果，下半表为操作路径存储结构，分别映射到 Java 类 TestStateRO、TestEventListRO。由于 TestStateRO 包含的网页文档信息数据可能过大，影响 MongoDB 正常存储，因此采用多个文档对象保存状态节点列表，而操作路径列表采用数组字段进行存储。浏览器环境信息存储主要包含浏览器名称以及操作系统名称，映射到 Java 类 TestEnvironmentRO。

[1]https://www.mongodb.com/

表 3.14: 软件质量评估报告模块数据库文档设计

字段名	**类型**	**字段描述**
taskId	String	任务编号
partitionId	String	浏览器编号
stateId	int	页面状态编号
name	String	页面状态名
url	String	页面链接
dom	String	页面 DOM 文档
字段名	**类型**	**字段描述**
taskId	String	任务编号
partitionId	String	浏览器编号
testNodeList	List<TestNode>	操作路径列表

3.8　本章小结

本章首先介绍了 Web 应用自动化测试系统的整体描述，对系统的功能、非功能需求进行分析，并对系统的用例进行描述；其次分析了 Web 应用自动化测试系统的整体架构，对本文中的报告生成服务进行模块划分，以及系统的 4+1 视图；然后对分别 Web 应应用自动测试系统报告生成服务的执行引擎监控模块、测试报告模块、自动化测试用例生成模块以及软件质量评估报告模块四个核心模块分别进行设计概述。执行引擎监控模块主要介绍了其监控功能、核心类图以及测试路径存储方式；测试报告模块主要描述了其结构设计以及核心类图；自动化测试用例生成模块主要介绍了其架构设计以及核心类图；软件质量评估报告模块主要介绍了其架构设计、核心类图以及数据库设计。本章总体上介绍了系统的需求分析、系统总体以及模块设计，为后续系统实现做准备。

第四章　报告生成服务实现

4.1　执行引擎监控模块实现

依据第三章的设计，执行引擎监控模块主要包括在执行引擎初始化过程中的环境配置信息，在执行引擎运行过程中的单页面状态信息获取、测试执行路径获取，以及执行引擎完成测试后的测试流程图获取。在本小节中将依次介绍以上四类信息的获取的实现。

4.1.1　环境配置信息获取实现

```
@Override
public void preCrawling(CrawljaxConfiguration config)
        throws RuntimeException {
    //生成对应的Redis Key
    String partitionConfigName = String.format("%s_%s_config",
configModel.getTaskId(), configModel.getPartitionId());
    //获取当前环境配置信息
    BrowserConfigurationModel configurationModel =
configModel.getBrowserConfig();
    TestEnvironment environment = new TestEnvironment();

environment.setBrowserName(configurationModel.getBrowserType().getName(
));
    environment.setBrowserVersion(configurationModel.getVersion() !=
null ? configurationModel.getVersion() : "ANY");
    environment.setPlatform(configurationModel.getPlatform() != null ?
configurationModel.getPlatform().name() : Platform.ANY.name());
    //Redis存储操作
    jedis.set(partitionConfigName, JSON.toJSONString(environment));
    if (config.getMaximumRuntime() != 0) {
        jedis.pexpire(partitionConfigName, config.getMaximumRuntime() *
5);
    } else {
        jedis.pexpire(partitionConfigName, TimeUnit.DAYS.toMillis(1));
    }
}
```

图 4.1: 获取环境配置信息代码

如图4.1所示是 PartionInitPlugin 的部分代码实现，此处省略了该类成员的构造函数和成员变量以及相关的数据类。系统通过 PartionInitPlugin 是实现环境配置信息的获取。

PartionInitPlugin 类通过实现 Crawljax Plugin 中 PreCrawlingPlugin 接口，来完成对浏览器初始化状态的监控。PreCrawlingPlugin 会在执行引擎完成浏览器初始化，加载待测 Web 应用 URL 的之前，触发 preCrawling 方法。此时浏览器环境配置信息初始化已经完成，通过 CrawljaxConfiguration 获取浏览器环境配置信息模型 BrowserConfigurationModel 对象，从而获取浏览器环境配置信息。环境配置信息包括浏览器名称、浏览器版本以及操作系统类型。在完成浏览器环境配置信息后，按照表3.13中 Redis Key 设计格式生成的对应的 Key 存储到 Redis 缓存数据库中。

4.1.2　单页面状态信息获取实现

```
@Override
public void onNewState(CrawlerContext context, StateVertex newState) {
    CrawljaxConfiguration config = context.getConfig();
    String redisKey = String.format("%s_%s_%s", this.task,
this.partition, newState.getName());

    if
(config.getBrowserConfig().getBrowserType().equals(EmbeddedBrowser.Bro
wserType.REMOTE)) {
        DesiredCapabilities capabilities =
config.getRemoteWebDriverCapabilities();
        if (!capabilities.getBrowserName().equals(BrowserType.CHROME))
{
            this.sendRabbitmq(redisKey);
            return;
        }
    } else {
        if (!
config.getBrowserConfig().getBrowserType().equals(EmbeddedBrowser.Brow
serType.CHROME)) {
            this.sendRabbitmq(redisKey);
            return;
        }
    }

    LogEntries performanceLogs =
context.getBrowser().getWebDriver().manage().logs().get("performance")
;

    this.savePerformanceLogs(redisKey, performanceLogs, config);
    this.sendRabbitmq(redisKey);
}
```

图 4.2: 获取浏览器性能日志代码

单页面的状态信息获取包括浏览器性能日志获取、浏览器日志获取以及执行截图获取。系统通过 SaveLogPlugin、SaveConsolePlugin 和 SaveScreenshotPlugin 类实现单页面状态获取。这三个类都实现了 OnNewStatePlugin 接口，在执行引擎发现并探索新的页面状态时，开始该页面状态信息获取的业务流程。

SaveLogPlugin 类实现了浏览器性能日志获取的功能，部分代码实现如图4.2所示。SaveLogPlugin 通过 CrawlerContext 上下文对象获取执行引擎中的浏览器 WebDriver 驱动对象，然后通过 WebDriver 驱动对象获取加载当前浏览器页面的 performance 类型的日志。由于执行引擎会使用多种环境配置的浏览器执行测试，仅对 Chrome 类型的执行引擎进行日志获取，所以在获取日志之前对执行引擎进行筛选。在获取浏览器性能日志后，按照表3.13中 Redis Key 设计格式生成的 Key 存储到 Redis 缓存数据库中。

```
private void saveScreenshot(EmbeddedBrowser browser, String name,
StateVertex vertex, CrawljaxConfiguration config) {
    LOG.debug("Saving screenshot for state {}", name);
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    try {
        BufferedImage screenshot =
browser.getScreenShotAsBufferedImage(500);
        ImageIO.write(screenshot, "png", out);

        // 创建PutObjectRequest对象。
        String path = objectName+name+".png";
        PutObjectRequest putObjectRequest = new
PutObjectRequest(bucketName, path, new
ByteArrayInputStream(out.toByteArray()));
        ossClient.putObject(putObjectRequest);

        //保存路径到redis
        String url = "http://" + bucketName + ".oss-cn-
shanghai.aliyuncs.com/" + path;
        String redisKey = taskId + "_" + partitionId + "_" + name +
"_screenshot";
        jedis.set(redisKey, url);

        if (config.getMaximumRuntime() != 0) {
            jedis.pexpire(redisKey, config.getMaximumRuntime() * 5);
        } else {
            jedis.pexpire(redisKey, TimeUnit.DAYS.toMillis(1));
        }
    } catch (CrawljaxException | WebDriverException | IOException e) {
        LOG.warn("Screenshots are not supported or not functioning for
{}. Exception message: {}",browser, e.getMessage());
        LOG.debug("Screenshot not made because {}", e.getMessage(),
e);
```

图 4.3: 获取执行截图代码

SaveConsolePlugin 类使用类似的方法实现浏览器日志的获取，在获取执行引擎中的浏览器 WebDriver 驱动对象后，获取 browser 类型的日志。在日志获取结束后，存储到 Redis 中。

如图4.3所示的是 SaveScreenshotPlugin 类的部分代码实现，系统通过该类实现执行截图的获取和存储。SaveScreenshotPlugin 类通过 CrawlerContext 上下文对象获取执行引擎中的浏览器对象，调用 getScreenShotAsBufferedImage 方法获取当前页面状态截图。获取截图后将图片转换为 ByteArrayOutputStream 对象存储到 OSS 中，并将得到的 OSS 链接地址存储到 Redis 缓存数据库中。

```
//SaveConsolePlugin
private void saveBrowserLogs(String redisKey, LogEntries
browserLogs, CrawljaxConfiguration config) {
    redisKey = String.format("%s_browserlog", redisKey);
    List<String> logEntryJsons = new LinkedList<>();
    Iterator<LogEntry> it = browserLogs.iterator();
    while (it.hasNext()) {
        LogEntry entry = it.next();
        try {
            logEntryJsons.add(JSON.toJSONString(entry));
        } catch (Exception e) {
            LOG.error(e.getMessage(), e);
        }
    }
    try {
        jedis.set(redisKey.getBytes(StandardCharsets.UTF_8),

CompressUtils.compress(JSON.toJSONString(logEntryJsons).getBytes(S
tandardCharsets.UTF_8), 9));
        if (config.getMaximumRuntime() != 0) {

jedis.pexpire(redisKey.getBytes(StandardCharsets.UTF_8),
config.getMaximumRuntime() * 5);
        } else {

jedis.pexpire(redisKey.getBytes(StandardCharsets.UTF_8), (int)
TimeUnit.DAYS.toMillis(1));
        }
    } catch (Exception e) {
        LOG.error(e.getMessage(), e);
    }
 }

//SaveLogplugin
private void sendRabbitmq(String rediskey) {
    try {
        channel.basicPublish("", RabbitmqUtils.QUEUE_A,
MessageProperties.PERSISTENT_TEXT_PLAIN, rediskey.getBytes());
```

图 4.4: Redis 及 RabbitMQ 交互代码

单页面状态信息获取与存储中间件 Redis 和消息中间件 RabbitMQ 交互的实现如图4.4。单页面状态信息获取的执行引擎监控插件在获取数据后，会将数据存储到 Redis 中，以便后续模块的数据调用。使用 Redis 存储数据过程中，

由于日志信息过大导致 Redis 交互速度慢，考虑到系统的性能，使用系统定义的 CompressUtils 数据压缩工具对获取的日志信息进行压缩。CompressUtils 使用 java.util.zip 包实现。对日志信息压缩后，使用 Jedis API 将数据存储到 Redis 中。由于所有实现了 OnNewStatePlugin 的接口执行引擎插件在新页面状态出现时的执行顺序取决于执行引擎加载的顺序，所以在最后执行引擎加载的 SaveLogPlugin 类中进行与 RabbitMQ 的交互。执行顺序使得在 SaveLogPlugin 类进行交互前，所有的单页面状态信息已经全部存储到 Redis 中。SaveLogPlugin 类向 RabbitmqUtils 中定义的单页面分析的队列，发送监控模块完成数据、测试报告模块可以开始单页面分析业务流程的消息。

4.1.3　测试执行路径获取实现

如图4.5所示的 TestPathPlugin 的部分代码实现，主要展示了从 CrawlerContext 上下文对象中获取测试路径信息的方法。此处省略了该类中的成员变量和构造函数和成员变量、相关的数据类以及 Redis 交互。系统通过 TestPathPlugin 是实现测试执行路径的获取。

TestPathPlugin 通过实现 Crawljax Plugin 中 OnNewStatePlugin 的接口，来实现对新的页面状态的监控。当执行引擎发现并探索新的页面状态时，TestPathPlugin 开始处理获取该页面状态测试执行路径的业务逻辑。TestPathPlugin 通过访问 CrawlerContext 上下文对象获取当前执行引擎的测试流程图，然后通过访问该图找到到达当前页面状态的测试路径。在获取执行的测试路径的过程中，由于网页到达单个页面状态可能存在很多不同的执行路径，但是为了保证路径的唯一性，TestPathPlugin 在获取执行路径的过程中，从测试流程图中选择最短路径，保障自动化测试用例生成模块的使用。测试路径中包含执行引擎中的操作集合。对每一个 Eventable 操作行为进行处理，提取有效信息保存在测试路径节点类 TestNode 中。Eventable 包括一系列的输入操作、页面切换操作以及点击操作。对于输入操作，使用 Java 流处理，将操作转换为 FormInputModel 的列表。其他两类操作作为成员变量直接存储到 TestNode 类中。在生成测试路径 TestNode 列表后，按照表3.13Redis Key 设计格式生成的 Key 存储到 Redis 缓存数据库中。Redis 存储的实现代码与上文中的 SaveLogPlugin 类似。

```
@Override
public void onNewState(CrawlerContext context, StateVertex newState)
{
    StateFlowGraph graph = context.getSession().getStateFlowGraph();
    ImmutableList<Eventable> path =
graph.getShortestPath(graph.getInitialState(),newState);
    List<TestNode> testPath = new ArrayList<>();
    for (Eventable event: path){
        List<FormInputModel> formInputModels =
event.getRelatedFormInputs()
                                                    .stream()
                                                    .map(e->{

FormInputModel formInputModel = new FormInputModel();
formInputModel.setIdentification(e.getIdentification());
formInputModel.setInputType(e.getType());
formInputModel.setInputValues(e.getInputValues());
                                                  return
formInputModel;
                                                    })
                                                    .collect(Collect
ors.toList());

        TestNode testNode = new
TestNode(event.getSourceStateVertex().getName(),
event.getSourceStateVertex().getUrl(),
event.getTargetStateVertex().getName(),
event.getTargetStateVertex().getUrl(),
event.getEventType().toString(),
event.getIdentification().getHow().toString(),
event.getIdentification().getValue(),
                                        formInputModels,
                                        event.getRelatedFrame());
        testPath.add(testNode);
    }
    String redisKey = String.format("%s_%s_%s_testpath", this.task,
this.partition, newState.getName());
    this.saveTestPath(redisKey, testPath, context.getConfig());
}
```

