南京大學

研 究 生 毕 业 论 文
（申请工程硕士学位）

论 文 题 目 **Java Web** 应用可移植性自动化测试系统的设计与实现

作 者 姓 名 薛晓波

学科、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

2021 年 **5** 月 **18** 日

学　　号：**MF1932216**

论文答辩日期：**2021** 年 **05** 月 **20** 日

指 导 教 师：  　（签字）

Design and Implementation of Automated Testing System for Portability of Java Web Application

by

Xiaobo Xue

Supervised by

Professor Zhenyu Chen

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering

Software Institute
Nanjing University

May 18, 2021

学 位 论 文 原 创 性 声 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，本论文不句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。

论文作者签名：薛晓波

日期：2021 年 5 月 20 日

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：Java Web 应用可移植性自动化测试系统的设计与实现

工程硕士（软件工程领域）专业 2019 级硕士生姓名：薛晓波

指导教师（姓名、职称）：陈振宇 教授

摘　要

随着互联网软件技术快速发展，Java Web 应用成为日常生活与工业生产中的重要组成部分。由于市场需求多样化和软硬件技术发展等因素影响，Java Web 应用的部署环境与部署方式也不停变化，对可移植性能力也提出更高的要求。因此如何测试 Java Web 应用的可移植性成为软件厂商和开发者日益关注的问题。目前主流可移植性测试方法依赖人工搭建多个测试环境，部署应用后进行验证测试。这种方法虽然有效，但是存在工作量大、测试成本高、效率低的问题。

基于上述背景，本文设计实现面向 Java Web 应用可移植性的自动化测试系统，包含测试环境生成、节点控制、调度执行、自动化测试和自动化分析五个核心模块。测试人员能够上传待测应用并创建可移植性测试任务，系统根据不同测试环境拆解为多个子任务。在可移植性测试任务触发执行时，由于资源有限，调度执行模块会负责子任务的执行管理，基于延迟队列和消息队列控制任务执行步骤和过程，并通过排队机制提高性能和可用性。每个子任务首先需要通过节点控制模块申请资源，该模块负责维护移植部署集群节点的资源状态，提供资源分配、节点注册注销等功能。测试环境生成模块基于 Docker 容器技术，会根据用户选择的测试环境配置组合，自动化生成测试环境，并运行上传的待测应用。在应用成功运行后，自动化测试模块调用 HttpRunner 进行功能与性能测试，为保证自动化测试工具的高可用，基于 Celery 框架实现工具调用的异步调度执行。在任务执行测试期间，本系统还会收集若干监控数据和日志信息，最后通过自动化分析模块，对测试结果进行分析，计算指标，生成可视化测试报告。为提高系统扩展能力，本系统模块均基于微服务架构开发，移植部署集群和 Celery Worker 集群均支持弹性伸缩，并可对自动化测试工具进行插拔式扩展。

为保证系统服务质量，本文设计编写 60 个单元测试用例和 30 个接口测试用例验证核心业务功能的正确性，并全部执行通过。目前系统已落地上线运行，并对 20 个真实 Java Web 应用进行可移植性测试，除 1 个应用由于自身原因无法执行测试外，其余均成功运行，测试成功率为 95%。结果表明本系统能够提供为客户提供符合预期的 Java Web 应用可移植性自动化测试服务。

关键词： Java Web 应用，软件可移植性，自动化测试，微服务

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Design and Implementation of Automated Testing System for Portability of Java Web Application

SPECIALIZATION: Software Engineering

POSTGRADUATE: Xiaobo Xue

MENTOR: Professor Zhenyu Chen

Abstract

With the rapid development of Internet software technology, java web application has become an important part of daily life and industrial production. Due to the diversification of market demand and the development of software and hardware technologies, the deployment environment and deployment methods of java web applications are constantly changing, and higher requirements are placed on portability. Therefore, how to test the portability of java web applications has become an issue of increasing concern for software vendors and developers. The current mainstream portability testing methods rely on manually setting up multiple test environments, and performing verification tests after deploying applications. Although this method is effective, it has the problems of large workload, high test cost, and low efficiency.

Based on the above background, this paper designs and implements an automated testing system for the portability of Java Web application, which includes five core modules: test environment generation, node control, scheduling execution, automated test and automated analysis. Testers can upload applications to be tested and create portability test tasks through this system. The system is disassembled into multiple subtasks according to different test environments. When the portability test task triggers execution, due to limited resources, the scheduling execution module will be responsible for the execution and management of the subtasks, and control the task execution steps and processes based on the delay queue and message queue, and improve performance and availability through the queuing mechanism. Each subtask first needs to apply for resources through the node control module, which is responsible for maintaining the resource status of the transplanted and deployed cluster nodes, and provides functions

such as resource allocation, node registration and cancellation. The test environment generation module is based on Docker container technology, and will automatically generate the test environment according to the test environment configuration combination selected by the user, and run the uploaded application to be tested. After the application runs successfully, the automated test module calls HttpRunner for functional and performance testing. To ensure the high availability of automated testing tools, asynchronous scheduling execution of tool calls is implemented based on the Celery framework. During the task execution test, the system will also collect a number of monitoring data and log information. Finally, through the automated analysis module, the test results are analyzed, the indicators are calculated, and the visual test report is generated. In order to improve the system scalability, the system modules are developed based on the microservice architecture. Both the transplantation deployment cluster and the Celery Worker cluster support elastic scaling, and the automated testing tools can be pluggable extensions.

In order to ensure the quality of system service, this paper designs and writes 60 unit test cases and 30 interface test cases to verify the correctness of the core business functions, and all of them are executed and passed. At present, the system has been put into operation, and portability tests have been performed on 20 real Java Web applications. Except for one application that could not be tested due to its own reasons, the rest were successfully run, with a test success rate of 95%. The results show that the system can provide customers with the expected automated testing services for the portability of Java Web applications.

keywords: Java Web Application, Software Portability, Automated Testing, Microservices

目　　录

插图清单

附表清单

第一章　引言

1.1　背景与研究意义

从 20 世纪中期软件开始兴起，软件技术不断发展，逐渐向各行各业中渗透交融，软件在人民群众的日常生活中扮演的角色也愈发重要 [1]。在这个软件百花齐放的时代，Web 应用因其独具优势的 B/S 架构，被诸多开发者和用户接受。作为使用者来说，Web 应用无需下载安装，直接通过浏览器进行访问，脱离了用户操作系统的局限 [2]，既缩减了开发和维护成本，又大幅降低用户使用门槛，因此被大量互联网公司采用，在市场中占据重要地位 [3]。根据研究表示，在过去的二十年中，由于互联网对我们现代经济的影响，对 Web 应用程序需求有了显著增加，同时，Web 应用的开发、测试、部署等过程变得越来越复杂且难以管理，其进展速度还不足以满足这些需求和期望 [4]。Java 作为一款适合用于开发 Web 应用的编程语言 [5]，具有较高的市场占有率，根据 TIOBE 调研统计，Java 成为 2020 年热度排名第二的编程语言 [6]，因此保障 Java Web 应用的软件质量，具有十分重要的现实意义。

近年来计算机硬件发展迅猛，操作系统、基础软件和云计算等技术也经历着一系列剧变，Java Web 应用部署运行环境的可选项也不断增多，如操作系统版本、CPU 和内存大小等都可能成为影响 Java Web 应用运行的因素。随着竞争加剧和客户需求多样化，企业和用户对 Java Web 应用在多种平台和环境中运行的能力提出了更高的要求 [7]。另一方面，我国为摆脱信息科技产业核心技术长期依赖国外输入的现状，大力发展国产软硬件和操作系统，并于 2019 年提出信息技术应用创新战略，旨在实现关键信息技术的自主创新和自主可控。根据《中国信创产业发展白皮书 (2021)》[①]，我国将基础软硬件、行业应用软件、信息安全等领域纳入发展规划，包含了 CPU、操作系统、数据库、中间件等产品。由于国产软硬件过去所占市场份额有限，Java Web 应用主要运行环境依然依赖于 Ubuntu、CentOS 等由国外组织主导的产品，为响应国家号召，将产品与国产软硬件进行适配势在必行，这对 Java Web 应用可移植性能力提出了新的

[①] http://www.cinic.org.cn/hy/zx/1040638.html

挑战。

基于上述背景，对 Java Web 应用的可移植性进行评估无法脱离软件测试的方法。软件测试作为软件生命周期中关键部分，用于保障软件质量 [8]，其地位在软件开发过程中日趋重要。可移植性作为软件质量之一 [9]，理想情况下，软件可以在不同环境之间移植而无需修改源代码，但这在现实场景中几乎不可能实现 [10]。因此为了评估 Java Web 应用的移植能力边界，保证其能够顺利在不同环境中部署运行，需要使用软件测试的理论和技术对 Java Web 应用进行可移植性测试。现在主流可移植性测试手段仍然高度依赖于人工 [11]，由测试人员根据需求设计、搭建多个不同配置的测试环境，再将应用程序逐一部署到每个测试环境，成功运行后测试人员结合测试工具进行一致性验证测试并填写测试报告 [12]。这种传统方式存在几个主要问题，一是耗费大量时间和资源，而 Web 应用开发通常建议使用敏捷开发模型进行迭代 [13]，传统测试手段显然难以适应高效快速的迭代节奏；二是在测试过程中人工能够收集到的数据有限，难以分析评估应用在不同环境中存在的问题；三是人工填写的测试报告可读性差。因此，如何解决人工进行 Java Web 应用可移植性测试过程中存在的问题成为本文探究的主要方向。

针对上述问题，本文设计实现面向 Java Web 应用可移植性的自动化测试系统，根据测试需求，只需要测试人员提供少量的配置信息，测试系统能够自动构建生成相应的移植测试环境，将 Java Web 应用部署其中，并调用自动化测试工具、监控工具对测试目标进行测试与监控，同时对运行日志进行分析，综合各项数据结果从多维度评估应用的可移植性，最后为用户输出可视化报告展示测试结果。从而减少测试人员用于搭建移植测试环境的成本，提高可移植性测试效率，帮助企业和开发人员提高 Java Web 应用可移植性。

1.2 国内外研究现状

上个世纪七十年代，Poole 和 Tanenbaum 等人提出软件可移植性的定义 [14, 15]：“如果花费合理的成本让软件在它最初编写的计算机之外的机器上运行，那么它就被认为是可移植的。相同的软件能够在许多不同的计算机上运行时，可移植软件就证明了它的价值”。对于软件开发者而言，软件需要具备耐久性和尽可能低的成本，并且希望进入更大更多样的市场，因此可移植性是不可缺少的属性 [16]。随后，许多软件研究人员逐步认识到软件可移植性的重要性。根

据 Ghandorh 等人统计 [17]，1990 年至 2019 年期间共有超过 7000 篇与软件可移植性相关的文献发表，但其中 83% 均为研究提高软件可移植性的开发方法和工具，仅有 2% 的文献以可移植性测试作为研究主题，这些研究在软件不同发展阶段提出了各种评估模型和指标，如成本评估模型、架构评估模型等。

最早的度量方法是成本评估模型，主要基于软件移植过程所需要的修改成本，Tanaka [18] 等人提出一种基于修改代码行数计算软件可移植性的方法，对软件移植过程中需要修改的代码行数进行计数，并与原有代码总行数进行对比，得出软件可移植性度量值，其计算公式如下：

$$\text{Portability} = 100 * \left[1 - \frac{(\text{number of modified LOC}) * \alpha}{(\text{number of total LOC})}\right] \tag{1-1}$$

其中 LOC 表示代码行数，α 表示修改一行代码工作量与开发一行代码工作量之比。Mooney[19]、Poulin[20] 等人在此基础上进一步改进了评估方法，将移植过程中需要修改代码的功能模块和方法数量纳入考量范围。

随着软件工程发展和软件规模的增长，仅仅考虑软件的代码和功能模块，已经不足以度量软件的可移植性。Hakuta 等人对软件可移植性的度量转变为工程角度，提出衡量软件可移植性的四个方面 [21]：

（1）待移植软件的大小；

（2）待移植软件内容；

（3）移植障碍因素；

（4）移植成本因素。

这四个方面基本涵盖了软件自身影响可移植性的可能因素，但 Washizaki 等人认为，软件的可移植性不仅受到自身因素影响，还应当考虑外部依赖的影响 [22]。由于客户需求的多样化，软件部署运行的环境往往不能完全与预期相同，常会受到客户环境的限制，如操作系统版本。因此，Matinlassi 和 Jönsson 分别提出 QADA [23] 和 SAAM [24] 两种基于场景的分析方法，通过描述和分析软件体系与架构，根据经验分析不同场景对架构元素的影响，从而评估软件整体可移植性。后来 Gafni [25] 进一步完善，提出根据不同设备之间的异构性，将用户配置文件和中间件纳入可移植性测试评估体系中。

近 10 年来，云计算技术快速发展，为软件运行环境带来新的选择，分布式云架构成为一种新的软件设计模式，大量传统软件迁移到云环境中，尤其是 Web 应用的部署。Martino [26] 等人认为云模式可以看作是经典设计模式的演变，通过映射设计和云模式元素，可以将遗留软件移植到云上，基于这个思路

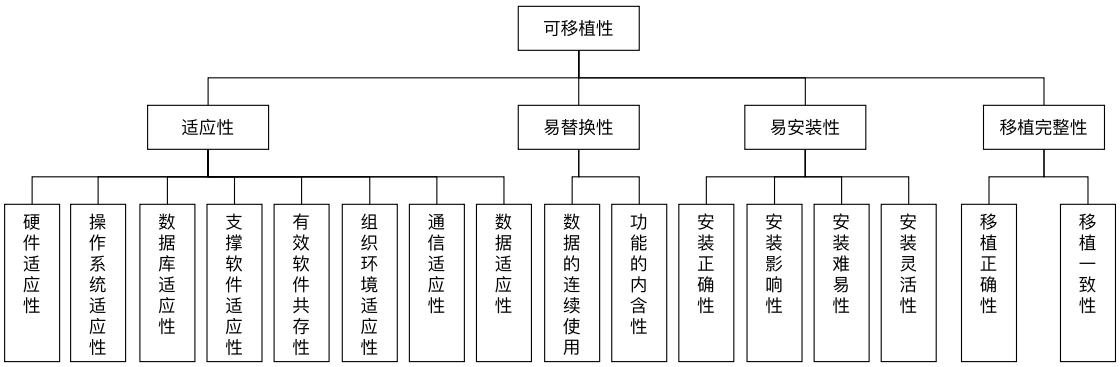

图 1-1: 可移植性测试度量指标

提出一种基于分数的方法来分析设计模式和云模式之间的实际可行性，来衡量遗留软件的可移植性。Angelis [27] 等人在该方法上进行了改进，从软件工程的角度，通过专家问卷的形式，衡量软件的耦合程度、多租户、数据库特征等因素，分析软件移植的难易程度。

上述研究提出的评估模型和方法对 Java Web 应用可移植性测试具有一定参考意义，但这些方法大多需要依赖于具备丰富经验的工程师对软件进行分析，或仅限于特定领域，如遗留软件迁移上云等。由上海计算机软件技术开发中心[①]等单位起草的国家推荐标准《系统与软件可移植性》提出了相对通用且完善的指标体系[②]、度量方法[③]和测试方法[④]。该标准提出从适应性、易替换性、易安装性和移植完整性四个方面对软件可移植性进行度量，其具体指标和因素如图 1-1所示。

其度量方法主要是根据不同指标设置测试环境，将软件成功移植的测试环境个数于测试环境总数进行对比，测试方法包含专家测试法、技术测试法、问卷调查法三种，其中技术测试法最为常用，也为本文提供了借鉴思路。具体的实施步骤可以概括如下：

（1）根据测试需求提取出多个测试环境配置；

（2）根据配置分别搭建测试环境；

（3）将待测软件在各个测试环境中进行部署；

（4）采用专家测试法、技术测试法等方法对各环境中部署好的软件进行功能、性能等测试，进行度量计数；

（5）根据度量方法，计算各个指标。

[①] https://www.ssc.stn.sh.cn/

[②] http://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=D22CB20C9B68154349F870E0BBB657D0

[③] http://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=72B01039A86895B99819EF50369E7A40

[④] http://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=4BEB9227417314C9FF17129AB9D6571E

该标准提出的可移植性测试方案也是目前被大多数测试团队所采用，但该流程主要依赖于人工，其中搭建测试环境进行部署，对移植后的软件进行各项测试均需要耗费大量成本。本文在此方案上，将其中人工操作成本较大的步骤进行自动化，采用容器技术实现测试环境快速生成，同时基于 Web 应用自动化测试技术，对移植部署的应用进行测试，收集数据进行评估。

1.3 本文的主要工作

传统可移植性测试方案用于 Java Web 应用时，由于大量依赖于人工导致测试成本过高、效率低下等问题。本文为解决这些问题，设计并实现一种面向 Java Web 应用可移植性的自动化测试技术，旨在通过自动化生成测试环境、自动化部署与测试目标应用等手段，实现低成本、高效率的可移植性测试，并为开发测试人员提供易于理解的可视化测试报告，从而提高 Java Web 应用的软件质量。本文主要工作具体如下：

（1）本文根据当前时代背景与现状调研，对系统进行需求分析和概要设计，分解出实际功能需求，并设计划分出测试环境生成服务、移植节点控制服务、调度执行服务、自动化测试服务和自动化分析服务等模块，明确了各模块职责并进行整体架构设计。

（2）本文设计实现了测试环境生成模块，其主要功能是根据用户所选择的测试组合分解出不同环境配置，基于 Docker 容器技术自动化构建出相应配置的测试环境，如操作系统、CPU 核心数、依赖版本等。针对每个测试环境配置，通过 Docker-Client 等 SDK 与 Docker Engine 进行交互，控制容器的创建、运行与销毁。

（3）本文设计实现了移植节点控制模块，该模块为整套系统提供弹性扩展移植服务节点的能力，实现服务节点可动态注册发现并自动管理资源状态，针对每次测试包含多个环境下的测试任务，以及任务对服务器类型的要求，提供分布式执行能力。在每个可进行移植部署的节点上启动服务，即可自动将该节点资源注册到业务系统中提供服务。

（4）本文设计实现了调度执行模块，根据各个移植服务节点的资源状态，对系统接收的多个移植任务进行调度执行，以满足每个任务对服务器资源的要求。对于任务过多或资源不足的情况，采用 RabbitMQ 和轮询节点控制服务的方式保证系统的高可用。

（5）本文设计实现了自动化测试模块，对移植部署在测试环境中的 Java Web 应用调用 HttpRunner 工具进行自动化测试，获取应用的接口功能与性能数据。该模块基于 Celery 实现测试任务的异步调度，并且设计实现了一套通用接口标准，以保证测试工具可扩展能力。

（6）本文设计实现了自动化分析模块，通过收集 Docker 容器运行日志，并构建 Java Web 异常经验库，用于分析移植后应用运行存在的问题；同时使用 Prometheus 与 Container Exporter 采集容器运行的各项监控数据；结合自动化测试结果分析 Java Web 应用的可移植性，生成测试报告。

1.4 本文的组织结构

第一章 引言。主要介绍可移植性测试在 Java Web 相关工程领域的重要性，特别是在国家信创战略背景下，Java Web 应用可移植性测试存在诸多问题与难点。之后通过介绍国内外关于软件可移植性的测试理论与方法，引出当前主流方案所存在的现实痛点，最后简要说明本文未解决这些问题所完成的主要工作内容。

第二章 技术综述。主要介绍本文系统中所涉及技术与工具，并从系统设计角度阐述使用该技术的原因和见解。

第三章 系统需求分析与概要设计。根据前期场景分析与用户调研，分析得出用户对该系统对功能性需求与非功能性需求，在此基础上进行系统体系结构设计，并将整个项目划分为测试配置模块、任务调度执行模块、日志分析模块与报告生成模块。

第四章 系统详细设计与实现。本章根据第三章分析设计的基础，对系统各个模块进行详细设计，通过流程图、UML 类图、时序图等对设计思路与实现方式进行阐述，并给出相关模块的核心代码实现在最后给出系统最终使用效果截图与文字描述。