图 4.5: 获取测试执行路径代码

4.1.4　测试流程图获取实现

测试流程图获取通过 SaveGraphPlugin 类实现 Crawljax Plugin 中 PostCrawlingPlugin 接口实现。当执行引擎完成测试执行后，SaveGraphPlugin 会检测到执行引擎状态并开始获取测试流程图。TestPathPlugin 通过访问 CrawlSession 对象获取执行引擎中的测试流程图对象，并将其拆解为页面状态节点列表和操作路径列表，按照表3.13Redis Key 设计格式生成的 Key 存储到 Redis 缓存数据库中。操作路径存储实现与上文测试路径获取实现中相似。页面状态节点保存其状态

编号、状态名、页面链接和 DOM 文档，前两者与操作路径中的编号对应，以供后续模块进行图的拼装。

4.2　测试报告模块实现

根据第三章对测试报告模块的设计，测试报告模块包括单页面分析、测试报告生成以及测试报告获取展示三个业务流程，本小节中将依次介绍这三个业务流程及其实现。

4.2.1　单页面状态分析顺序图

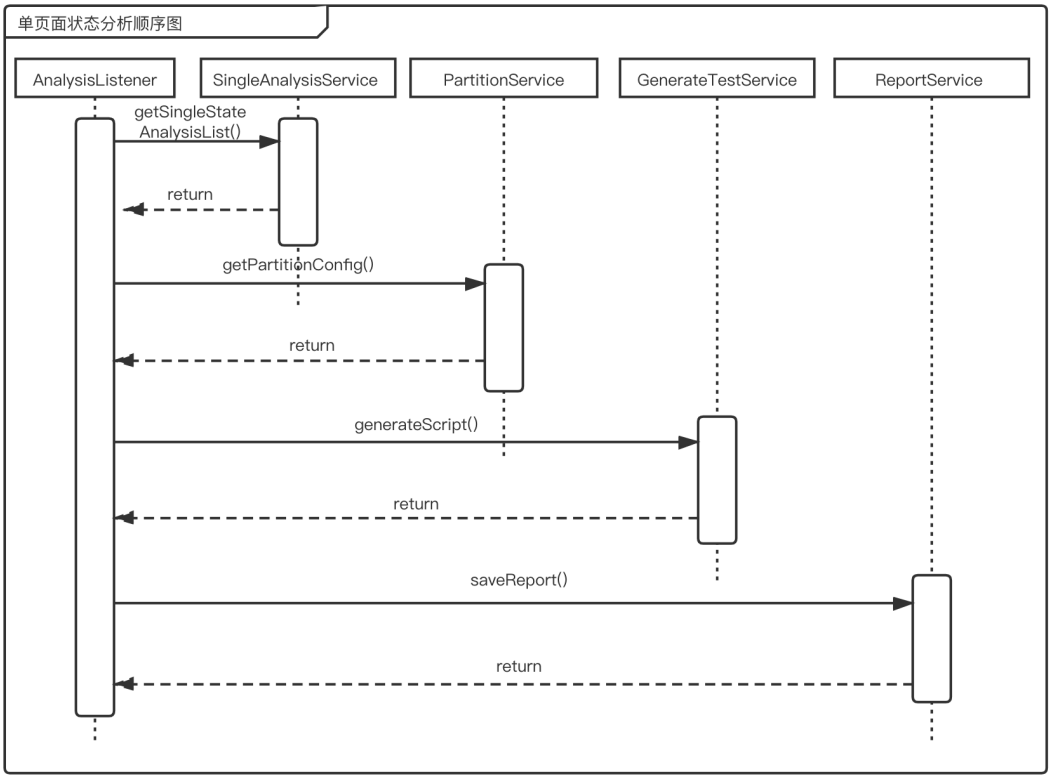

图 4.6: 单页面状态分析顺序图

单页面状态分析顺序图如图4.6所示。该流程主要负责单页面状态下的分析方法的调用。当一个新的页面状态在执行引擎中被探索，并且执行引擎监控模块完成单页面状态的信息收集后，AnalysisListener 类会收到 RabbitMQ 中开始单页面状态分析的消息，并开始单页面状态分析流程。AnalysisListener 类会调用 SingleAnalysis 类提供的 getSingleStateAnalysisList 方法获取单页面的缺陷信息列表，保存到单页面状态信息对象中，接着调用 PartitionService 类提供的

getPartitionConfig 方法获取当前页面的环境信息，绑定到单页面状态信息对象中，完成单页面缺陷列表的生成。然后开始针对缺陷生成自动化测试用例，调用 GenerateTestService 类提供的 generateScript 生成自动化测试用例后，获取生成的自动化测试用例的 OSS 文件路径，也存储到单页面状态信息对象中。在生成完整单页面状态信息对象后，会调用 ReportService 类提供的 saveReport 方法，将生成单页面分析的结果存储到 Redis 中。

4.2.2　单页面状态分析实现

```
@RabbitHandler
@RabbitListener(queues = RabbitmqConfig.QUEUE_A)
public void process(String stateKey) {
     ......   //performanceLogList等初始化，stateKey预处理
    byte[] performanceLogCompress =
byteRedisTemplate.opsForValue().get(stateKey + Common.PERFORMANCELOG);
    if (Objects.nonNull(performanceLogCompress)) {
        try {
            performanceLogList.addAll(JSON.parseArray(new
String(CompressUtils.uncompress(performanceLogCompress),
StandardCharsets.UTF_8), String.class)); //性能日志的获取
        } catch (Exception e) {
            LOG.error(e.getMessage(), e);
        }
    }
    ...... //浏览器日志browserLogList、测试路径testPath、截图链接
screenshotPath的获取
    StateParam stateParam = new StateParam();
    ...... //stateParam分析参数初始化、赋值过程
    List<StateVulnerability> currentStateVulnerability =
singleAnalysisService.getSingleStateAnalysisList(stateParam);
    currentStateVulnerability =
partitionService.getPartitionConfig(stateKey, currentStateVulnerability);
    currentStateVulnerability =
generateTestService.generateScript(stateKey,testPath,currentStateVulnerab
ility);
    reportService.saveReport(stateKey, currentStateVulnerability,
testPath, screenshotPath);
    ...... //异常处理以及删除缓存数据库中的数据
}
```

图 4.7: AnalysisListener 类调用实现代码

单页面状态分析流程的调用，主要有 AnalysisListener 类的 process 方法来完成，其部分代码实现如图4.7所示。process 方法使用 Spring 中的 RabbitMQ 相关注解绑定到单页面分析相关的消息队列中，当执行引擎探索新的页面状态后，process 方法会从队列中消费单页面状态分析的信息，开始单页面分析流程调用。首先通过消息队列传入的参数 stateKey 的解析，获取该页面状态相关的信息存

储在 Redis 中的 Key。接着，根据 Key 值获取相关信息，图中仅展示了性能日志的获取方法，其他信息还包括浏览器日志、测试执行路径、截图链接路径。通过 Key 得到压缩的性能日志信息，然后使用 CompressUtils 工具类对获取的数据进行解压，得到日志对象。然后将获取的单页面分析数据存储到 StateParam 对象中，方便调用。接下来依次调用，SingleAnalysisService 类的 getSingleStateAnalysisList 方法获取当前页面状态的分析结果列表，PartitionService 类的 getPartitionConfig 方法获取当前页面状态浏览器配置信息，GenerateTestService 类的 generateScript 方法获取执行到当前页面状态的自动化测试用例。最后，调用 ReportService 类的 saveReport 方法，将当前分析结果存储到 Redis 存储中，以供最终测试报告生成使用。

4.2.3　测试报告生成顺序图

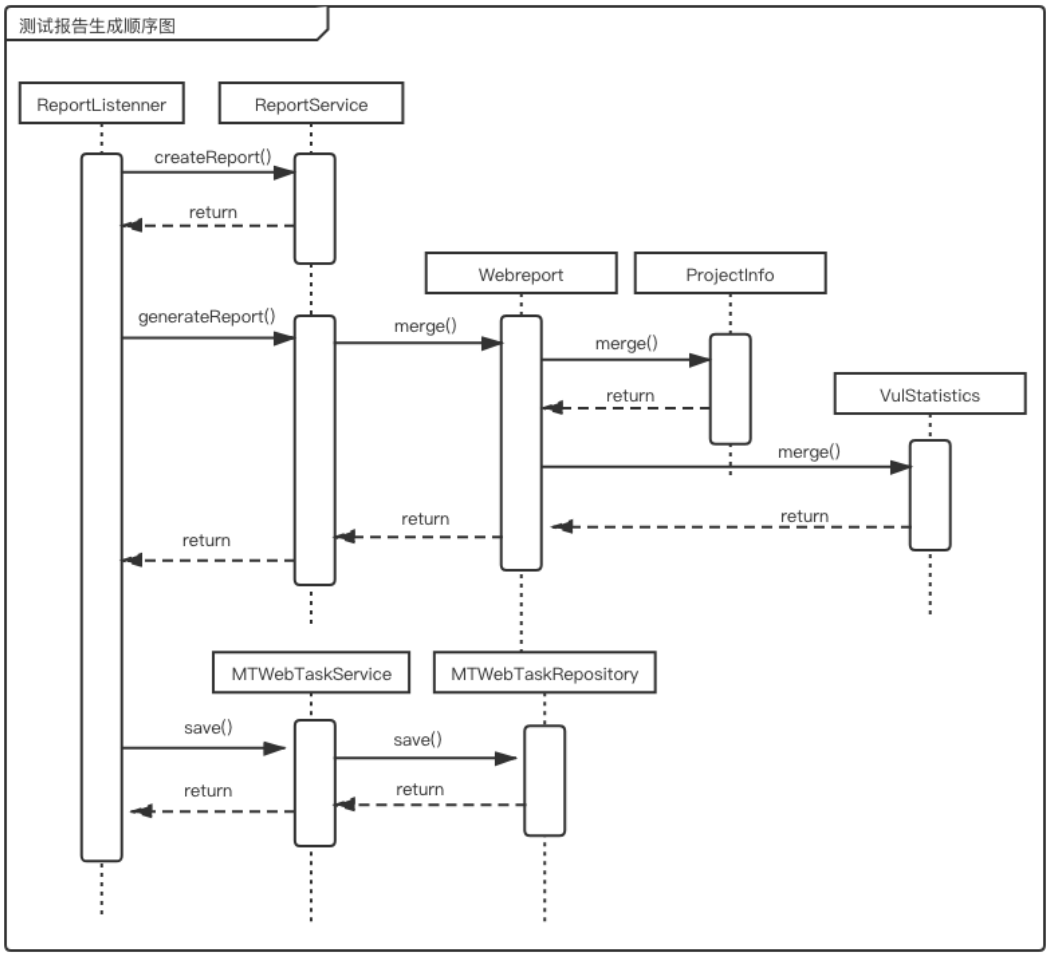

图 4.8: 测试报告生成顺序图

测试报告生成流程的顺序图如图4.8所示。测试报告生成流程中主要包括两次调用过程，测试报告的初始化以及测试报告的生成。测试报告生成流程主要通过消费 RabbitMQ 中的消息来进行调用。当一个新的测试任务被发布后，ReportListener 类接收到来自 RabbitMQ 消息队列中初始化测试报告的消息，调用 ReportService 类提供的 createReport 方法，初始化测试报告基本信息，并存储到 Redis 中。当执行引擎完成测试任务后，ReportListener 类会接收到来自 RabbitMQ 消息队列中生成测试报告的消息，，调用 ReportService 类提供的 gererateReport 方法，获取测试报告。generateReport 方法会从 Redis 中获取初始化的测试报告，以及单页面状态分析的结果，调用获取的 WebReport 对象的 merge 方法，将单页面分析结果添加到该对象中。WebReport 对象会调用自身成员变量 ProjectInfo 以及 VulStatistics 对象的 merge 方法，完成测试报告的生成工作。

ReportListener 在获得返回的 WebReport 对象后，会通过调用 MTWebTaskService 以及 MTWebTaskRepository 中对应的 save 方法，将其保存在 MongoDB 数据库中对应的 MTWebTask 文档对象中，以供测试报告展示业务使用。

4.2.4　测试报告生成实现

```
@RabbitHandler
@RabbitListener(queues = RabbitmqConfig.QUEUE_LIFETIME)
public void resolveTaskMessage(String message){
          TaskCommand taskCommand = JSONObject.parseObject(message,
TaskCommand.class);
        if (taskCommand.getStatus() == TaskCommand.TaskStatus.START){
            reportService.createReport(taskCommand.getTaskId());
        }else {
            WebReport webReport =
reportService.generateReport(taskCommand.getTaskId());

            MTWebTask mtWebTask =
mtWebTaskService.findByTaskId(taskCommand.getTaskId());
            mtWebTask.setReport(webReport);
            mtWebTask.setStatus(CeleryStatus.FINISHED);
            ScoreModel scoreModel =
acceptanceScoreService.analyseAcceptanceScore(mtWebTask);
            mtWebTask.setScoreModel(scoreModel);
            mtWebTask.setUpdatedAt(new Date());
            mtWebTaskService.save(mtWebTask);
        }
}
```

图 4.9: ReportListener 类调用实现代码

如图4.9所示的是 ReportListener 类中的部分调用实现代码，主要展示核心方法 resolveTaskMessage，省略了方法中的一些异常处理。resolveTaskMessage 方法