第五章 系统测试。本章介绍系统测试的设计、执行过程和结果。根据前文需求分析与系统设计相关内容，从系统可用性角度进行单元测试、接口测试和验收测试的设计和执行。

第六章 总结与展望，从整体上总结当前系统完成情况，分析系统存在问题和不足，并据此对后续的迭代计划进行展望。

第二章　技术综述

2.1　服务框架相关

2.1.1　Spring Boot

Pivotal 团队在 Spring 框架基础上设计实现了更加轻量级的 Web 应用开发框架——Spring Boot，其旨在尽可能减少配置文件，帮助开发人员快速搭建 Spring 应用来应对日益庞大和复杂的软件规模 [28]。Spring Boot 框架具有如下特点：

（1）Spring Boot 具有强大的自动化配置功能 [29]，Spring 官网[①]提供的配置管理器可根据开发者选择快速生成一个可运行 Spring Boot 项目。另外，Spring Boot 基于“约定大于配置”的思想提供很多高效的基于注解的自动化配置方式，取代繁杂的 XML 配置方式，大幅度提高了开发效率。

（2）Spring Boot 通过内嵌 Tomcat、Jetty 等常见 Servlet 容器，简化了 Java Web 应用部署方式，除了配置 Java 环境外无需进行其他部署配置，降低了部署成本。

（3）Spring Boot 框架提供的 Starter 组件使得项目依赖管理简洁，其会对相关 Maven 依赖的版本关系进行自动管理，让开发人员将精力投注于业务功能，而不是处理错综复杂的 Maven 依赖 [30]。另外，通过 Starter 能够帮助开发者快速集成 RabbitMQ、Redis 等企业级应用程序中常用组件，降低了开发者在代码中整合这些组件的难度，简化了编码过程。

本系统用作 Java Web 应用可移植性测试，在多个服务器扩展移植部署节点，因此对系统的易部署性有较高的要求；另外，由于自动化测试工具需要保证后期可扩展性，在开发系统时应尽量避免过度侵入的配置，减少耦合，并且整套系统需要进行分布式部署，上下游以微服务的方式提供服务，因此本系统选择 Spring Boot 作为基础开发框架。

[①]https://start.spring.io

2.1.2 Celery

在大型系统中,任务调度是一项基础性的需求，对于一些需要重复、定时执行或者耗时比较长的任务经常会被剥离出来单独处理,而随着任务规模与复杂性的上升,任务调度服务框架也就随需而生 [31]。Celery [①]是一款简单易用、灵活扩展且可靠性较高的分布式任务调度框架，它支持实时任务处理和任务调度，能够高效地处理大量消息。Celery 由消息中间件 (Broker)、任务执行单元 (Worker) 和结果存储 (Backend) 三部分组件构成，其工作流程由应用程序通过 Broker 向 Celery 发送任务消息，Worker 通过监听 Broker 获取调度的任务并执行，将任务结果输出到 Backend 进行存储。这种分布式队列能够将系统中不同业务模块依赖的任务处理相互解耦，同时其分布式架构保证了良好的可扩展性。

Celery 基于 Python 开发，其无需配置文件，直接编写 Python 脚本即可定义 Worker 处理任务所要执行的操作，因此使用极其简单且易于维护。Worker 还支持水平扩展，多个 Worker 可运行于不同机器，通过配置监听消息队列来处理相应的任务，Celery 提供了自动重试机制，令 Worker 尽量避免由于连接丢失等情况导致的执行失败。另外，Celery 具有强大的扩展和集成能力，能够快速与 Flask、Django 等框架整合开发，同时其自身不提供消息服务，而是让用户根据自身系统的技术架构选择合适的第三方消息中间件，支持与 RabbitMQ、Redis、Amazon SQS 等主流组件集成。

本文设计的系统中，需要调用自动化测试工具对在移植环境中部署好的 Java Web 应用进行测试，该过程是一个异步执行的任务，因此本文基于 Celery 开发一套自动化测试服务框架，对测试任务进行调度，保证任务能够稳定执行，同时根据定义不同 Worker 和队列，能够方便地扩展接入不同的测试工具。

2.1.3 Docker 容器技术

Docker [②]是一种使用 Go 语言开发的开源容器技术，在 Docker 中构建的应用程序可以与所需依赖环境一起打包成标准镜像形式。这些镜像能够通过简单的操作命令在任何安装了 Docker 引擎的机器上快速生成容器，运行应用程序，完成应用的部署 [32]。由于容器中包含了应用程序运行所需的所有依赖环境，所以容器能够完全脱离特定的物理设备或软件环境，快速地在不同服务器中创

[①]https://github.com/celery/celery
[②]https://www.docker.com/

建应用的所需依赖，降低了测试与运维人员搭建不同运行环境的成本，大幅度提高了部署效率。

Docker 容器的本质是进程 [33]，不同于宿主机中直接执行的进程，容器拥有其独立的资源环境。Dcoker 底层基于 Linux Container（LXC）技术实现，利用 Linux 提供的 Nampspace 和 Cgroup 技术，能够 Docker 在操作系统层上对硬件资源进行虚拟化，并构建沙盒环境让容器以一种隔离的方式在操作系统内核上运行。这样的机制保证了应用的安全性不受外界干扰 [34]。

Docker 镜像是容器运行的基础，提供了容器运行所需的程序、库、资源、配置等文件，还包含运行时参数，如匿名卷、环境变量等。Docker 提供了声明式脚本 Dockerfile 来帮助开发者快速构建镜像，镜像采用分层存储的方式进行构建，开发者可以基于已有镜像，进一步添加新的内容，这使得镜像的重用与定制变得十分方便。

根据 Docker 所具备的优势与特性，十分适用于本系统中自动化生成移植部署的测试环境，通过离线定制出 Java Web 应用常见运行环境，在运行时利用 Docker Engine 开放出的接口来限制容器运行时资源，代替人工构建不同的运行环境。另外，Docker 还可以用于本系统自身服务与测试工具的封装，方便快速进行测试和部署。

2.2 测试与监控工具相关

2.2.1 HttpRunner

HttpRunner [①]是一个基于 Python 开发用于 HTTP/HTTPS 协议接口的开源通用测试框架，其使用简单，通过编写 YAML/JSON 脚本来维护 Web 接口测试用例，以及接口之间的逻辑关系，HttpRunner 框架原理如图 2-1所示。

使用 HttpRunner 能够实现对 Web 应用的自动化接口测试、性能测试、线上监控、持续集成等多种测试需求。其核心特性如下：

（1）继承了 Python Requests 包的全部特性，支持 Cookie 与 Session 维持等功能，轻松实现 Web 应用接口的各种测试需求。对于接口返回结果支持丰富的校验机制，借助 debugtalk 辅助函数，还能够在测试中实现更加复杂的动态计算逻辑。

[①]https://github.com/httprunner/httprunner

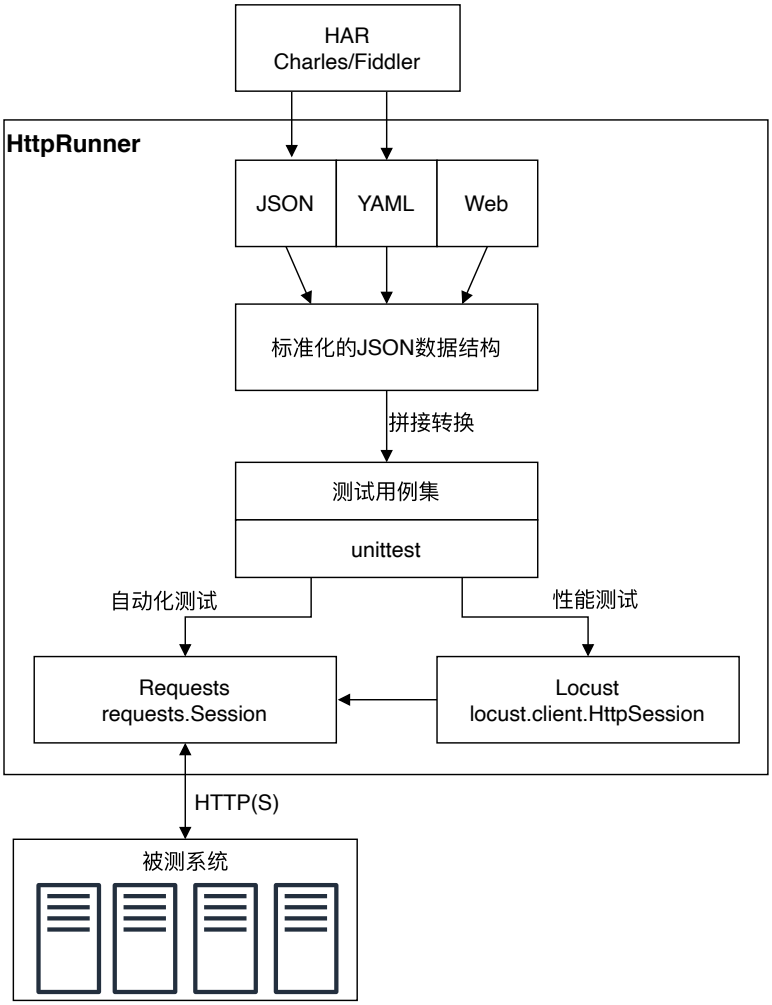

图 2-1: HttpRunner 原理图

（2）使用 YAML/JSON 脚本描述测试场景，测试人员能够轻松定义接口的类型、参数和返回值等信息，提高了测试用例的可维护性。同时采用完善的测试用例分层机制，能够根据工程项目的结构对测试用例进行划分和复用，YAML 格式脚本格式如图 2-2所示。

（3）采用命令行调用方式，同时支持 hook 机制，在测试前后可定义 hook 动作，方便与 Jenkins 等持续集成平台配合使用。

（4）基于 HAR 和 har2case 能够实现接口录制和测试用例生成，减少测试人员编写用力脚本的工作量。另外，基于 HttpRunner 的脚本，结合 Locust 无需额外工作即可实现分布式性能测试。

（5）HttpRunner 支持输出 HTML 格式的测试报告，报告中简洁清晰地展示了测试结果统计，并包含详细的测试数据和日志。同时，支持自定义报告模

```
- config:
    name: testcase description
    variables: {}
- test:
    name: /api/users/1000
    request:
        headers:
            Content-Type: application/json
            User-Agent: python-requests/2.18.4
            device_sn: FwgRiO7CNA50DSU
            token: baNLX1zhFYP11Seb
        json:
            name: user1
            password: '123456'
        method: POST
        url: http://127.0.0.1:5000/api/users/1000
    validate:
        - eq: [status_code, 201]
        - eq: [headers.Content-Type, application/json]
        - eq: [content.success, true]
        - eq: [content.msg, user created successfully.]
```

图 2-2: HttpRunner 接口定义 YAML 示例

板，在执行测试时可动态指定模板。

本系统中需要对在移植环境中运行的 Java Web 应用进行测试，以评估其功能、性能等状态。HttpRunner 使用简单、功能强大，因此本文将其封装成 Docker 镜像，接入 Celery 中作为任务处理工具形成自动化测试服务框架。

2.2.2 Prometheus

Prometheus [①]是由 SoundCloud 公司 [②]开源的一套轻量级系统监控与告警解决方案。自 2012 年进入开源社区后，获得大量开发者的青睐，并于 2016 年加入云原生计算基金会（CNCF) [③]，成为继 Kubernetes 后第二个进入该基金会的开源项目 [35]。

Prometheus 采用 Go 语言开发，其内置了时序数据库，不依赖于分布式存储，单节点即可工作。Prometheus 将所有数据存储为时间序列，具有相同度量

[①]https://prometheus.io/
[②]https://soundcloud.com/
[③]https://cncf.io/

名称以及标签属于同一个指标。每个时间序列都由指标名称和一组键值对来唯一标识，并支持使用 PromSQL 进行查询，PromSQL 是一种灵活的查询语言，可以对多维数据进行复杂的查询。结合 AlertManager，定义不同的报警规则，可实现异常状态报警，并可以通过邮件和 Webhook 等方式通知给运行人员；另外，Prometheus 还支持多种图形模式，以及和 Grafana 等可视化仪表盘进行集成，用于展示监控数据。

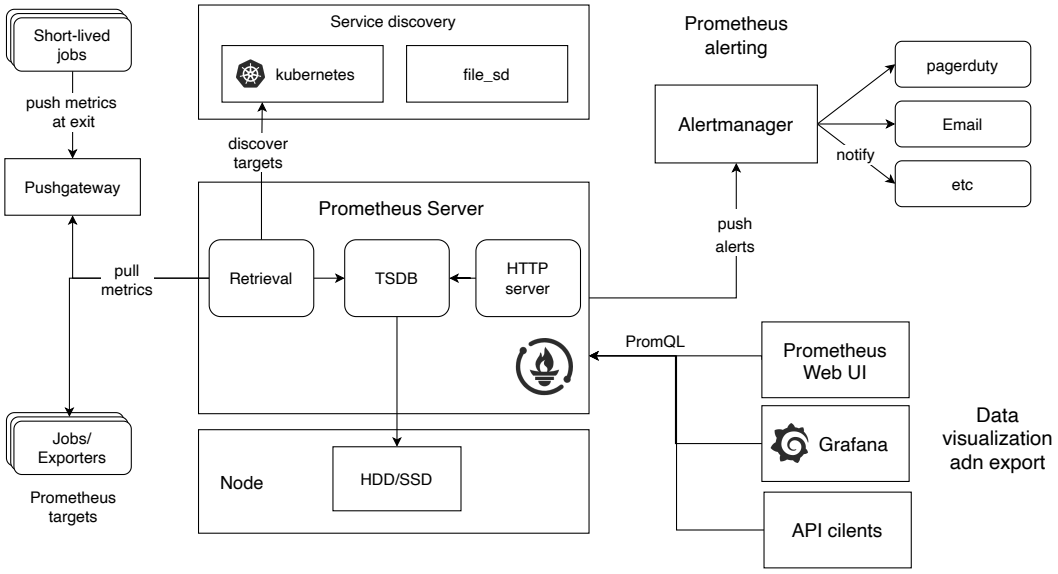

图 2-3: Prometheus 架构图

Prometheus 由多个模块组件构成，其架构如图 2-3所示。其中核心组件 Prometheus Server 用于收集指标和存储时序数据，并对外提供查询接口。Prometheus 可通过配置文件或服务发现两种方式配置监控对象，采于 HTTP 轮询的方式来拉取监控数据。由于实践场景中，可能存在监控对象过多或 Prometheus 于监控对象之间网络无法直接相通的情况，因此 Prometheus 提供了 Pushgateway 组件作为监控数据的中转站，各个监控对象将数据使用 Push 的方式推送到 Pushgateway 中，Prometheus 直接从 Pushgateway 拉取数据。Prometheus 提供了多种 Exporter 用于暴露已有第三方服务的指标数据，如用于服务器监控的 Node Exporter，并支持自定义扩展；此外 Prometheus 还具有丰富的客户端 SDK，支持 Java、Go、Python 等，用于在服务代码中暴露相应的指标接口给 Prometheus。

本系统中需要对用作移植部署环境的 Docker 容器进行监控，以获取 Java Web 应用在目标环境中运行时的 CPU、内存等使用情况；因此，本文选择

Prometheus 作为容器监控方案，并搭配 Container Exporter 采集监控数据。

2.3 数据服务相关

2.3.1 RabbitMQ

随着软件系统规模不断扩大，大型 Web 应用往往基于分布式体系结构，对各个功能拆解细分，进行模块化和微服务化，从而达到功能上低耦合高内聚的目的，并提高各个模块的复用程度。这些模块和微服务作为大型 Web 应用的组件往往需要以异步的方式进行调用与访问，并克服点对点通信的限制 [36]。

RabbitMQ [①]是一款应用使用最为广泛的开源消息中间件。其基于 Erlang 语言开发，实现了面向消息中间件设计的应用层开放标准协议——AMQP 协议。该协议能够规避编程语言、软件类型的限制，实现在客户端与中间件的消息通信。RabbitMQ 最早诞生并应用于金融行业，在大型分布式应用系统中表现优秀，在转发和存储消息的同时保证良好的扩展性和高可用。详细特点如下：（1）RabbitMQ 使用持久化、传输确认、发布确认等机制保证了消息的高可靠性。并且提供跟踪机制，方便使用者排查异常消息的状态和原因。（2）RabbitMQ 支持集群化，多个 RabbitMQ 服务器能够搭建成集群，在逻辑上表现为一个 Broker；使用集群时，队列会在多个服务器上进行备份，当部分部分节点宕机，仍可以保证队列不会丢失或失效，保证了服务高可用。（3）RabbitMQ 支持 STOMP、MQTT 等多种消息队列协议，并且提供 SDK 与服务代码集成，支持 Java、Python、Ruby 等几乎所有常用编程语言。

本系统在执行移植测试任务时，需要根据各个移植部署节点的状态对任务进行调度分配，在特定情况下需要阻塞等待，该过程是一个异步的过程，因此选用 RabbitMQ 作为本系统的消息中间件保证移植测试任务的稳定执行。同时作为自动化测试服务框架中 Celery 的 Broker 为系统提供服务。

2.3.2 Redis

Redis [②]是一款基于内存的高性能开源 key-value 数据库 [37]，在实践中被许多企业和开发者用于消息中间件、缓存和数据持久化。

[①]https://www.rabbitmq.com
[②]https://redis.io/

Redis 具有许多优良特点，其为开发者提供了多种数据类型，包括字符串 (string)、列表 (list)、哈希 (hash)、集合 (set)、有序集合 (zset) 和 HyperLogLog [38] 等。Redis 使用 C 语言开发，能够方便地与操作系统交互，并且底层基于内存实现，能够直接操作内存进行数据读写；Redis 使用的单线程模型减少了多线程上下文切换和线程同步产生的资源消耗，并且保证了操作的原子性，另外通过 I/O 多路复用，Redis 的单线程也能够处理大量的连接请求。基于上述这些设计，Redis 具备了极高的读写性能和吞吐量，根据官方文档，其读写速度分别能达到 11 万次/秒和 8 万次/秒。

Redis 虽然是基于内存的数据库，但同样支持将数据持久化到磁盘，它提供 AOF(Append-Only-File) 和 RDB 快照两种方式对数据进行持久化存储。AOF 的原理是记录日志，开启这种 AOF 后，Redis 会对每次写操作的指令和数据记录到日志中，然后通过回放操作来还原数据到内存中，这种方式能够保证数据的完整性，但随着 Redis 运行，日志文件会不停增长占用磁盘空间，并且 AOF 恢复速度较慢。RDB 快照则与 AOF 相对，其根据配置文件在指定时间间隔时对数据进行快照，将数据写入到临时文件中，持久化结束后会用这个临时文件替换上次快照的文件。RDB 使用单独的子进程进行持久化，主进程不会进行任何 I/O 操作，避免了对 Redis 性能的影响，但 RDB 每次快照之间存在一段时间间隔，如果两次持久化操作之间可能会发生数据丢失，影响数据完整性。两种方法各有优劣，所以目前主流解决方案是同时使用 RDB 快照与 AOF，AOF 仅记录每次快照后的数据，保证数据完整性的同时尽可能减少 AOF 产生的日志。

本系统在设计自动化测试服务框架时需要为 Celery 接入 Backend 进行测试结果存储，每次 Celery 调用测试工具测试完成后，会通过消息队列通知业务系统任务完成，但是测试工具的输出数据可能十分庞大，用消息队列传输大量结构化数据不是合适的解决方案。而且，不同测试工具输出结果可能是异构的，关系型数据库显然不适用。另外，为了保证系统对测试工具的扩展性，业务系统拿到测试结果数据后会进行清洗和处理，原始数据无需长时间存在。因此本系统选用 Redis 作为缓存数据库。

2.4 本章小结

本章主要概述项目所使用的相关技术框架和工具，通过对各技术和工具的优点和特性进行介绍和分析，结合项目研发和架构需求，阐述做出相应选择的

原因。首先对搭建系统核心服务所需技术选型 Spring Boot 框架、Celery 框架和 Docker 技术进行介绍；其次，对可移植性测试所需要的自动化测试工具和监控服务展开说明，主要包括 HttpRunner 和 Prometheus，这两者可以帮助我们获取移植部署后 Web 应用的状态和一致性数据；最后阐述 Redis 和 RabbitMQ 的特点，说明了本项目选用二者作为缓存数据库和消息中间件的原因和必要性。