使用 Spring 中的 RabbitMQ 相关注解绑定到测试任务生命周期相关的消息队列中，监听测试报告初始化以及测试报告生成的消息。该消息队列中的详细会携带一个 TaskCommand 类封装的命令对象，其中的 TaskStatus 属性表示了这条信息的目的，taskId 属性表示这条消息对应的测试任务编号。通过判断 TaskStatus 属性进行不同的调用，进行相应的测试报告初始化和测试报告生成。

```
public void createReport(String taskId) {
    MTWebTask task = mtWebTaskService.findByTaskId(taskId);
    WebReport webReport = new WebReport();
    ProjectInfo initProjectInfo = new ProjectInfo();
    initProjectInfo.setStartTime(LocalDateTime.now());
    initProjectInfo.setProjectName(task.getUrl());
    initProjectInfo.setUserName(task.getUser());
    webReport.setProjectInfo(initProjectInfo);
    redisTemplate.opsForValue().set(taskId, webReport);
}

public WebReport generateReport(String taskId) {
    JSONObject webReportJson = (JSONObject)
redisTemplate.opsForValue().get(taskId);
    WebReport webReport = webReportJson.toJavaObject(WebReport.class);
    String taskVulnerabilityList = taskId + Common.VULNERABILITY;

    List<StateVulnerability> stateVulnerabilities =
Objects.requireNonNull(redisTemplate.opsForList()
            .range(taskVulnerabilityList, 0, -1))
            .stream()
            .flatMap(i -> {
                if (i instanceof JSONArray) {
                    return ((JSONArray)
i).toJavaList(StateVulnerability.class).stream();
                } else {
                    return Stream.of(((JSONObject)
i).toJavaObject(StateVulnerability.class));
                }
            })
            .collect(Collectors.toList());
     ......
    webReport.merge(stateVulnerabilities, stateCount);
     ......
    return webReport;
}
```

图 4.10: ReportService 类测试报告生成服务实现代码

如图4.10所示的是 ReportService 类中测试报告生成相关服务的实现代码，图中省略了部分实现以及异常处理，包括 createReport、generateReport 两个方法。createReport 方法负责测试报告的初始换，通过输入的测试任务编号从数据库中获取相关信息，然后将这些信息初始化到 WebReport 对象中，最后使用任务编号为 Key，将生成的 WebReport 对象存储到 Redis 中。generateReport 方法负责在任务结束后测试报告的生成。首先通过任务编号获取初始化的测试报告，然

后通过预先设置好的 Redis Key 获取该测试任务的单页面分析结果，图中展示了缺陷列表的处理方式，使用 Java 流处理将 Redis 中的数据转换为缺陷列表对象。此外还对测试任务中的页面状态进行统计。最后使用获取的信息调用 WebReport 对象的 merge 方法，生成完成的测试报告对象返回。

```
//VulStatistics
public void merge(List<StateVulnerability> vulnerabilityList) {
    vulnerabilityList = vulnerabilityList
                                    .stream()
                                    .filter((StateVulnerability s)-
>s.getBugDetailList() != null )
                                    .collect(Collectors.toList());
    this.categoryStat = Stream.concat(
            vulnerabilityList
                    .stream()
                    .flatMap(i -> i.getBugDetailList().stream())
                    .collect(Collectors.groupingBy(BugDetail::getCategory,
Collectors.counting()))
                    .entrySet()
                    .stream()
                    .map(e -> {
                        StatisticsNode node = new StatisticsNode();
                        node.setName(e.getKey());
                        node.setValue(e.getValue().intValue());
                        return node;
                    }), this.categoryStat.stream())
            .collect(Collectors.groupingBy(StatisticsNode::getName,
Collectors.summingInt(StatisticsNode::getValue)))
            .entrySet()
            .stream()
            .map(e -> {
                StatisticsNode node = new StatisticsNode();
                node.setName(e.getKey());
                node.setValue(e.getValue());
                return node;
            }).collect(Collectors.toList());//缺陷类型的统计
   this.riskLevelStat =......    //缺陷级别的统计
   this.platformStat = ......    //运行平台环境的统计
}
//ProjectInfo
public void merge(int bugCount, long stateCount) {
   ......
}
```

图 4.11: WebReport 生成实现代码

如图4.11所示的是 WebReport 对象的生成实现，WebReport 的 merge 方法主要包括缺陷信息的统计、测试任务基本信息的更新以及缺陷列表的合成。缺陷信息的统计由图中 VulStatistics 类的 merge 方法实现，包括缺陷类型的统计、缺陷级别的统计以及运行平台环境的统计。图中仅展示了缺陷类型统计的实现，首先将不存在缺陷的页面状态从带分析的页面状态列表中去除，然后使用 Java 流处理对页面状态列表中的缺陷进行提取，列举所有的缺陷类型，并以此为维度，

统计所有缺陷类型的缺陷数量。ProjectInfo 类的 merge 方法是实现了测试任务基本的更新，包括任务执行时间、页面状态数、缺陷数的更新。缺陷列表的合成使用 Java List 中的工具方法实现。

4.2.5　测试报告展示顺序图

测试报告展示顺序图如图4.12所示。当 Web 应用自动化测试系统前端需要调用完整测试报告时，会通过 AutoTestingController 类提供的接口获取。AutoTestingController 类会调用 Service 层的 MTWebTaskService 类提供的 findByCelerySessionId 方法。接着，MTWebTaskService 类会调用数据层的 MTWebTaskRepository 类提供的 findByCelerySessionId 方法访问 MongoDB 数据库获取相应的测试报告并返回。

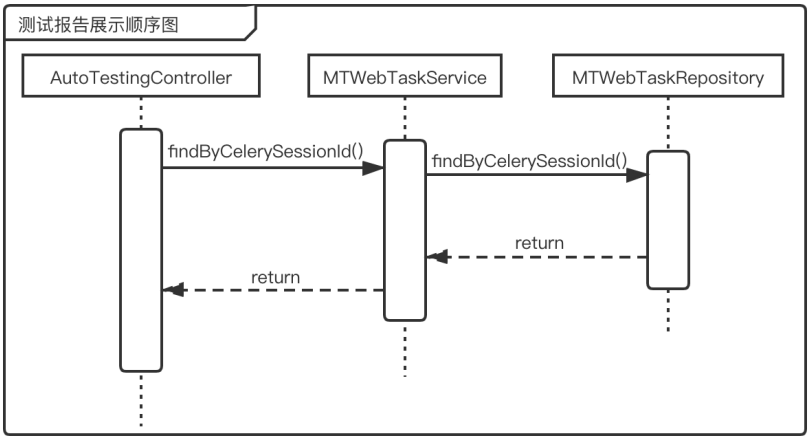

图 4.12: 测试报告展示顺序图

4.2.6　测试报告展示实现

测试报告展示包括测试报告总览、缺陷列表展示以及缺陷详情展示三个方面的展示，分别对应了第三章系统需求中的查看测试结果统计、查看测试结果详情以及查看缺陷详情这三个用例。

测试报告总览界面如图4.13所示，页面上半部分是应用测试报告的基本信息，包括应用 URL、提交用户、提测时间、测试耗时、测试中执行的状态数以及测试发现的缺陷数。这部分主要展示待测应用的基本信息以及测试报告情况总览。页面下半部分是测试报告中的统计信息，通过平台分类、缺陷类型以及缺陷级别这三个维度对测试中发现的缺陷进行统计。测试报告总览界面主要目的是展示测试报告的整体信息，使用户对测试报告有总体上的认知。

图 4.13: 测试报告总览界面

缺陷列表展示界面如图4.14所示。该页面主要展示测试中发现的所有缺陷，展示的信息包括单页面状态名、缺陷类型、缺陷级别、操作系统、浏览器。在页面的上方提供了对所有缺陷的筛选模块，用户可以通过缺陷类型、缺陷级别以及操作系统对展示的缺陷进行筛选，方便用户快速查看其关心的缺陷。该筛选模块与测试报告总览界面的统计图相关联，用户可以通过直接点击统计图中的一个饼图块跳转到相应筛选下的缺陷列表展示界面。

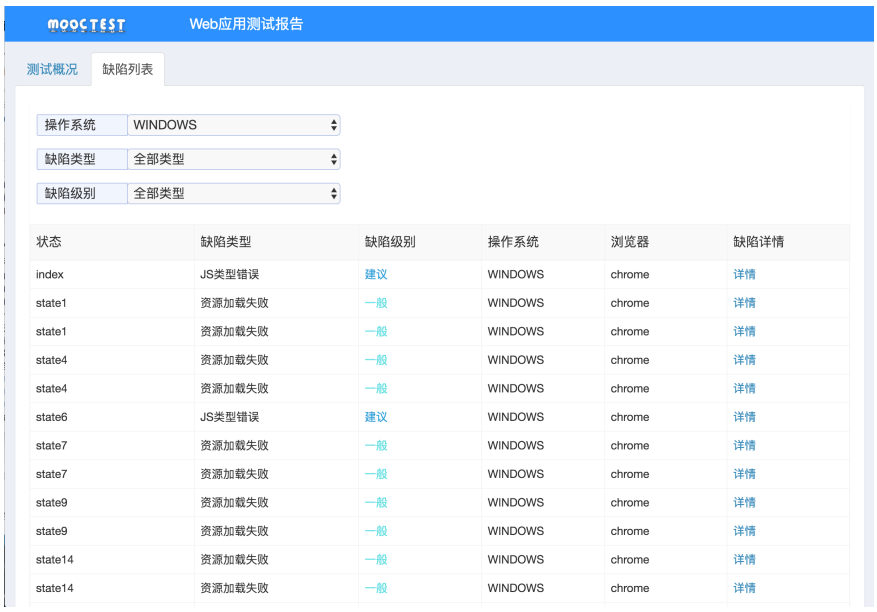

图 4.14: 缺陷列表展示界面

缺陷详情展示界面如图4.15所示。该页面主要分为缺陷概述和缺陷上下文两个部分。缺陷概述主要展示缺陷的一些基本信息，其中包括缺陷类别、缺陷所在页面路由、执行该页面测试的操作系统以及浏览器。缺陷上下文主要展示了缺陷的详细信息以及执行到该缺陷所在页面的操作步骤，缺陷的详细信息包括缺陷发生的测试时间、缺陷所在的问题资源、严重等级、当前单页面状态名、当前单页面路由、前序单页面状态名、前序单页面路由以及缺陷日志。操作步骤展示了从待测 Web 应用执行到当前页面状态所有的执行步骤，用户可以通过该操作步骤自主在浏览器中复现该缺陷。在缺陷详情页面中提供下载复现该缺陷的自动化测试用例下载，用户也可以通过下载自动化测试用例来复现该缺陷。

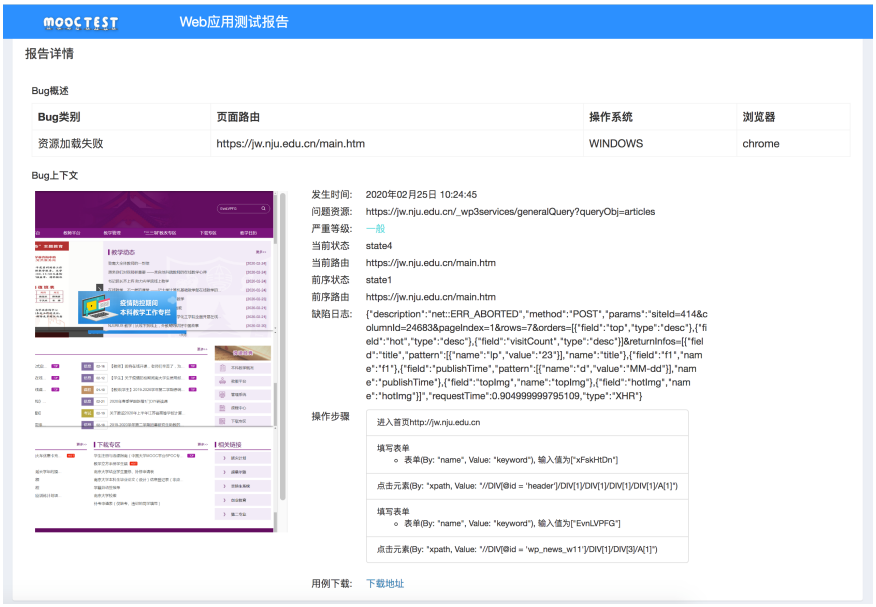

图 4.15: 缺陷详情展示界面

4.3　自动化测试用例生成模块实现

4.3.1　自动化测试用例生成模块顺序图

自动化测试用例生成模块主要的业务流程是通过测试报告模块单页面状态分析服务的调用，根据执行引擎监控模块获取的浏览器配置信息和执行路径信息生成自动化测试用例。生成自动化测试用例的模块调用顺序图如图4.16所示。

从图中可以看出，整个自动化测试用例生成模块由测试报告模块的 AnalysisListener 调用。AnalysisListener 调用自动化测试报告生成模块中的 GenerateTestService 类提供的 generateScript 方法，生成自动化测试用例并存储到单页面数据类中。GenerateTestService 类会调用 SeleniumScriptFactory 工厂类提供的

getSeleniumScript 方法，获取相应的 SeleniumScript 对象，即顺序图中的 JavaSeleniumScript 对象。

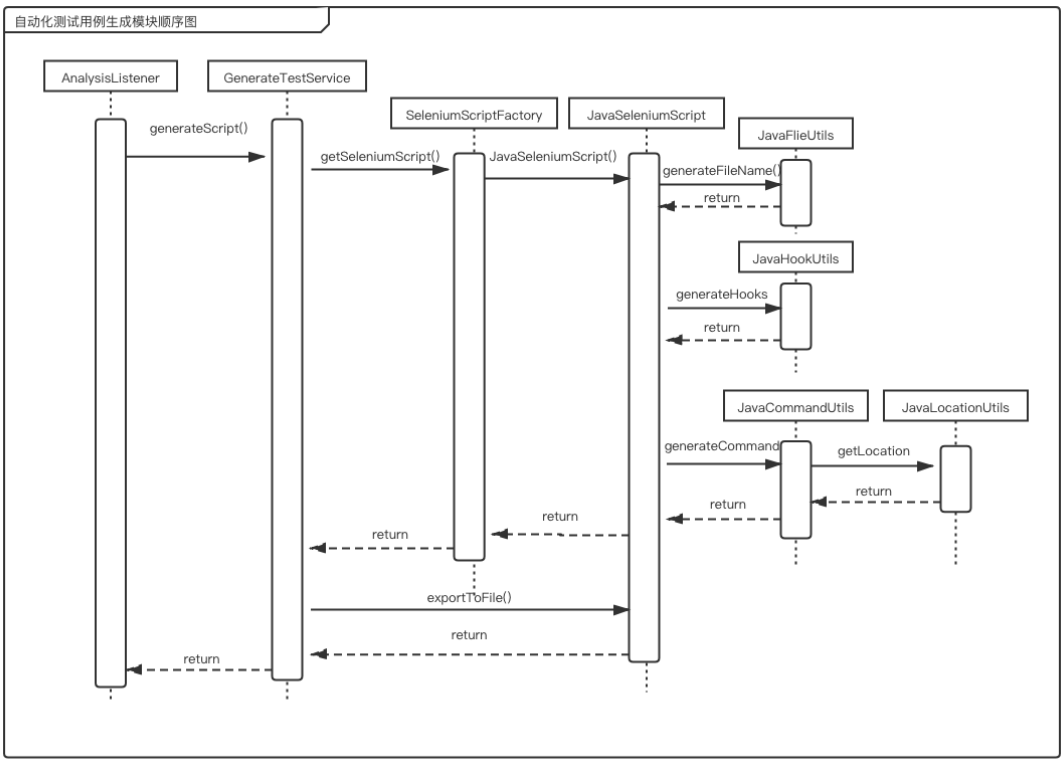

图 4.16: 生成自动化测试用例顺序图

SeleniumScriptFactory 创建 JavaSeleniumScript 对象前，会根据输入的浏览器配置信息以及执行路径信息调用 JavaSeleniumScript 对象的有参构造函数，自动化测试用例生成过程由该构造函数完成。JavaSeleniumScript 的构造函数通过调用 JavaFlieUtils、JavaHookUtils、JavaCommandUtils 提供的方法生成自动化测试用例的各个组成部分。调用 JavaFileUtils 提供的 generateFileName 生成 Java 类名。调用 JavaHookUtils 中的 declareDependencies、declareParameters、before、after 方法完成自动化测试用例中固定部分的生成，其中 before 方法需传入浏览器配置参数，获取相应的浏览器驱动 WebDriver 相关代码。调用 JavaCommandUtils 提供的不同的 Selenium 命令生成方法，生成自动化测试用例中 Selenium 命令部分。在生成 Selenium 命令的过程中，使用 JavaLocationUtils 提供的元素定位方法，完成网页元素 WebElement 的定位。

4.3.2　测试用例生成流程实现

自动化测试用例生成模块的实现包括测试用例生成流程的实现以及生成测试用例工具类的实现。

```
//GenerateTestService
public List<StateVulnerability> generateScript(String stateName,
List<TestNode> testPath, List<StateVulnerability> vulnerabilities){
        if (vulnerabilities == null || vulnerabilities.size() == 0) {
            return vulnerabilities;
        } else {
            SeleniumScriptFactory seleniumScriptFactory = new
SeleniumScriptFactory();
            vulnerabilities.stream().forEach(e -> {
                String driverType = e.getEnvironment().getBrowserName();
                SeleniumScript seleniumScript =
seleniumScriptFactory.getSeleniumScript("java", stateName, testPath,
driverType);
                String ossPath = seleniumScript.exportToFile();
                e.setSeleniumOSSLink(ossPath);
            });
            System.out.println(vulnerabilities);

            return vulnerabilities;
        }
}

//SeleniumScriptFactory
public SeleniumScript getSeleniumScript(String type, String stateName,
List<TestNode> testPath, String driverType){
    SeleniumScript seleniumScript = null;
    switch (type) {
        case "java":
            seleniumScript = new JavaSeleniumScript(stateName, testPath,
driverType);
            break;
          ……
    return seleniumScript;
}
```

图 4.17: 测试用例生成流程代码

测试用例生成流程的实现是 AnalysisListener 类调用到获得返回值的过程，主要的代码实现如图4.17中所示，包括 GenerateTestService 类和 SeleniumScript-Factory 类，此处展示生成流程核心代码。

generateScript 方法是 GenerateTestService 类提供外部调用的服务接口，通过输入状态名称、执行路径信息和状态缺陷列表，返回带自动化测试用例的缺陷列表。如果输入的状态缺陷列表为空，generateScript 方法会直接返回。如果非空，方法使用 Java 流处理方法遍历状态缺陷列表，对每个状态缺陷生成自动化测试

用例。generateScript 方法新建 SeleniumScriptFactory 类中的 getSeleniumScript 方法，按照输入的参数生成 SeleniumScript 接口对象。SeleniumScript 提供了导出自动化测试用例到 OSS 文件，并返回 OSS 链接的 exportToFile 方法，生成用例文件的 OSS 链接会赋值给 seleniumOSSLink 属性。

SeleniumScriptFactory 类提供获取自动化测试用例对象 getSeleniumScript 方法。测试用例生成流程中使用工厂方法来创建不同语言的自动化测试用例。该方法通过输入的语言类型不同，用输入的其他参数调用不同的自动化测试用例对象类的构造方法，来创建不同语言的自动化测试用例对象。

4.3.3　工具类实现

生成测试用例工具类的实现包括具体的 SeleniumScript 类的实现以及相应代码生成工具类的实现。JavaSeleniumScript 类的部分实现如图4.18所示, 包括类构造函数和 generateTest 方法，省略了异常处理部分代码。

```
public JavaSeleniumScript(String stateName, List<TestNode> testPath,
String driverType){
    this.fileName = JavaFileUtils.generateFileName(tmp[2]);
    this.bucketName = tmp[0]+tmp[1];
    this.dependencies = JavaHookUtils.declareDependencies();
    this.classStart = Stream.of(new ScriptCommand(0, "public class " +
stateName + "Test {")
    ).collect(Collectors.toList());
    this.classEnd = Stream.of(new ScriptCommand(0, "}")
    ).collect(Collectors.toList());
    this.parameters = JavaHookUtils.declareParameters();
    this.before = JavaHookUtils.before(driverType);
    this.after = JavaHookUtils.after();
    this.test = generateTest(testPath);
}
```

图 4.18: 测试用例生成流程代码

JavaSeleniumScript 类实现了 Java 自动化测试用例的初始化以及装配工作。Java 自动化测试用例被 JavaSeleniumScript 类划分为引用声明、类起始、类结束、类参数、类初始化、类关闭以及 Selenium 命令七个部分。

引用声明是 dependencies 在 Java Selenium 测试用例中需要 import 导入的包，使用 JavaHookUtils 工具类提供的 declareDependencies 方法生成。类起始 classStart 和类结束 classEnd，包括代码文件中的类声明以及类代码块结束后的终止符。类参数 parameters 是指 Java Selenium 类的成员变量声明，包括 WebDriver、JavascriptExecutor，使用 JavaHookUtils 工具类提供的 declareParameters 方法生成。类初始化 before 是对 Java Selenium 类的成员变量进行初始化，通过

输入 driverType 初始化不用类型的 WebDriver，使用 JavaHookUtils 工具类提供的 before 方法生成。类关闭 after 是结束 Selenium 执行后的连接关闭部分，关闭 Webdriver，使用 JavaHookUtils 工具类提供的 after 方法生成。

Selenium 命令 test 部分是自动化测试用例的核心部分，通过 generateTest 方法实现初始化，如图4.19所示。根据定义的测试节点 TestNode 类，Selenium 命令主要包括三种类型的命令，在代码中的实现依次是 iframe 切换命令、输入命令、点击命令。iframe 切换命令通过输入要切换的 iframe 的 id，使用 JavaCommandUtils 工具类提供 switchToframe 方法生成。由于输入命令不一定会导致新的页面状态的出现，所以 TestNode 可能包含了多个输入命令，使用 Java 流处理的方式遍历所有输入命令，输入命令使用 JavaCommandUtils 工具类提供一系列输入命令方法生成。generateTest 方法使用 Java 反射的方法，获取 JavaCommandUtils 的类对象，通过方法名调用不同的输入命令生成方法，生成输入命令。点击命令包括 click 和 hover，通过 TestNode 的 EventType 反射调用 JavaCommandUtils 工具类中的方法。另外构造函数还根据输入的状态名称，生成了文件名和 OSS 文件系统中的 Bucket 名。

```
private List<ScriptCommand> generateTest(List<TestNode> testPath) {
    if (Objects.nonNull(testPath) && !testPath.isEmpty()) {
     ......
        Class CommandUtilsClazz =
Class.forName("edu.nju.ise.analysis.utils.generate.JavaCommandUtils");
        for (TestNode e : testPath) {
            if (Objects.nonNull(e.getRelatedFrame()) && !
e.getRelatedFrame().equals("")) {

commands.add(JavaCommandUtils.switchToframe(e.getRelatedFrame()));
            }
            if (Objects.nonNull(e.getRelatedFormInputs()) && !
e.getRelatedFormInputs().isEmpty()) {
                for (FormInputModel input:e.getRelatedFormInputs()){
commands.addAll((List<ScriptCommand>)CommandUtilsClazz.getMethod("input" +
input.getInputType().toString().toLowerCase(), FormInputModel.class)
                            .invoke(null, input));
                }
            }
            commands.add((ScriptCommand)
CommandUtilsClazz.getMethod(e.getEventType(),TestNode.class)
                    .invoke(null, e));
        }
    ......
  }
```

图 4.19: Selenium 命令生成代码

JavaSeleniumScript 类是使用了 ScriptCommand 类，该类存储 Selenium 测试测试用例的一行代码，其属性包括 statement 和 level，statement 存储代码信息，

level 存储代码信息前的缩进格式。该类实现了 JavaSeleniumScript 类中的内容保存成 OSS 文件的格式规范。

```
public class JavaCommandUtils {

    private static Class locationUtilsClazz;

    static {
            locationUtilsClazz =
Class.forName("edu.nju.ise.analysis.utils.generate.JavaLocationUtils");
          }
    private static String getLocation(String how, String value){
        String location = (String)
locationUtilsClazz.getMethod("locationBy" +  how,
String.class).invoke(null, value);
       return location;
    }
     public static List<ScriptCommand> inputselect(FormInputModel input){
    List<ScriptCommand> result = new ArrayList<>();
           String location =
getLocation(input.getIdentification().getHow().toString().toLowerCase(),
                input.getIdentification().getValue());
        String statement = "WebElement dropdown = driver.findElement(" +
location + ");";
        result.add(new ScriptCommand(2, "driver.findElement(" + location +
");"));
        result.add(new ScriptCommand(2, "{"));
        result.add(new ScriptCommand(3, statement));

        Iterator<InputValue> iterator = input.getInputValues().iterator();
        String sendValue = "";
        if (iterator.hasNext()) {
            sendValue = iterator.next().getValue();
        }
        location = "By.xpath(\"//option[. = '" + sendValue + "']\")";
        statement = "dropdown.findElement(" + location + ").click();";
        result.add(new ScriptCommand(3, statement));

        result.add(new ScriptCommand(2, "}"));
       return result;
   }
}
```

图 4.20: JavaCommandUtils 工具类实现代码

相应代码生成工具类生成类包括 JavaFileUtils、JavaHookUtils、JavaCommandUtils 以及 JavaLocationUtils。JavaFileUtils、JavaHookUtils 工具类主要负责 Selenium 测试用例中的一些静态代码块、类名的实现。JavaCommandUtils、JavaLocationUtils 工具类主要负责动态代码块的实现。

JavaCommandUtils 工具类的部分实现代码如图4.20中所示，仅展示了获取元素对象定位的 getLocation 方法以及一种处理输入命令的 inputselect 方法。getLoation 方法主要通过 Java 反射机制调用 JavaLocationUtils 工具类中的一系列的

元素定位方法。在 JavaCommandUtils 类加载的过程中会执行图中的静态语句块 static，将 JavaLocationUtils 类加载到类静态变量 locationUtilsClazz 中，此种处理方式参考了 Java 类的生命周期模型。JavaCommandUtils 工具类中的其他方法统一调用此方法获取网页元素的定位方法实现。inputselect 方法是处理网页下拉选择框元素的方法。该方法首先通过 getLocation 方法生成获取网页下拉框元素对象 dropdown 的 Selenium 命令。然后通过访问 input 值列表获取需要选择的选项在下拉框元素中的序号，然后生成选择该选项的 Selenium 命令。其他的输入命令生成方法也有类似的实现过程。

```
public class JavaLocationUtils {

    public static String locationByxpath(String value){
        return "By.xpath(\""+ value +"\")";
    }

    public static String locationByname(String value){
        return "By.name(\""+ value +"\")";
    }

    public static String locationByid(String value){
        return "By.id(\""+ value +"\")";
    }
     ......
}
```

图 4.21: JavaLocationUtils 工具类实现代码

JavaLocationUtils 工具类的部分实现代码如图4.21中所示，JavaCommandUtils 中的 getLocation 方法通过方法名访问这些元素定位方法，这些方法的名称由 locationBy 和相应的元素定位方法构成，例如使用 xpath 定位网页元素的方法名为 locationByxpath，这种命名方式便于通过 Java 反射机制调用，避免复杂判断，简化代码实现。

4.4　软件质量评估报告模块实现

4.4.1　软件质量评估报告模块顺序图

软件质量评估报告模块主要的业务流程是实现评估报告的展示以及功能、性能、健壮性、兼容性、稳定性详细报告的展示，同时为了能够更加直观地展示功能和兼容性的详细报告，软件质量评估报告模块通过执行引擎监控模块获取了测试任务的测试流程图。