第三章　需求分析与概要设计

本章根据业务需求对系统进行需求分析和概要设计，梳理规划系统的目标功能范围。使用软件工程的方法，根据 Java Web 应用可移植性自动化测试技术的应用场景，分析出系统功能性需求和非功能性需求，并进行用例设计和描述。明确系统需求后，划分出功能结构，最后对系统整体架构、自动化测试流程和数据持久化存储进行设计。

3.1　系统整体概述

近年来，Web 应用和服务日益深入工业生产和人民群众的生活，随着操作系统、云计算、容器化等技术快速发展更新，Web 应用的架构体系、部署方式，也随之不断升级变化，其部署运行的软硬件环境愈发多样化。同时，由于市场需求的快速增长与多变，企业提供公网产品和服务的同时，对私有云产品和解决方案的需求也不断增正。在私有云环境下，不同客户提供的服务器在操作系统、硬件环境配置和软件环境依赖等方面可能各不相同且存在诸多限制，Web 应用需要在不同的客户环境下成功部署运行，并提供完善的功能。另一方面，为响应国家信创战略，改变我国核心系统软件在国际上受制于人的现状，诸多爱国软件厂商、服务商将其产品与国产服务器、操作系统等软硬件的适配性测试提上日程。

因此，客户与厂商均对 Web 应用的可移植性提出越来越高的要求。Java 作为生态完善、受欢迎程度较高的编程语言，使用其开发的 Web 应用和服务市场占比也较高，因此如何方便快速地测试 Java Web 应用的可移植性，逐渐成为被许多应用服务厂商所重视的问题。但是目前主流的可移植性测试方案主要由测试人员手动搭建不同测试环境，依赖于经验或专家进行分析，测试效率低下且成本高昂。

针对上述现状，本系统为用户提供移植测试环境自动化构建生成、Java Web 应用自动化部署、自动化测试与监控、分析评估等功能，旨在降低可移植性测试成本，提高测试效率，并输出有效的测试报告为开发人员改进应用提供

参考。图 3-1表示本系统进行 Java Web 应用可移植性测试的总体过程。本系统进行可移植性测试时，首先由用户选择组合多个移植环境配置和 Web 应用的参数，包括测试环境的操作系统版本，CPU 和内存等硬件资源配置，支撑软件版本，以及 Java Web 应用的接口列表和启动命令；系统根据配置所需的资源在部署节点集群上进行调度分配，按需求生成多个测试环境的 Docker 容器并运行 Java Web 应用；系统根据移植部署好的应用访问 ip 和接口列表生成自动化测试任务，在测试过程中通过监控服务进一步收集应用运行状态信息；自动化测试任务结束后，系统会对测试结果、应用运行日志和监控数据进行分析，输出测试报告。

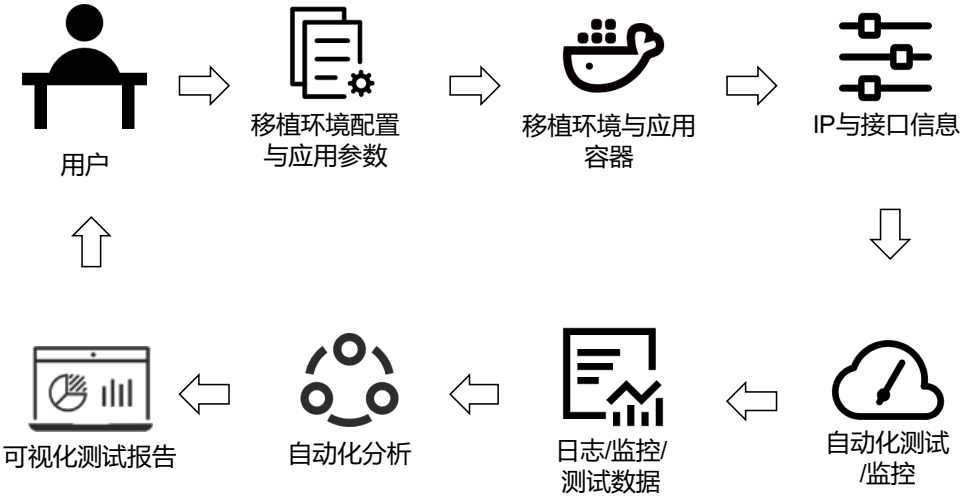

图 3-1: Java Web 应用可移植性自动化测试过程

3.2 系统需求分析

3.2.1 系统涉众分析

本系统为用户提供面向 Java Web 应用可移植性的自动化测试服务，其涉众分析结果如表 3-1所示，只存在测试需求方这一种涉众。

作为测试需求方，可以使用本系统进行上传待测应用，发布测试任务，查看测试任务的执行情况，查看测试报告等操作。测试需求方通常具有一定开发能力，并且有能力理解测试报告中的各项测试结果和监控数据，对报告中影响可移植性的因素或代码能够进行判断和验证；同时具有一定的决策和分析能力，可以根据系统提供的测试报告对目标应用的可移植性进一步的评价和

决策。

测试需求方希望借助于本系统，只填写少量配置即可快速自动生成多种移植部署测试环境，系统能够对运行在测试环境中的 Java Web 应用，进行自动化测试和监控，并通过自动化分析输出可读性较高的测试报告。

表 3-1: 系统涉众分析结果

涉众名称	涉众特征与期望
测试需求方	作为 Java Web 应用可移植性测试的需求方，他们期望在提交待测应用的 Jar/War 包之后，本系统能够根据其需求自动化生成多种配置的移植测试环境，用于运行应用，并对应用进行自动化测试，生成测试报告。测试需求方具有一定的应用开发能力，能够理解测试报告，判断或验证报告中功能、性能等数据的真实性；并且具有根据测试报告进行评分与决策的能力。

3.2.2 功能性需求分析

本文主要从系统业务功能、测试环境生成、自动化测试、自动化分析四个部分对系统的功能性需求进行分析，表 3-2展示了功能性需求的具体信息。

系统通过 Web 页面为用户提供系统进行交互的相关业务功能，主要包括上传待测应用、发布测试任务、停止任务、查询任务状态、查看测试结果报告等。用户通过这些功能来完成对 Java Web 应用可移植性的完整测试流程。

测试环境生成模块是在用户发布测试任务后，按照用户填写的任务配置信息与参数组合，自动生成 Docker 容器作为应用部署运行的测试环境。根据 Java Web 应用部署常见的环境因素，系统提供操作系统版本、计算资源等选项供用户组合配置，并自动分配内网 IP 和端口，用于接入自动化测试服务。

自动化测试模块的主要功能是基于 Celery 框架实现自动化测试工具的调用，对移植部署好的应用的功能正确性和性能表现进行测试验证。通过这些测试能够验证移植部署后应用的功能和性能是否符合预期，并将测试结果输入到自动化分析模块统一进行分析评估，生成测试报告。

自动化分析模块为用户输出可移植性测试结果和报告。为了评估 Java Web 应用的可移植性，系统会对其运行日志进行收集，同时使用监控组件对其状态进行监控，结合自动化测试服务输出的测试结果，综合分析整理，生成可视化测试报告，通过 Web 页面进行展示。

表 3-2: 系统功能需求列表

需求 ID	需求名称	需求描述
R1	上传测试目标	测试需求方可以在系统中上传待测的 Java Web 应用安装包，支持 Jar、War 两种格式。
R2	发布测试任务	任务发布者通过填写少量配置与参数，即可发布一个可移植性测试任务，所需信息与参数包括任务名称、测试目标、接口信息、测试环境配置。
R3	查询任务状态	任务发布者可查看测试任务当前状态，各个测试环境执行状态。任务状态包括未开始、运行中、执行出错、已完成。
R4	停止测试任务	任务发布者可以随时终止已开始的可移植性测试任务，在终止后，仍可以重新执行该任务。
R5	环境自动生成	任务发布者选择待测应用，并选择可移植性测试的环境配置，系统自动生成相应环境并运行待测应用。测试环境配置选项操作系统版本、CPU 核心数、内存大小、Java 版本、Tomcat 版本，需支持扩展。
R6	自动化测试	系统提供对移植部署后的 Java Web 应用进行自动化测试的能力，包括功能测试与性能测试。
R7	自动化分析	系统对 Java Web 应用的运行日志、监控数据、自动化测试结果进行分析评估，给出功能表现、性能表现、移植效率和适应性四个维度评分。
R8	查看测试报告	测试任务完成后，任务发布者可通过 Web 页面查看测试结果报告，报告内容包含可移植性评分，每个测试环境下的运行日志、报错分析、功能与性能测试结果、运行中监控数据。

3.2.3 非功能性需求分析

表 3-3: 系统非功能需求列表

名称	描述
稳定性	本系统应当能在突发流量或测试任务过多时保证稳定运行，对超出可用资源的访问请求提供排队机制。
可用性	本系统应该保证正常情况下能够持续运行，对于错误输入、意外情况等能够有正确的处理。如遇到断电、服务器重启等特殊情况，应当能在主机恢复后 15 分钟内重启提供服务。
可扩展性	本系统应对测试环境生成服务、自动化测试服务进行抽象，提供扩展接口，方便未来迭代升级，如新增测试环境支持的操作系统、软件依赖，增加自动化测试工具等。
性能	在稳定的内网环境或带宽 4M 以上的公网中，本系统的接口返回时间应该不大于 300ms，对数据库进行读写操作的响应时间应小于 500ms。
兼容性	本系统应支持在主流 Linux 发行版的最新版本和稳定版本上部署和执行，并且支持至少一种主流国产服务器和操作系统。
伸缩性	当移植部署节点资源无法满足当前测试任务需求时，应能快速添加新的移植部署节点加入到集群中提供服务。

本系统定位是为企业与开发者提供可移植性测试服务，为保证服务质量，本系统除了满足功能性需求外，还应该具备良好的性能、可用性、稳定性，并且提供良好的可扩展性为未来的迭代升级服务。具体的非功能性需求如表 3-3所示。

3.2.4 系统用例图

经过对系统进行涉众分析和功能性需求分析，梳理得到本系统用例如图 3-2所示。作为测试人员，使用本系统进行 Java Web 应用可移植性测试的过程总体共包含测试目标管理、发布测试任务、停止测试任务、查询任务状态、查看测试报告五个用例。

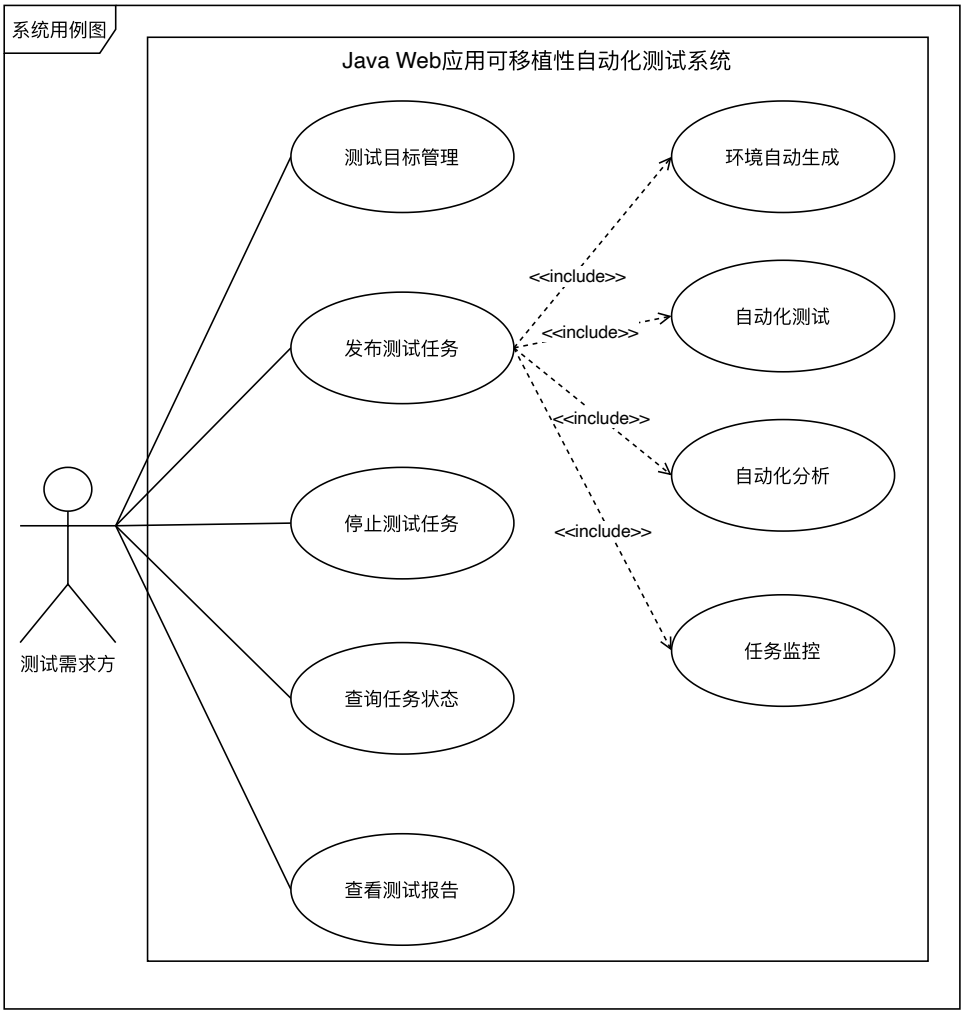

图 3-2: 系统用例图

系统用例反映用户在使用系统时的操作场景，根据前文对功能性需求的分

析，系统用例应当覆盖功能需求，用例与功能需求对应关系如表 3-4所示。

表 3-4: 系统用例与需求对应关系表

用例 ID	名称	对应需求编号
UC1	上传待测目标	R1
UC2	发布可移植性测试任务	R2,R5,R6,R7
UC3	停止可移植性测试任务	R4
UC4	查询可移植性测试任务运行状态	R3
UC5	查看可移植性测试报告	R8

3.2.5 系统用例描述

上传 Java Web 应用测试目标是进行可移植性自动化测试的基础，测试需求方需要上传系统能够接受的应用安装包，才能进行后续操作。上传测试目标的详细用例的详细描述如表 3-5所示。

表 3-5: 上传待测目标用例详情

用例编号	UC1
用例名称	上传待测目标
优先级	高
参与者	测试人员，目的是上传待测应用至系统
前置条件	测试人员进入待测应用管理页面
后置条件	上传是否成功
触发条件	测试人员点击“上传”按钮
正常流程	1. 测试人员进入待测应用管理页面 2. 测试人员点击上传待测应用按钮，选择本地文件 3. 系统将文件上传到服务器进行存储，提示上传成功
扩展流程	2a. 测试人员上传文件的格式非法 1. 系统终止请求并返回异常信息 2. 系统提示“请上传 Jar/War”

发布测试任务是本系统的关键功能，是可移植性自动化测试流程的开端。测试需求方需要填写与测试任务相关的基本信息，测试环境的配置参数，以及 Java Web 应用的功能接口列表，任务发布后即开始整个执行可移植性测试任务。该用例的详细描述如表 3-6所示。

表 3-6: 发布可移植性测试任务用例详情

用例编号	UC2
用例名称	发布可移植性测试任务
优先级	高
参与者	测试人员，目的是创建并执行可移植性测试任务
前置条件	测试人员已上传测试目标并正确填写配置信息与参数
后置条件	可移植性任务是否发布成功
触发条件	测试人员点击确认创建任务按钮
正常流程	1. 测试人员进入创建可移植性测试任务页面 2. 测试人员填写配置信息、参数，选择测试目标 3. 测试人员点击确认创建按钮 4. 系统接执行任务请求，开始分配资源，调度执行该任务
扩展流程	可移植性测试任务缺少所需的参数或配置 1. 系统终止请求并返回异常信息 2. 系统根据所缺信息给出弹窗提示信息

表 3-7: 停止可移植性测试任务用例详情

用例编号	UC3
用例名称	停止可移植性测试任务
优先级	高
参与者	测试人员，目的是手动停止正在运行中的可移植性测试任务
前置条件	该可移植性测试任务必须处于运行状态中
后置条件	系统提示可移植性测试任务停止成功
触发条件	测试人员点击“停止任务”按钮
正常流程	1. 测试人员已成功启动可移植性自动化测试任务 2. 测试人员进入到任务详情页 3. 测试人员点击“停止任务”按钮 4. 系统收到请求后根据任务 ID 停止任务，释放资源
扩展流程	测试人员操作的任务未处于运行状态中 1. 系统终止请求并返回异常信息 2. 系统提示“操作失败，该任务已停止” 测试人员停止的任务不存在或已被删除 1. 系统终止请求并返回异常信息 2. 系统提示“不存在的可移植性测试任务”

停止测试任务为用户提供终止执行中测试任务的能力，以应对用户操作失误等情况。该用例的详细描述如 3-7所示。停止测试任务只针对已成功运行的可移植性测试任务，直接终止对应的任务进程，但是由于需要释放部署与测试资源，该过程需要一段时间才能完成。

表 3-8: 查询可移植性测试任务运行状态用例详情

用例编号	UC4
用例名称	查询可移植性测试任务运行状态
优先级	高
参与者	测试人员，目的是查看某个可移植性测试任务当前的运行进度和状态
前置条件	测试人员已成功发布并运行该可移植性测试任务
后置条件	无
触发条件	测试人员进入可移植性测试任务列表或详情页面
正常流程	1. 测试人员已成功发布可移植性自动化测试任务 2. 测试人员进入到任务列表或任务详情页 3. 系统获取任务当前状态，通过不同颜色的标签与文字信息展示
扩展流程	3a. 测试人员所查询的可移植性测试任务不存在或已被删除 1. 系统终止请求并返回异常信息 2. 系统提示“不存在的可移植性测试任务”

表 3-9: 查看可移植性测试报告用例详情

用例编号	UC5
用例名称	查看可移植性测试报告
参与者	测试人员，目的是在任务运行结束后查看测试报告信息
优先级	高
前置条件	该测试任务已执行完成
后置条件	测试人员可查看可移植性测试报告
触发条件	测试人员进入可移植性测试任务详情页
正常流程	1. 可移植性测试任务已成功执行并正常完成 2. 测试人员进入到可移植性测试详情页 3. 系统展示可移植性测试结果报告
扩展流程	3a. 测试人员查看的任务处于未开始状态 1. 系统获取当前任务信息与状态 2. 系统展示任务已有信息
扩展流程	3a. 测试人员查看的任务不存在或已被删除 1. 系统终止请求并返回异常信息 2. 系统提示“不存在的可移植性测试任务”

查询任务状态功能是帮助用户了解可移植性测试任务的执行情况，除了可以查看该任务的总体状态，还为用户展示各个测试环境中被测应用运行情况与测试状态。可移植性测试任务的状态包括未开始、运行中、执行出错、已完成。该用例的详细描述如表 3-8所示。

在可移植性测试任务执行结束后，用户可以查看该任务的执行结果与测试

报告。测试报告用于帮助用户理解和评估 Java Web 应用的可移植性程度，其内容包括：任务基本信息、可移植性多维度评分，各个移植部署环境下应用的运行日志、资源消耗监控数据、功能与性能测试结果等。查看测试报告用例描述如表 3-9所示。

3.3 系统总体设计

3.3.1 系统整体架构设计

为满足上一节需求分析中对系统提出的一系列功能性和非功能性需求，对系统整体架构设计如图 3-3所示。本系统总体采用前后端分离的模式进行开发和部署，前后端之间采用 RESTful 风格的 Http 接口进行通信和数据传输，保证了页面渲染与业务逻辑的解耦。