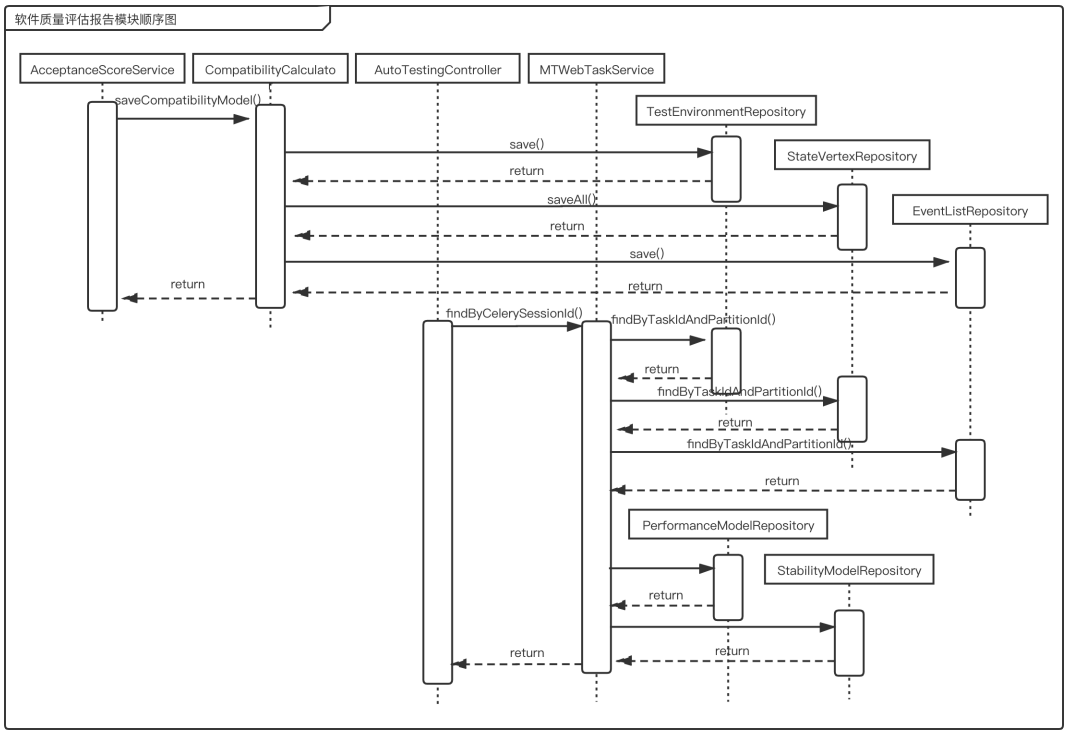

图 4.22: 软件质量评估报告模块顺序图

软件质量评估报告模块的顺序图如图4.22所示，包括测试流程图的获取以及质量评估报告数据的获取两个业务流程。测试报告模块接收到测试执行结束信号，整合缺陷列表和测试报告后，调用 AcceptanceScoreService 类提供的接口，开始进行测试流程图数据的获取和存储。获取和存储的业务逻辑在 CompatibilityCalculator 类中，通过调用 saveCompatibilityModel 方法实现。由于功能报告与兼容性报告共用测试流程图信息，而兼容性报告需要执行引擎在不同浏览器环境下的测试流程图，因此使用兼容性评估结果类统一进行存储。在获取测试流程图信息后，CompatibilityCalculator 类分别调用 TestEnvironmentRepository、StateVertexRepository、EventListRepository 类对测试流程图中的浏览器环境列表、网页状态节点列表以及操作路径列表进行存储。其余维度的详细报告计算以及存储过程通过调用分析服务实现。

系统通过模块提供的 AutoTestingController 类接口实现获取质量评估报告的数据的业务流程。MTWebTaskService 类提供质量评估报告数据整合的业务逻辑层接口，通过调用数据层接口从 MongoDB 数据库获取质量评估报告以及各个维度的详细报告。

4.4.2　软件质量评估报告模块实现

软件质量评估报告模块实现包括评估报告前端展示的实现以及业务逻辑部分的实现。前端展示部分展示了软件质量评估报告整体结果以及功能、性能、健壮性、兼容性、稳定性五个维度的详细报告。

图 4.23: 软件质量评估报告界面

软件质量评估报告界面如图4.23所示，用户通过访问测试报告的多维评估标签页即可查看软件质量评估报告。该部分主要展示评估报告的整体结果，使用雷达图的方式展示各个维度的评分情况。该页面的主要交互目的是向用户提供软件质量评估报告的综合情况，同时通过该页面雷达图中的链接可以访问各个维度的详细切面报告。

图 4.24: 性能评估详细报告界面

性能评估报告界面如图4.24所示，采用性能问题列表来展示被测应用可能存在的性能问题，对软件性能维度进行评估。Web 应用的性能问题主要集中在对

资源的请求过程，包括接口、文档、图片的请求，请求响应时间过程会导致软件的性能问题。性能问题列表展示内容包括请求类型、种类、响应时间、所在页面状态、被请求的资源以及资源的大小。健壮性评估详细报告以及稳定性详细评估报告也采用相应问题列表的形式进行展示。健壮性问题有被测应用存在的多种类型的缺陷导致，包括资源加载问题，关键信息未加密、JavaScript 错误等，缺陷列表展示内容包括健壮性缺陷类型、所在页面状态、问题资源、描述信息以及页面截图。稳定性问题主要有页面断链导致，问题列表展示内容包括页面状态名、页面链接以及截图。

图 4.25: 功能评估详细报告界面

功能评估详细报告界面如图4.25所示，Web 应用的功能复杂度主要体现在网页的状态数以及用户可交互的操作数。功能评估报告主要展示被测应用在测试过程中执行到的所有页面状态和可交互情况，页面上半部分的表格展示了两者的统计数，下半部分展示了功能导航图，主要体现了测试中执行服务模拟人工操作与被测应用的交互情况。功能导航图引入 ECharts[1] 实现，提供了高可视化的评估报告。兼容性评估详细报告界面，使用不同浏览器环境下测试执行的测试流程图进行展示，实现方式与功能评估详细报告中功能导航图类似，展示了在不同浏览器下模拟人工与被测应用交互的差异。

业务逻辑部分的实现包括测试流程图的获取、存储实现以及展示过程中质量评估报告数据的获取。测试流程图的获取、存储通过 CompatibilityCalculator 类实现主要的业务逻辑，从 Redis 中获取测试流程图信息，然后将其保存到数据

[1]https://www.echartsjs.com/zh/index.html

库中，Redis 交互实现与测试报告模块中类似，在此不做赘述。业务逻辑部分数据库操作类通过继承 MongoRepository 类，将数据库文档映射到 Java 对象。

4.5　本章小结

本章主要对 Web 应用自动测试系统报告生成服务务的实现细节进行阐述。在第三章系统设计过程中，将 Web 应用测试系统报告生成服务为分为执行引擎监控模块、测试报告模块以及自动化测试用例生成模块，本章依次对三个模块的具体实现进行详述。首先，对执行引擎监控模块获取三类信息的实现以及关键代码进行阐述。然后，通过顺序图以及关键代码，阐述测试报告模块中的单页面状态分析、测试报告生成以及测试报告展示三个部分的实现。最后，介绍自动化测试用例生成模块的实现，描述其顺序图、流程实现和工具类实现。

第五章　报告生成服务的测试和实验分析

第三、四章给出了 Web 应用自动化测试系统系统报告生成服务的需求分析、各个模块的详细设计和实现，在本章中根据上文提供的这些内容，对系统进行测试并进行实验评估。

5.1　测试环境

表 5.1: 测试环境

测试环境参数	参数详情
服务器机型	阿里云 ECS
数量	2 台（包括 Linux 服务器 1 台，Windows 服务器 1 台）
操作系统	Linux 服务器：Ubuntu 16.04.6 LTS Windows 服务器：Windows Server 2016
CPU	Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz
内存	16GB
宽带	10M
JVM	Linux 服务器： OpenJDK 64-Bit Server VM (build 25.232-b09, mixed mode) Windows 服务器： Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)
Redis	3.0.6 开放端口 6379
RabbitMQ	3.8.2 开放端口 5672
MongoDB	4.2.2 开放端口 27017
Selenium	3.141.59

Web 应用自动化测试系统报告生成服务的测试环境信息如表5.1所示，包括系统的服务器硬件环境以及系统集成的软件或服务的版本信息。测试过程中，使用两台阿里云 ESC 服务器用于部署系统，其中包含 1 台 Linux 服务器以及 1 台 Windows 服务器。Linux 服务器用于系统后台服务、数据库、消息中间件等的部署，Windows 服务器用于执行服务的部署，服务器参数信息包括机型、操作系统、CPU、内存、宽带。系统集成的软件或服务包括 JVM、Redis、RabbitMQ、MongoDB、Selenium。部署在 Linux、Windows 服务器上的服务依赖于 JVM，Linux 服务器中使用 OpenJDK，而 Windows 服务器中使用 HotSpot。Linux 服务器中的服务使用缓存中间件 Redis 3.0.6、消息中间件 RabbitMQ 3.8.2、数据库 MongoDB 4.2.2。执行 Web 应用测试过程中，需使用 Selenium，其版本为 3.141.59。

此外，在执行 Web 应用自动化测试的过程中，需要将截图文件以及生成的自动化测试用例存储，以供测试报告访问或下载，因此测试中还需要使用阿里云 OSS 服务。

5.2　功能测试

根据第三章中的系统设计，本文中的 Web 应用自动化测试系统被划为为执行引擎监控模块、测试报告模块、自动化测试用例生成模块以及软件质量评估报告模块。本小节以模块为单位对系统各个模块进行功能测试，并依据不同模块的使用场景设计并进行不同类型的功能测试。执行引擎监控模块和自动化测试用例模块均为独立的服务模块，向系统提供业务服务，因此需要根据模块的功能构建功能测试用例并编写 Junit 测试代码。测试报告模块、软件质量评估报告模块包含系统业务服务部分及前端展示部分，因此采用 Junit 测试代码对系统业务服务部分进行测试，使用系统界面操作对前端展示部分进行测试。

5.2.1　执行引擎模块功能测试

执行引擎监控模块功能测试的测试目的的为了保证在执行引擎在测试执行过程中，监控器能够正确并完整地获取测试执行中所产生的数据。在成功获取数据后，监控器还需要将数据存储到缓存中间件 Redis 中，并向消息中间件 RabbitMQ 发出对应的数据处理消息。

根据执行引擎监控模块的需求分析以及设计实现，需要对该模块的五个功能进行测试，分别是环境配置信息的获取、浏览器性能日志的获取、浏览器日志的获取、浏览器执行截图的获取以及执行测试路径信息的获取。对该模块的功能测试，除了检测该模块获取信息的功能以外，还需要测试与 Redis、RabbitMQ 中间件的交互。同时，考虑到执行服务中的执行引擎可能是多例的，因此在功能测试过程中对于每一项信息的获取，考虑多线程的场景。

执行引擎监控模块的功能测试用例如表5.2所示。表中针对执行引擎监控模块的五个功能，分别开展了五个功能测试用例的测试。由于多线程情况下的测试用例与表中单线程的测试用例类似，仅用多例的方式实现测试执行引擎，其他测试流程均与表中一致，所以不做多线程测试用例赘述。由于获取性能日志的监控器，除了需要存储 Redis 数据库以外，还需要向 RabbitMQ 发送分析消息，因此还需要检查 RabbitMQ 队列。执行截图存储需要使用 OSS 服务，因此需要检查 OSS 文件是否完成存储。

表 5.2: 执行引擎监控模块功能测试用例表

测试用例编号	用例描述	输入/步骤	预期输出
TC1	获取环境配置信息	1. 配置待测 Web 应用信息 2. 新建 Redis 连接、 初始化环境信息监控器 3. 初始化执行引擎	执行引擎预备执行任务； 监控器获取环境配置信息； Redis 中相应的 Key 中存储执行引擎的环境配置信息；
TC2	获取浏览器性能日志	1. 配置待测 Web 应用信息 2. 新建 Redis 连接、 新建 RabbitMQ 连接 3. 初始化性能日志监控器 4. 初始化并启动执行引擎	执行引擎执行任务； 浏览器显示任务执行； Redis 中相应的 Key 中存储浏览器中页面的性能日志；
TC3	获取浏览器日志	1. 配置待测 Web 应用信息 2. 新建 Redis 连接 3. 初始化浏览器日志监控器 4. 初始化并启动执行引擎	执行引擎执行任务； 浏览器显示任务执行； Redis 中相应的 Key 中存储浏览器中页面的浏览器日志；
TC4	获取执行截图	1. 配置待测 Web 应用信息 2. 新建 Redis 连接、 新建 OSS 连接 3. 初始化执行截图监控器 4. 初始化并启动执行引擎	执行引擎执行任务； 浏览器显示任务执行； Redis 中相应的 Key 中存储浏览器中截图路径； 通过路径能够在 OSS 访问截图
TC5	获取执行测试路径	1. 配置待测 Web 应用信息 2. 新建 Redis 连接 3. 初始化执行路径监控器 4. 初始化并启动执行引擎	执行引擎执行任务； 浏览器显示任务执行； Redis 中相应的 Key 中存储浏览器中页面的执行路径信息；

5.2.2　测试报告模块功能测试

测试报告模块提供系统业务服务以及前端展示，功能测试的目的是保证测试报告模块提供的单页面状态分析服务、测试报告生成服务的正确性，以及测试报告展示信息的完整性和合理性。因此在对测试报告模块进行功能测试的过程中分为系统业务服务部分功能测试以及前端展示部分功能测试。

测试报告模块系统业务服务部分功能测试用例如表5.3所示，包括针对测试报告生成流程、单页面分析流程正确性的测试。测试报告生成流程包括测试报告的初始化以及测试报告的生成，在测试报告模块业务流程的同时还测试了接收消息队列的功能。生成的测试报告需要完整地保存到 Redis 缓存数据库中。单页面分析流程包括调用子系统中的分析服务模块、获取环境配置信息、调用自动化测试用例生成模块以及将结果存储到 Redis 缓存数据库中，该流程的测试需要保证这些操作的正确性。

表 5.3: 测试报告模块系统业务服务部分功能测试用例

测试用例编号	用例描述	输入/步骤	预期输出
TC6	测试报告初始化	1. 初始化测试报告信息 2. 向消息队列发送初始化消息	测试报告生成流程绑定的消息队列接收到消息； 模块接收消息初始化测试报告； Redis 中相应的 Key 中存储测试报告;
TC7	单页面状态分析	1. 初始化单页面分析请求 2. 向消息队列发送分析消息	单页面分析绑定的消息队列接收到消息； 模块接收消息进行分析； Redis 中相应的 Key 中存储分析结果；
TC8	测试报告生成	1. 初始化结束消息 2. 向消息队列发送结束消息	测试报告生成流程绑定的消息队列接收到消息； 模块接收消息开始生成测试报告； Redis 中相应的 Key 中存储测试报告;