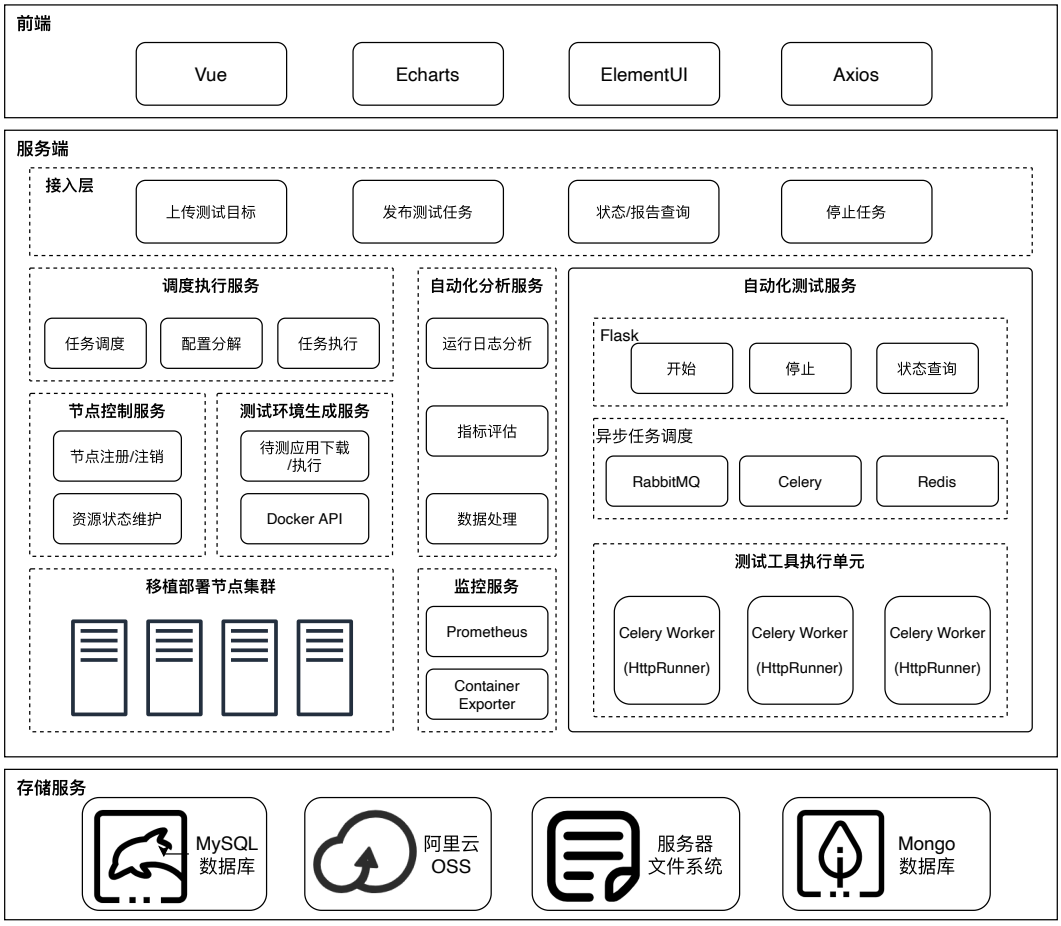

图 3-3: 系统整体架构图

系统前端使用业内较为流行的 Vue 框架进行开发，它是一套用于构建用户界面的渐进式框架，能够方便地与第三方库进行整合，支持丰富的组件来简化开发。本系统通过整合 Echarts 图表框架和 ElementUI 组件，为用户渲染交互良好，美观易用的前端界面；为了方便与后端进行通信，将接口调用交由封装了 Http 请求的 Axios 组件进行处理，让前端开发更集中于数据处理和页面逻辑。

系统服务端通过接入层对外提供服务调用接口。除自动化测试服外，其余所有服务均使用 Spring Boot 框架开发，其对微服务构建提供了良好的支持，能够快速开发 RESTful 接口，并且方便地整合各类组件。Spring Boot 自带的 RestTemplate 组件，提供了简单易用的 API 辅助调用 Http 接口，从而实现各个服务模块之间的相互通信。

调度执行服务主要负责对可移植性测试任务的配置进行分解，根据测试环境拆分为多个子任务，并调用节点控制服务获取移植部署集群中各个节点的资源状态，对子任务进行调度分配；同时通过调用其他服务统一控制可移植性测试任务的执行过程。

节点控制服务运行于各个移植部署节点上，其负责节点的注册、注销和状态维护。测试环境生成服务通过操作 Docker，自动化生成符合配置的容器作为测试环境，并下载测试目标的安装包，将其在容器内启动。

自动化测试服务基于 Celery 框架实现对自动化测试工具 HttpRunner 的异步调用，并整合 Flask 对外提供了统一的调用接口；为了保证测试工具可扩展，将测试工具封装成定义标准接口的 Docker 镜像，后期增加新的测试工具只需构建出符合标准的 Docker 镜像即可。监控服务基于 Prometheus 和 Container Exporter 搭建，用于获取移植部署节点上 Docker 容器的各项数据，从而达到监控待测应用的目的。

自动化分析服务负责对应用运行日志进行分析，通过查询异常经验库，识别应用运行中报错信息和原因；同时对自动化测试工具输出的结果和监控数据进行处理，评估 Java Web 应用的可移植性指标。

本系统存储服务主要使用 MySQL 数据库、Mongo 数据库、阿里云 OSS 对象存储服务和服务器文件系统。MySQL 用于对系统业务数据进行持久化；Mongo 负责以文档形式对可移植性测试结果数据进行存储；OSS 负责存储测试目标的安装包和 HttpRunner 脚本等文件，提高文件下载的性能和高可用；服务器文件系统主要负责存储可移植性测试任务执行过程中产生的日志等中间数据；组合使用这些存储服务，提高系统整体性能。

3.3.2 4+1 视图

4+1 视图 [39] 是 Philippe Kruchten 教授提出的一种采用可视化图形结构描述软件体系架构的方法。该方法从软件的用户场景、逻辑、开发、进程和物理五个不同角度的视图来对软件进行说明。大型软件系统由于架构复杂，单纯的使用整体架构图和描述难以将其说明清楚，对开发、运维等人员而言难以快速理解；因此采用 4+1 视图，从系统不同参与人员的角度来剖析整体架构，能够更好地对系统架构设计进行说明。

用户场景视图已在需求分析中通过用例图和用例描述等方式说明，本章节中不再重复赘述，下面主要对系统的其他四个视图进行阐述。

（1） 逻辑视图

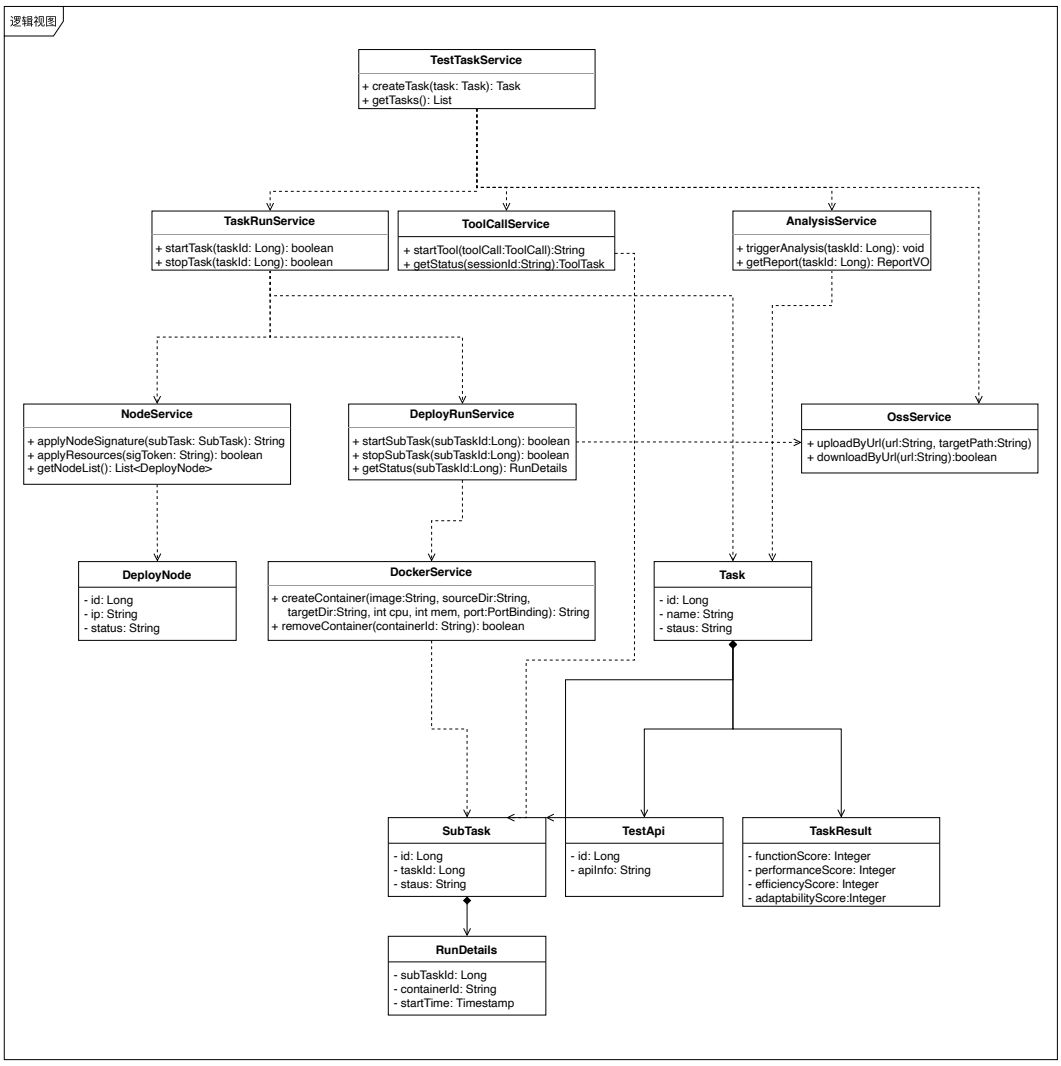

图 3-4: 系统逻辑视图

逻辑视图基于用户的场景和角度，对系统的主要功能和业务逻辑进行描述说明，通常使用面向对象的设计方法，如 UML 类图，将系统拆解为一系列功能抽象，并与用户的功能性需求相对应。本系统的逻辑视图如图 3-4所示。