表 5.4: 测试报告模块前端展示部分功能测试用例

测试用例编号	用例描述	输入/步骤	预期输出
TC9	查看测试报告总览	1. 在平台中进入 Web 测试任务 2. 点击查看报告	显示已发布的测试任务； 跳转测试报告； 显示测试报告总览；
TC10	查看缺陷列表	1. 进入测试报告 2. 点击缺陷列表标签	显示完整的缺陷列表；
TC11-1	缺陷列表筛选	1. 进入测试报告 2. 点击缺陷列表标签 3. 点击筛选框进行筛选	显示完整的缺陷列表； 筛选后显示相应的缺陷列表；
TC11-2	缺陷列表筛选	1. 进入测试报告 2. 点击饼图	显示饼图部分对应的缺陷列表；
TC12	查看缺陷详情	1. 进入缺陷列表 2. 点击缺陷详情	显示缺陷详情页面； 显示缺陷上下文；
TC13	下载自动化测试用例	1. 进入缺陷列表 2. 点击缺陷详情 3. 点击下载地址	显示缺陷详情页面； 浏览器下载自动化测试用例文件；

测试报告模块前端展示部分功能测试用例如表5.4所示，该部分主要用于向用户展示生成的测试报告。该模块的功能测试的目的是保证用户能够提供系统查看完整的测试报告，测试报告的内容包括报告总览、缺陷列表以及缺陷详情。除了向用户展示上述三个内容以外，测试报告模块还提供缺陷列表筛选以及自动化测试用例下载。缺陷列表筛选可以通过缺陷列表页面的筛选框进行，也可以通过报告总览页面的饼图跳转，功能测试用例中分别进行了测试。自动化测试用例下载功能需要对链接的正确性以及下载进行测试。

5.2.3　自动化测试用例生成模块功能测试

自动化测试用例模块功能测试的目的是保证系统能够根据执行引擎监控模块获取的测试路径信息，生成正确的自动化测试用例。同时，生成的自动化测试用例需要能够正确运行，正常运行包括能够不出现程序异常，以及能够成功的复现页面状态或缺陷。

自动化测试用例生成模块的功能测试用例如表5.5所示，包括针对生成测试用例流程的测试以及自动化测试用例的正确性测试。生成测试用例的流程测试是针对自动化测试用例生成模块向测试报告模块提供业务服务的功能测试，验证该模块是否能提供完整的通过获取的测试执行路径来生成自动化测试用例，并存储到 OSS 文件系统中返回文件链接的服务。

自动化测试用例正确性测试是为验证生成的测试用例的用例完整性、语法正确性以及流程正确性。用例完整性是指生成的测试用例以 Java 源文件形式存在，文件名、类名符合规范，代码完整，包含引用声明、类初始化、Selenium 命令以及类关闭四部分。语法正确性要求生成的测试用例符合 Java 语法规范，并能够通过编译、执行，驱动浏览器进行流程复现，不出现语法错误或执行故障。流程正确性是指生成的测试用例能够正确地复现输入的测试执行路径，浏览器中操作过程与测试执行路径保证一致。

表 5.5: 自动化测试用例生成模块功能测试用例表

测试用例编号	用例描述	输入/步骤	预期输出
TC6、14	自动化测试用例生成	1. 初始化置信路径信息 2. 调用自动化测试用例生成服务 3. 获取用例存储地址并查看	成功调用用例生成模块服务； 获取测试用例对象； 通过 OSS 链接能够访问测试用例代码文件；
TC15	用例正确性测试	1. 下载生成的自动化测试用例 2. 使用命令行执行用例	下载完整的自动化测试用例； 成功运行测试用例； 浏览器复现执行操作；

5.2.4　软件质量评估报告模块功能测试

软件质量评估报告模块提供系统业务服务以及前端展示，功能测试的目的是保证软件质量评估报告模块提供的测试流程图获取的正确性以及报告展示的完整性和可读性。

表 5.6: 软件质量评估报告模块业务服务部分功能测试用例表

测试用例编号	用例描述	输入/步骤	预期输出
TC16	测试流程图获取及存储	1. 执行测试流程 2. 调用软件质量评估部分测试相关服务 3. 查看数据库存储内容	完成测试，Redis 存储测试流程图； MongoDB 相关文档存储网页状态节点列表以及操作路径列表；
TC17	软件质量评估报告获取	1. 完成测试任务 2. 获取质量评估报告数据	从数据库中获取软件质量评估各个维度的详细报告数据；

软件质量评估报告模块业务服务部分功能测试用例如表5.6所示，主要针对软件质量评估报告展示所需的数据获取的正确性进行测试。展示所需的数据包括各个维度的详细报告数据，其中功能以及兼容性的评估报告需要使用测试流程图辅助展示，因此功能测试还包括测试软件质量评估报告模块从 Redis 中获取执行引擎监控模块存储的测试流程图信息的可靠性。

表 5.7: 软件质量评估报告模块前端展示部分功能测试用例表

测试用例编号	用例描述	输入/步骤	预期输出
TC18	查看软件质量评估报告	1. 进入测试报告 2. 点击多维评估标签页	显示软件质量多维评估结果雷达图；
TC19	查看各维度详细报告	1. 查看软件质量评估报告 2. 点击雷达图	显示各个维度的详细评估报告；

软件质量评估报告模块前端展示部分功能测试用例如表5.7所示，该部分用于向用户展示质量评估结果雷达图以及各个维度的详细评估报告。用户通过访问测试报告的软件质量评估报告部分，查看被测应用的评估结果。同时，用户还可以通过该模块查看功能、性能、健壮性、兼容性、稳定性的详细评估报告。

5.3　实验分析

Web 应用自动化测试系统在测试环境中完成部署后，为了验证系统在真实业务场景中进行 Web 应用自动化测试的稳定性与可用性，设计并进行实验对系统进行评估。实验旨在验证系统能否在真实生产环境下稳定运行，并评估系统

进行自动化测试产生的缺陷。本节详细阐述了 Web 应用自动化测试系统的实验设计以及分析评估。

5.3.1　测试对象与实验设计

为了对系统在 Web 应用自动化测试能力进行全面地评估，基于 Web 应用不同的实现方式，从不同业务领域选取了 50 个 Web 应用网站作为测试对象。如图5.1所示，所选的 Web 应用涉及领域广泛，包括电商、计算机、教育、金融、旅游、生活、体育、新闻、娱乐以及综合共九个行业。各个行业的具有自身特殊的业务场景，如旅游行业的应用存在大量的表单输入操作，新闻行业的应用存在大量新闻列表点击操作，这些行业特征都会对自动化测试流程产生一定影响。由于计算机行业的 Web 应用，相较其他行业开发技术更为典型、丰富，因此计算机行业的应用占比最大，达到 34%。

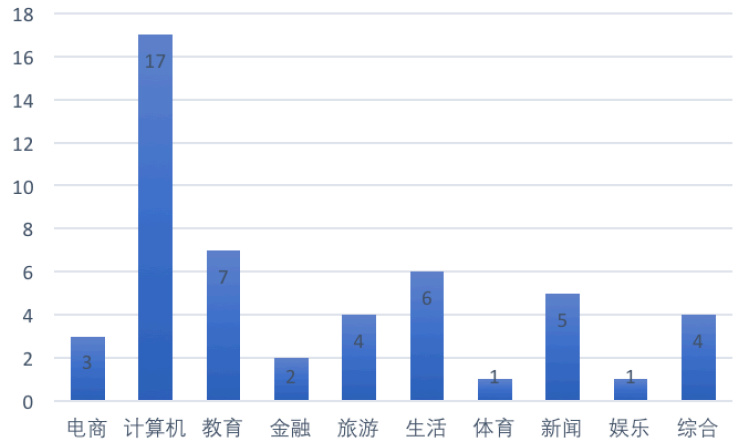

图 5.1: 测试对象行业统计

测试对象的实现方式统计如图5.2所示，包括单页应用、多页应用以及包含动态加载的多页应用，不同的实现方式可能会对自动化测试产生影响。单页应用只包含一个主页面，浏览器开始加载主页面的时候，会加载包括 HTML、JavaScript、CSS 等所有资源，所有的页面内容都会包含在主页面之中。由于资源集中加载，会增大资源加载过程中出现问题的可能性。在加载完成后，页面的跳转仅涉及刷新局部资源，降低整体页面出错风险。多页应用以及包含动态加载的多页应用由多个完整的页面组成，页面跳转会带来整页刷新。各个页面之间不共用 CSS、JavaScript，重复加载会增大缺陷出现的概率。此外，由于多页应用页面之间的数据传输依赖 URL、Cookie 或者 LocalStorage，这些数据传输过程，也可能导致缺陷。包含动态加载的多页应用，还可能会引入 AJAX 等技术带来的风险。

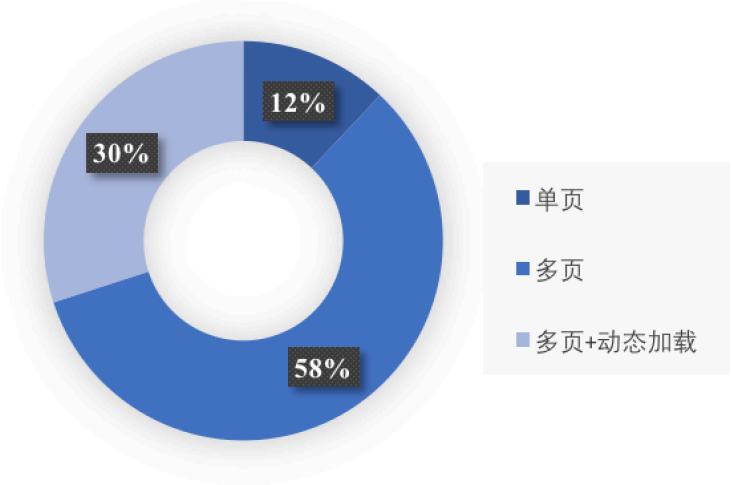

图 5.2: 测试对象实现方式统计

除此以为，网页前端可能使用不同的前端框架。目前最主流的框架包括 Vue.js、AngularJS、React 等，由于前端框架判断缺乏一定的标准以及考虑到测试用户不向系统提供前端框架信息，为保证 Web 应用自动化测试兼容主流的前端框架，测试对象中包含 Vue.js[1]、AngularJS[2]、React[3]的官方网站。

完成测试对象的选取工作后，需要对实验进行设计。根据第三章中描述 Web 应用自动化测试系统的需求，实验内容包括系统的使用、测试结果的查看以及评估，主要的流程分为发布任务、执行测试、查看测试报告、下载脚本、缺陷复现五个步骤。首先进入 Web 应用自动化测试系统后，通过输入待测 Web 应用 URL 发布自动化测试任务。任务发布后，系统启动测试流程，通过 RDP 登陆 Windows Server 监控执行服务状态。接着，在测试完成后，通过系统前端查看测试报告是否生成，测试报告生成后通过提供的链接进入。在报告中查看测试结果总览、统计以及详细的缺陷列表。在缺陷列表中选择单个缺陷查看缺陷具体信息。在单个缺陷详情下载缺陷的自动化测试用例脚本，执行该脚本尝试复现缺陷。

5.3.2　评估与分析

依据上文设计的实验设计，获取待测 Web 应用的 URL 作为输入进行自动化测试。由于 Web 应用网页元素和操作的组合数量大，进而产生数量极大的页面

[1]https://cn.vuejs.org/v2/guide/

[2]https://ng.ant.design/docs/introduce/zh

[3]https://react.docschina.org/

状态，在实验过程中，限定测试执行时间为 1 小时，使用相同的时间来验证系统对各类应用的测试效率。测试对象一共包含 50 个 Web 应用，在实验中全部通过完成相应的自动化测试任务，并生成测试报告。

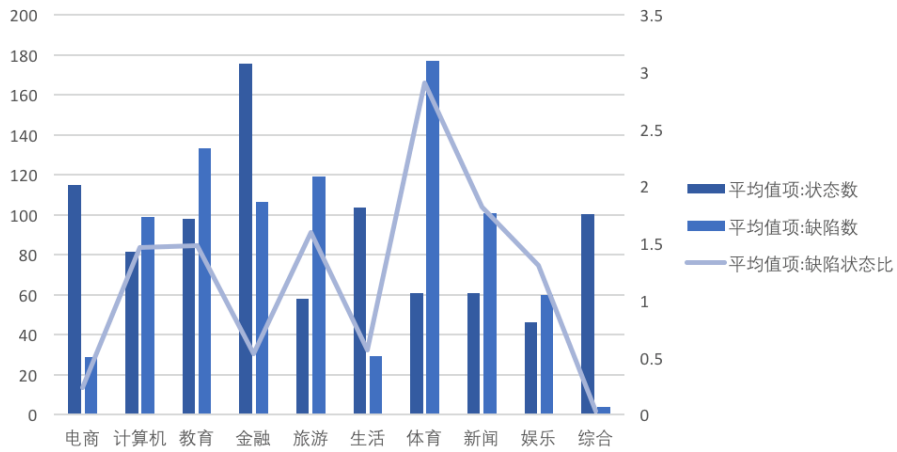

图 5.3: 实验结果行业统计

实验结果行业统计如图5.3所示，在测试对象所在的九个行业中，所有行业的应用均通过自动化测试并产生测试报告。测试执行状态数的中位数为 81.3，平均数为 88.9，检测出的缺陷数的中位数为 98.8，平均数为 85.9，大部分行业缺陷状态比在 1 左右，系统在各个行业的 Web 应用测试过程中，都能够在限定的时间内执行并分析大量的页面状态，并能够检测出一定数量的缺陷，系统针对各个行业的 Web 应用均具有较高测试效率和测试性能。

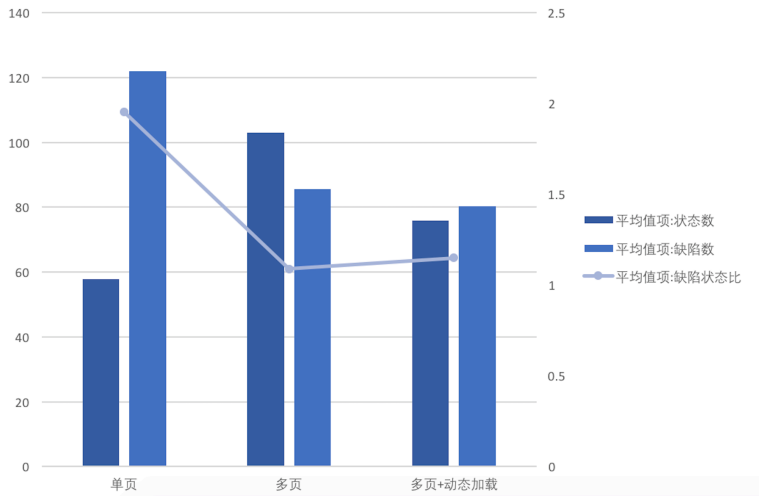

图 5.4: 实验结果实现方式统计

实验结果实现方式统计如图5.4所示，使用三种不同的方式实现的 Web 应用，均通过自动化测试并产生测试报告。系统在对这些以不同方式实现的 Web 应用进行测试过程中，都执行并分析了大量的页面状态并检测出一定数量的缺陷，系统对于 Web 应用的实现方式具有良好的测试兼容性。此外，由于被测的 Web 应用包含 Vue.js、AngularJS、React 等前端框架，实验结果表明系统对主流前端框架也具有良好的测试兼容性。

在完成测试报告生成后，通过缺陷详情页面中下载自动化测试用例，下载后执行自动化测试用例对缺陷进行复现，对缺陷复现率、缺陷分类准确率、测试用例执行成功率进行评估。

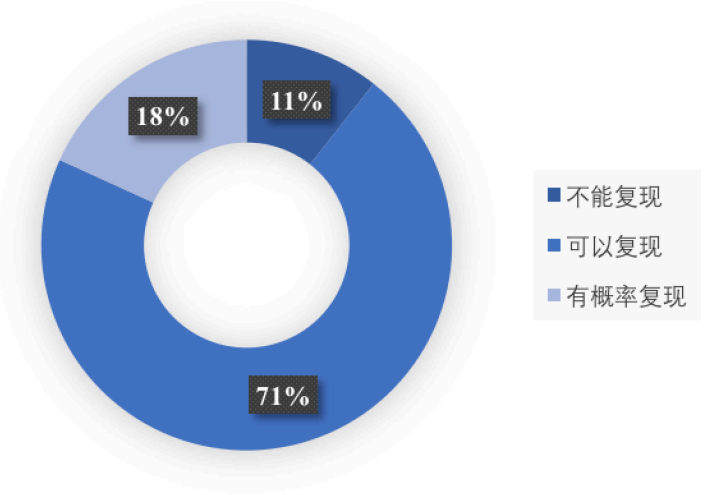

图 5.5: 缺陷复现情况

实验结果的缺陷复现情况如图5.5所示，在使用自动化测试用例复现缺陷过程中，能够复现的缺陷占所有缺陷的 89%，其中必然复现的概率占 71%，有概率复现的概率占 18%。经分析发现，有些缺陷涵盖资源加载问题和性能问题，此类缺陷的出现存在一定的偶然性，在复现缺陷的过程中不能够必然复现。这类缺陷主要是由于网络状态和应用服务器压力导致的，也属于 Web 应用缺陷的范畴，又区别于必然能够复现的缺陷。

在使用自动化测试用例复现缺陷的过程中，系统提供的测试用例执行成功率达到 100%。由于测试测试人员采用人工编写测试脚本方法复现缺陷，不仅需要测试人员熟悉 Web 应用测试套件，而且需要通过浏览器手动对元素进行定位，因此系统提供的自动化测试用例能够有效地帮助测试人员简化工作流程，大大降低测试复现成本，提高复现缺陷的效率。

采用人工审阅的方式缺陷进行分析，判断系统检测出得缺陷是否分类准确，分析缺陷分类准确率。在人工审阅过程中，通过测试报告中的缺陷日志、缺陷截图以及复现缺陷的方式，对缺陷进行评估。实验后发现，系统对缺陷的分类准确率达到了 97%，缺陷准确率表明了系统对缺陷检测、分类的有效性。

此外，由于实验复现缺陷采用自动化测试用例以及人工审阅的方式相结合的半自动化方式，对于每一项系统的自动化测试任务的分析时间都远远超过 1 小时。上文实验过程中，对所有测试对象，均限定自动化测试执行时间为 1 小时。因此，系统的测试效率远高于半自动化的测试方法。如果，采用纯人工的进行 Web 应用测试，需要测试人员使用浏览器对网页元素进行定位并使用 Selenium 编写测试脚本，然后进行网页缺陷分析，所花费的测试时间较前两者更长，这也验证 Web 应用自动化测试系统的可用性和高效性。

综上的实验结果表明，Web 应用自动化测试系统对于 Web 应用的行业特征以及实现方式都提供了良好的支持，系统能够针对被测的 Web 应用进行高效、全面的测试，能够有效地检测、分析、整理缺陷并提供完整的 Web 应用测试报告，简化了 Web 应用测试流程，提高了测试效率，降低了测试成本。

5.3.3　典型案例

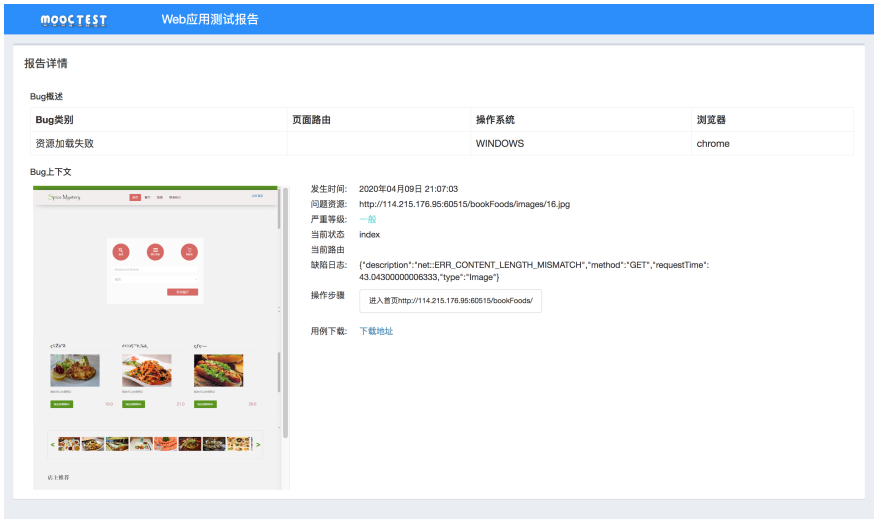

图 5.6: 资源加载失败缺陷示例

Web 应用自动化测试系统测试报告，可以向用户展示待测应用中检测出页面或操作断链、请求无效、未认证、资源加载失败、资源加载时间过长、跨域警告、JavaScript 错误等缺陷，这些缺陷主要通过提取执行引擎监控模块获取的运行时数据中的两类信息进行分析，包括浏览器请求日志和控制台日志，本节中

通过典型案例分析展示用户通过测试报告分析这两类缺陷信息。

如图5.6所示的是资源加载失败缺陷示例，该缺陷检测自生活行业的一个 Web 应用，该应用属于多页应用。该缺陷是表征为页面中间的背景图片 16.jpg 未成功加载，导致网页的显示缺失，用户可以通过网页的截图发现这种表征。用户通过查看缺陷上文展示的信息，可以发现缺陷出现的问题资源为 16.jpg，并且通过缺陷日志可以发现资源请求失败的情况。此时用户可以通过报告提供的操作步骤手动操作复现缺陷，或者使用报告提供的测试用例自动化复现缺陷，通过缺陷出现的流程和复现，动态、更加精准地对该缺陷进行分析。

图 5.7: JavaScript 错误缺陷示例

如图5.7所示的 JavaScript 错误示例，该缺陷检测自计算机行业的一个单页应用。用户通过缺陷详情，明确该缺陷属于 JavaScript 出现的错误，输入图中所示的页面路由。通过缺陷详情页面提供的自动化测试用例，尝试复现该缺陷，可以发现控制台中打印了如图中缺陷日志项中所示的 JavaScript 错误信息。

5.4　本章小结

本章首先对主要对本文中的报告生成服务进行系统测试，在阐述了系统测试的测试环境后，基于第三、四章中的系统设计与实现，按照模块划分对每个模块分别设计了功能测试用例并进行了功能测试，保证报告生成服务的可用性和可靠性。在完成系统测试后，对 Web 应用自动化测试系统进行实验与分析，在真实场景下选取 50 个测试对象进行系统实验，并列出评估指标对实验结果进行分析，最后结合典型案例对系统进行分析。

第六章　总结与展望

6.1　总结

随着 Web 应用开发的快速发展，研究 Web 应用自动化测试方法十分必要。现有的 Web 应用测试技术已经取得一定的进展，提出了一些 Web 应用自动化测试方法，但现有的测试报告无法满足测试人员的需求。为了提升 Web 应用测试效率，降低测试成本，本文实现了 Web 应用自动化测试系统报告生成服务。

本文首先阐述 Web 应用自动化测试的项目背景以及国外研究现状，针对 Web 应用测试的挑战性、研究自动化测试方法以及清晰易用测试报告的必要性，提出了本文针对自动化测试业务流程的报告生成服务系统，然后介绍了本文系统在实现过程中使用的主要技术，包括用于构建服务端的 Spring Boot 框架、构建测试报告前端的 JQuery 框架，消息中间件 RabbitMQ、缓存中间件 Redis，用于采集执行过程信息的 Crawlijax Plugin 以及用于生成自动化测试用例的 Selenium 测试框架。

本文在第三章对系统的整体进行的概述，进而分析系统的功能性和非功能性需求，通过系统用例图描述了系统用例，使用系统架构图以及 4+1 视图描述了系统架构，并依据功能将系统划分为执行引擎监控、测试报告、自动化测试用例生成以及软件质量评估报告模块。在划分模块后，使用模块架构图、类图对模块进行详细设计。在第四章中依照第三章对三个模块的设计，使用顺序图、关键代码对模块的实现进行阐述。在第五章中，设计并执行系统功能测试，保证各个模块的可靠性，用时基于真实生产环境进行实验，分析整个系统的稳定性和可靠性。

目前系统已经上线运行并能够提供 Web 应用自动化测试报告生成服务，用户只需要在系统中配置待测 Web 应用的 URL 即可进行自动化测试获取详细的测试报告。此外，针对更加专业使用需求，系统也提供配置测试过程的服务，用户可以通过系统配置测试过程中的认证信息、输入框值等信息。报告生成服务通过监控执行服务获取自动化测试日志，从而进行单页面分析以及自动化测试用例生成，构建出包含完整的缺陷上下文的缺陷详细模型，包括缺陷分类、缺陷所在资源、缺陷出现的页面状态、缺陷日志、缺陷出现前的操作流程、缺陷复现方法等信息。在生成测试报告后，通过前端页面进行数据可视化，生成可读性高、信息明确的测试报告，并提供便捷的复现方法，极大提高用户发现、甄别、修复缺陷的效率。同时，提供软件质量评估报告，用于展示功能、性能、健壮性、

兼容性、稳定性五个维度的软件质量评估结果。本系统的实现，简化了 Web 应用测试的流程，降低了 Web 应用的测试成本。

6.2　展望

第一，系统向用户展示了被测 Web 应用存在的缺陷，但未提供针对缺陷的修复建议。系统提供修复建议后，用户在查看缺陷的同时，可以依据系统的指导对出现的缺陷进行有效地修复。当指导修复功能上线后，还可以依靠平台的用户积累和修复方案反馈，不断更新和提升各类缺陷的修复建议的质量，提高 Web 应用缺陷报告对用户的指导价值。

第二，系统提供了自动化测试用例的下载，方便用户自主复现缺陷，但现阶段仍然需要用户自主执行。为了更方便用户，系统后续可以提供复现执行环境，用户不仅可以通过缺陷详情页面下载测试用例，还可以直接在线执行测试用例，查看复现情况。

第三，在系统实现方式上还可以优化。系统未使用备用数据库，容灾能力差，可以考虑使用主备模式。此外，测试报告模块与子系统中的分析服务处于用一个应用之中，耦合度较大，考虑到系统鲁棒性，可以采用微服务架构进行重构。

参考文献

[1] S. Dogan, A. Betin-Can, V. Garousi, Web application testing: A systematic literature review, J. Syst. Softw. 91 (2014) 174–201.
URL https://doi.org/10.1016/j.jss.2014.01.010

[2] N. Gupta, V. Yadav, M. Singh, Automated regression test case generation for web application: A survey, ACM Comput. Surv. 51 (4) (2018) 87:1–87:25.
URL https://doi.org/10.1145/3232520

[3] V. Garousi, A. Mesbah, A. Betin-Can, S. Mirshokraie, A systematic mapping study of web application testing, Inf. Softw. Technol. 55 (8) (2013) 1374–1396.
URL https://doi.org/10.1016/j.infsof.2013.02.006

[4] Z. Qian, H. Miao, H. Zeng, A practical web testing model for web application testing, in: K. Yétongnon, R. Chbeir, A. Dipanda (Eds.), Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, SITIS 2007, Shanghai, China, December 16-18, 2007, IEEE Computer Society, 2007, pp. 434–441.
URL https://doi.org/10.1109/SITIS.2007.16

[5] M. Leotta, D. Clerissi, F. Ricca, C. Spadaro, Comparing the maintainability of selenium webdriver test suites employing different locators: A case study, in: International Workshop on Joining Academia Industry Contributions to Testing Automation, 2013, pp. 53–58.

[6] D. Cruzes, T. Dybå, Synthesizing evidence in software engineering research, in: G. Succi, M. Morisio, N. Nagappan (Eds.), Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, 16-17 September 2010, Bolzano/Bozen, Italy, ACM, 2010.
URL https://doi.org/10.1145/1852786.1852788

[7] N. Alshahwan, M. Harman, Automated web application testing using search based software engineering, in: P. Alexander, C. S. Pasareanu, J. G. Hosking (Eds.), 26th IEEE/ACM International Conference on Automated Software Engineering

(ASE 2011), Lawrence, KS, USA, November 6-10, 2011, IEEE Computer Society, 2011, pp. 3–12.
URL https://doi.org/10.1109/ASE.2011.6100082

[8] M. E. Özkinaci, A. B. Can, Detecting execution and HTML errors in ASP.NET web applications, in: M. J. E. Cuaresma, B. Shishkov, J. Cordeiro (Eds.), ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, Volume 2, Seville, Spain, 18-21 July, 2011, SciTePress, 2011, pp. 172–178.

[9] J. Bau, E. Bursztein, D. Gupta, J. C. Mitchell, State of the art: Automated black-box web application vulnerability testing, in: 31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA, IEEE Computer Society, 2010, pp. 332–345.
URL https://doi.org/10.1109/SP.2010.27

[10] A. A. Andrews, J. Offutt, R. T. Alexander, Testing web applications by modeling with fsms, Software and Systems Modeling 4 (3) (2005) 326–345.
URL https://doi.org/10.1007/s10270-004-0077-7

[11] A. A. Andrews, A. J. Offutt, C. E. Dyreson, C. J. Mallery, K. Jerath, R. T. Alexander, Scalability issues with using fsmweb to test web applications, Inf. Softw. Technol. 52 (1) (2010) 52–66.
URL https://doi.org/10.1016/j.infsof.2009.06.002

[12] 缪淮扣, 陈圣波, 曾红卫, 基于模型的 web 应用测试, 计算机学报 34 (6) (2011) 1012–1028.

[13] P. Ammann, J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008.
URL https://doi.org/10.1017/CBO9780511809163

[14] U. Praphamontripong, J. Offutt, Applying mutation testing to web applications, in: Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010, Workshops Proceedings, IEEE Computer Society, 2010, pp. 132–141.
URL https://doi.org/10.1109/ICSTW.2010.38

[15] S. Mirshokraie, A. Mesbah, K. Pattabiraman, Efficient javascript mutation testing, in: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013, IEEE Computer Society, 2013, pp. 74–83.
URL https://doi.org/10.1109/ICST.2013.23

[16] A. Marchetto, F. Ricca, P. Tonella, A case study-based comparison of web testing techniques applied to AJAX web applications, Int. J. Softw. Tools Technol. Transf. 10 (6) (2008) 477–492.
URL https://doi.org/10.1007/s10009-008-0086-x

[17] A. Marchetto, P. Tonella, F. Ricca, State-based testing of ajax web applications, in: First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008, IEEE Computer Society, 2008, pp. 121–130.
URL https://doi.org/10.1109/ICST.2008.22

[18] A. Mesbah, E. Bozdag, A. van Deursen, Crawling AJAX by inferring user interface state changes, in: D. Schwabe, F. Curbera, P. Dantzig (Eds.), Proceedings of the Eighth International Conference on Web Engineering, ICWE 2008, 14-18 July 2008, Yorktown Heights, New York, USA, IEEE Computer Society, 2008, pp. 122–134.
URL https://doi.org/10.1109/ICWE.2008.24

[19] 贺涛, 缪淮扣, 钱忠胜, 基于 ajax 技术的 web 应用的建模与测试用例生成, 计算机科学 41 (8) (2013) 219–223,244.

[20] S. Sprenkle, S. Sampath, E. Gibson, L. L. Pollock, A. L. Souter, An empirical comparison of test suite reduction techniques for user-session-based testing of web applications, in: 21st IEEE International Conference on Software Maintenance (ICSM 2005), 25-30 September 2005, Budapest, Hungary, IEEE Computer Society, 2005, pp. 587–596.
URL https://doi.org/10.1109/ICSM.2005.18

[21] S. Thummalapenta, K. V. Lakshmi, S. Sinha, N. Sinha, S. Chandra, Guided test generation for web applications, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco,

CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 162–171.
URL https://doi.org/10.1109/ICSE.2013.6606562

[22] V. Dallmeier, B. Pohl, M. Burger, M. Mirold, A. Zeller, Webmate: Web application test generation in the real world, in: Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014 Workshops Proceedings, March 31 - April 4, 2014, Cleveland, Ohio, USA, IEEE Computer Society, 2014, pp. 413–418.
URL https://doi.org/10.1109/ICSTW.2014.65

[23] S. Wang, W. Wu, J. Sun, A method for test cases reduction in web application testing based on user session, in: International Conference on Networking and Network Applications, NaNA 2018, Xi'an, China, October 12-15, 2018, IEEE, 2018, pp. 378–383.
URL https://doi.org/10.1109/NANA.2018.8648767

[24] 武晋南, 高建华, 基于用户行为和会话的 web 应用测试方法 application test method based on user behavior and session, 计算机工程 036 (8) (2010) 83–85.

[25] E. Kiciman, V. B. Livshits, Ajaxscope: a platform for remotely monitoring the client-side behavior of web 2.0 applications, in: T. C. Bressoud, M. F. Kaashoek (Eds.), Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007, ACM, 2007, pp. 17–30.
URL https://doi.org/10.1145/1294261.1294264

[26] A. Mesbah, M. R. Prasad, Automated cross-browser compatibility testing, in: R. N. Taylor, H. C. Gall, N. Medvidovic (Eds.), Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, ACM, 2011, pp. 561–570.
URL https://doi.org/10.1145/1985793.1985870

[27] S. R. Choudhary, H. Versee, A. Orso, WEBDIFF: automated identification of cross-browser issues in web applications, in: 26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania, IEEE Computer Society, 2010, pp. 1–10.
URL https://doi.org/10.1109/ICSM.2010.5609723

[28] S. R. Choudhary, M. R. Prasad, A. Orso, Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications, in: G. Antoniol, A. Bertolino, Y. Labiche (Eds.), Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012, IEEE Computer Society, 2012, pp. 171–180.
URL https://doi.org/10.1109/ICST.2012.97

[29] S. Raina, A. P. Agarwal, An automated tool for regression testing in web applications, ACM SIGSOFT Software Engineering Notes 38 (4) (2013) 1–4.
URL https://doi.org/10.1145/2492248.2492272

[30] S. Mirshokraie, A. Mesbah, JSART: javascript assertion-based regression testing, in: M. Brambilla, T. Tokuda, R. Tolksdorf (Eds.), Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings, Vol. 7387 of Lecture Notes in Computer Science, Springer, 2012, pp. 238–252.
URL https://doi.org/10.1007/978-3-642-31753-8_18

[31] A. Chaturvedi, A. Gupta, A tool supported approach to perform efficient regression testing of web services, in: A. D. Ionita, G. A. Lewis, M. Litoiu (Eds.), 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2013, Eindhoven, The Netherlands, September 23, 2013, IEEE Computer Society, 2013, pp. 50–55.
URL https://doi.org/10.1109/MESOCA.2013.6632734

[32] C. Caci, Testing object-oriented systems, ACM SIGSOFT Software Engineering Notes 25 (3) (2000) 60–61.
URL https://doi.org/10.1145/505863.505884

[33] L. Ran, C. E. Dyreson, A. A. Andrews, R. C. Bryce, C. J. Mallery, Building test cases and oracles to automate the testing of web database applications, Inf. Softw. Technol. 51 (2) (2009) 460–477.
URL https://doi.org/10.1016/j.infsof.2008.05.016

[34] K. Pattabiraman, B. G. Zorn, Dodom: Leveraging DOM invariants for web 2.0 application robustness testing, in: IEEE 21st International Symposium on Software Reliability Engineering, ISSRE 2010, San Jose, CA, USA, 1-4 November 2010, IEEE Computer Society, 2010, pp. 191–200.
URL https://doi.org/10.1109/ISSRE.2010.17

[35] S. Artzi, J. Dolby, F. Tip, M. Pistoia, Fault localization for dynamic web applications, IEEE Trans. Software Eng. 38 (2) (2012) 314–335.
URL https://doi.org/10.1109/TSE.2011.76

[36] Y. Zou, Z. Chen, Y. Zheng, X. Zhang, Z. Gao, Virtual DOM coverage for effective testing of dynamic web applications, in: C. S. Pasareanu, D. Marinov (Eds.), International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014, ACM, 2014, pp. 60–70.
URL https://doi.org/10.1145/2610384.2610399

[37] 彭树深, 顾庆, 陈道蓄, Web 应用测试用例生成研究, 计算机科学 37 (6) (2010) 159–163.

[38] V. Debroy, L. Brimble, M. Yost, A. Erry, Automating web application testing from the ground up: Experiences and lessons learned in an industrial setting, in: 11th IEEE International Conference on Software Testing, Verification and Validation, ICST 2018, Västerås, Sweden, April 9-13, 2018, IEEE Computer Society, 2018, pp. 354–362.
URL https://doi.org/10.1109/ICST.2018.00042

[39] F. Gutierrez, Spring Boot Messaging, Messaging APIs for Enterprise and Integration Solutions, Springer, 2017.
URL https://doi.org/10.1007/978-1-4842-1224-0

[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 124 736–743.

[41] A. van Deursen, A. Mesbah, A. Nederlof, Crawl-based analysis of web applications: Prospects and challenges, Sci. Comput. Program. 97 (2015) 173–180.
URL https://doi.org/10.1016/j.scico.2014.09.005

[42] P. Rubio-Conde, D. Villarán-Molina, M. García-Valls, Measuring performance of middleware technologies for medical systems: Ice vs AMQP, SIGBED Review 14 (2) (2017) 8–14.
URL https://doi.org/10.1145/3076125.3076126

[43] C. B. Hauser, J. Domaschka, Vice registry: An image registry for virtual collaborative environments, in: IEEE International Conference on Cloud Computing

Technology and Science, CloudCom 2017, Hong Kong, December 11-14, 2017, IEEE Computer Society, 2017, pp. 82–89.
URL https://doi.org/10.1109/CloudCom.2017.11

[44] X. J. Hong, H. S. Yang, Y. H. Kim, Performance analysis of restful API and rabbitmq for microservice web application, in: International Conference on Information and Communication Technology Convergence, ICTC 2018, Jeju Island, Korea (South), October 17-19, 2018, IEEE, 2018, pp. 257–259.
URL https://doi.org/10.1109/ICTC.2018.8539409

[45] T. Walkowiak, Asynchronous system for clustering and classifications of texts in polish, in: W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, J. Kacprzyk (Eds.), Dependability Engineering and Complex Systems - Proceedings of the Eleventh International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 27-July 1, 2016, Brunów, Poland, Vol. 470 of Advances in Intelligent Systems and Computing, Springer, 2016, pp. 529–538.
URL https://doi.org/10.1007/978-3-319-39639-2_46

[46] S. Heule, M. Nunkesser, A. Hall, Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm, in: G. Guerrini, N. W. Paton (Eds.), Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013, ACM, 2013, pp. 683–692.
URL https://doi.org/10.1145/2452376.2452456

[47] 刘高军, 王帝澳, 基于 redis 的海量小文件分布式存储方法研究, 计算机工程与科学 35 (10) (2013) 58–64.

[48] A. Bruns, A. Kornstädt, D. Wichmann, Web application tests with selenium, IEEE Software 26 (5) (2009) 88–91.
URL https://doi.org/10.1109/MS.2009.144

[49] R. Chen, H. Miao, A selenium based approach to automatic test script generation for refactoring javascript code, in: 2013 IEEE/ACIS 12th International Conference on Computer and Information Science, ICIS 2013, Niigata, Japan, June 16-20, 2013, IEEE Computer Society, 2013, pp. 341–346.
URL https://doi.org/10.1109/ICIS.2013.6607864

[50] 周玲余, 基于 jquery 框架的页面前端特效的设计与实现%design and implementation of webpage effects based on jquery framework, 计算机与现代化 000 (1) (2013) 61–63.

简历

基本情况 张晨剑，男，汉族，1996 年 8 月出生，江苏省苏州市人。

教育背景

2014.9～2018.6 南京大学软件学院 硕士

2018.9～2020.6 南京大学软件学院 本科

致　　谢

光阴似箭，两年的研究生生涯到了收尾的时候。回首过去两年，老师的谆谆教导，同学的朝夕相处，都让我记忆犹新。在临近毕业之际，我满怀感激之情，真诚地向所有关心过我、帮助过我的人表达衷心的感谢与祝福。

首先要向我的研究生导师陈振宇老师献上衷心的感谢，无论是毕业设计的选题、方案构思还是系统开发、实验设计、论文撰写，都离不陈老师的悉心指导和帮助。陈老师严谨的治学态度和开阔的学术思想都深深的感染了我，值得我去学习和体会。

其次还要感谢实验室的其他老师，每次例会的讨论，他们都会对我实现方案和实验设计提出指导，帮助我不断完善，使我受益良多。也十分感谢实验室与我协作开发的尹子越同学和周赛同学，他们以严谨的态度和密切的配合在项目开发期间给了我很大的帮助。

特别感谢我的父母和家人一直以来给予我的支持和关新，虽然他们并没有给我学术上的指导，但是他们的支持让我可以全身心的投入科研工作之中，并给予了我前进的动力。

另外，感谢研究生期间所有帮助过我、关心过我的人!

致谢基金:

国家自然科学基金项目：基于可理解信息融合的人机协同移动应用测试研究（61802171）, 2019-2021

中央高校基本科研业务费专项资金资助项目：基于群智协同的众包测试技术（14380021），2020-2020

南京大学技术创新基金项目：基于 AI 中台的移动应用测试技术研究与实现（14913413），2020-2020

版权与原创性说明

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

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

作者签名：

日期: 2020 年 5 月 23 日