TestTaskService 表示可移植性测试任务信息相关的系统功能模块，提供任务相关的管理业务功能接口，包括创建可移植性测试任务、运行和停止测试任务，查询结果和状态等，其处理前端或其他服务的请求数据，并返回相应的业务数据。TaskRunService 负责对测试任务调度执行，并通过调用其他服务接口控制整个可移植性测试过程和步骤。NodeService 是节点控制模块，其运行于移植部署集群的每个节点上，集群节点信息抽象为 DeployNode 对象，NodeService 通过维护 DeployNode 对象实现对移植部署节点的注册、注销、资源状态查询和更新等功能，为系统提供动态伸缩移植部署节点的能力。DeployRunService 提供移植环境生成、销毁和待测应用运行等功能。DockerService 负责与 Docker 底层 API 进行交互，创建和销毁 Docker 容器。OssService 对文件上传、下载等功能进行抽象和封装，负责与阿里云 OSS 进行交互。TooCallService 通过请求自动化测试服务接口，提供对自动化测试工具的调用功能。AnalysisService 负责对可移植性自动化测试任务执行中产生的运行日志、监控数据、测试工具结果等数据进行处理，并对应用的可移植性进行计算评估，输出测试报告数据。Task、SubTask、TestApi、TaskResult、RunDetails 等对象是对可移植性测试中各类数据的抽象。

（2） 开发视图

系统的开发视图是基于开发人员视角，根据开过程中代码、文件的组织形式来说明软件架构，主要展示前后端代码和静态资源的包和目录组织结构。本系统的开发视图如图 3-5所示。本系统基于分层架构思想将系统分为三层，分别为展示层、业务逻辑层和数据存储层。展示层主要对前端代码和静态资源进行管理，App 包用于存放前端 Vue.js 的代码，其中根据功能模块会进一步划分；Component 包中包含定义的通用组件或第三方组件；Css 包存放前端央视文件；Assets 包统一存放前端静态资源，如图片、i18n 资源、PDF 等。

业务逻辑层根据 MVC 模式进行了包组织，Controller 层对请求进行校验和分发；Service 层实现了系统具体功能，负责对业务逻辑和数据进行处理；Dao 层抽象了数据库读写相关操作，为 Service 层提供数据查询、更新等接口；Model 包中存放系统所需对象实体；Commons 包中存放一些公共类和静态常量，如异常对象、枚举类、字符串常量等；Utils 包将 YAML 解析、文件 I/O、

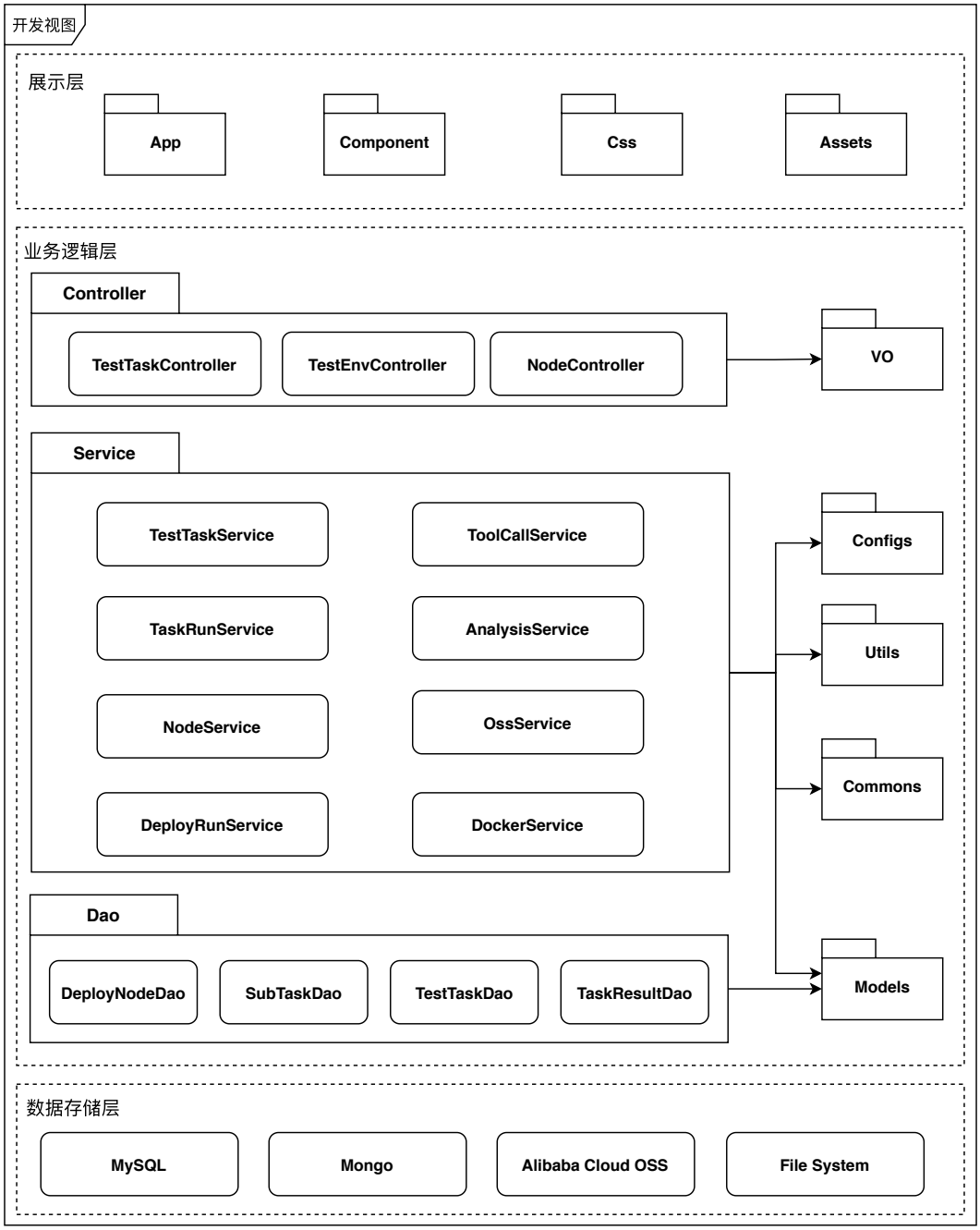

图 3-5: 系统开发视图

执行系统命令等操作封装成一系列静态工具类，方便其他模块调用；Configs 包中定义了各种配置类，如 RabbitMqConfiguration 类配置了系统监听的消息队列等信息；VO 包提供一系列 VO 对象，用于服务端与前端进行交互。通过上述对开发视图的设计，系统各层代码符合高内聚低耦合的要求，提高了代码的可维护性和扩展性。

（3） 进程视图

进程视图基于软件在操作系统上的运行特性，说明软件在处理业务逻辑时进程和线程的通信和调度等情况。本系统的进程视图如 3-6所示。

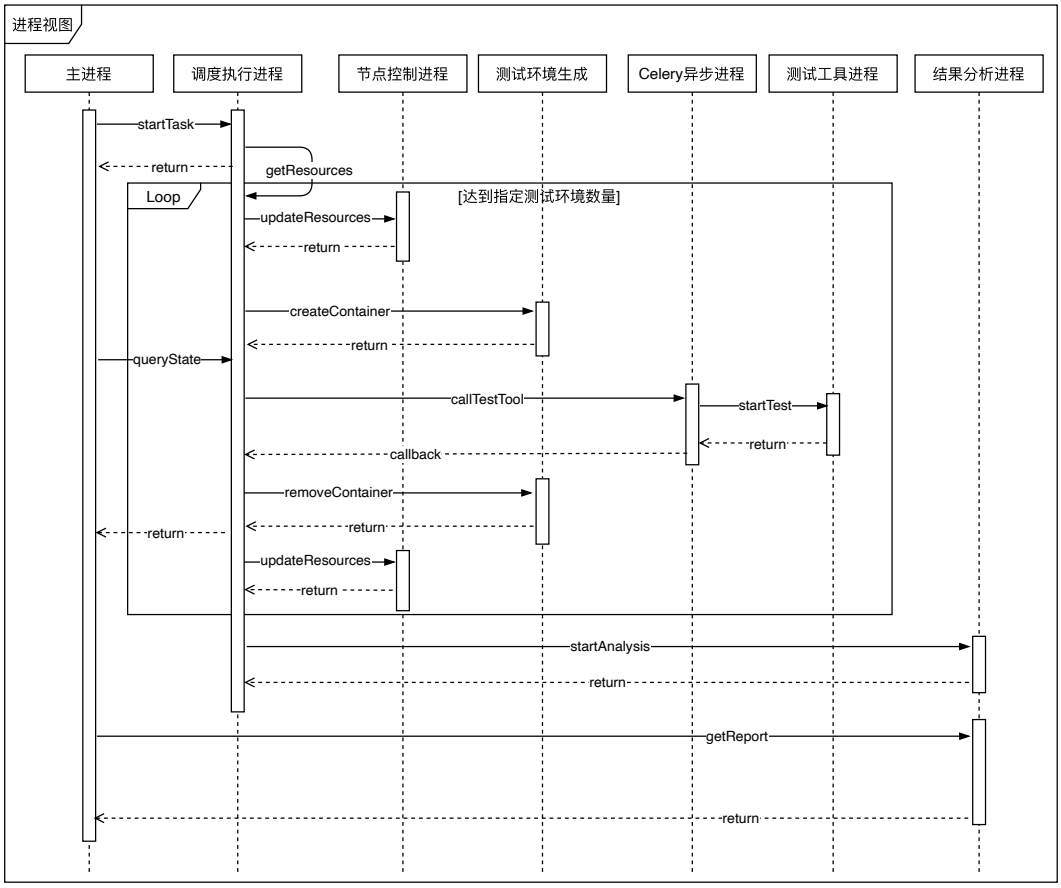

图 3-6: 系统进程视图

在本系统中，当用户开始执行一个 Java Web 应用可移植性测试任务时，主进程接收到请求后，将任务信息交给调度执行进程处理；调度执行进程获取到部署资源后，通知节点控制进程更新节点资源状态，而后测试环境生成进程启动一个线程用于创建 Docker 容器，容器与应用成功运行后，调度执行进程通过 Celery 异步进程调用自动化测试工具，并通过 callback 回调返回测试结果；由于一个可移植性测试任务包含多个测试环境的子任务，因此上述过程需要循环执行完成所有子任务。当所有子任务完成后，结果自动化分析进程启动对待测应用的分析；分析结束后，主进程可以发出请求获取测试报告。

（4） 物理视图

系统等物理视图是基于系统实施与运维人员的视角，从系统部署的角度来描述软件架构到硬件设备之间的映射关系，以及对各个物理服务器节点的服务

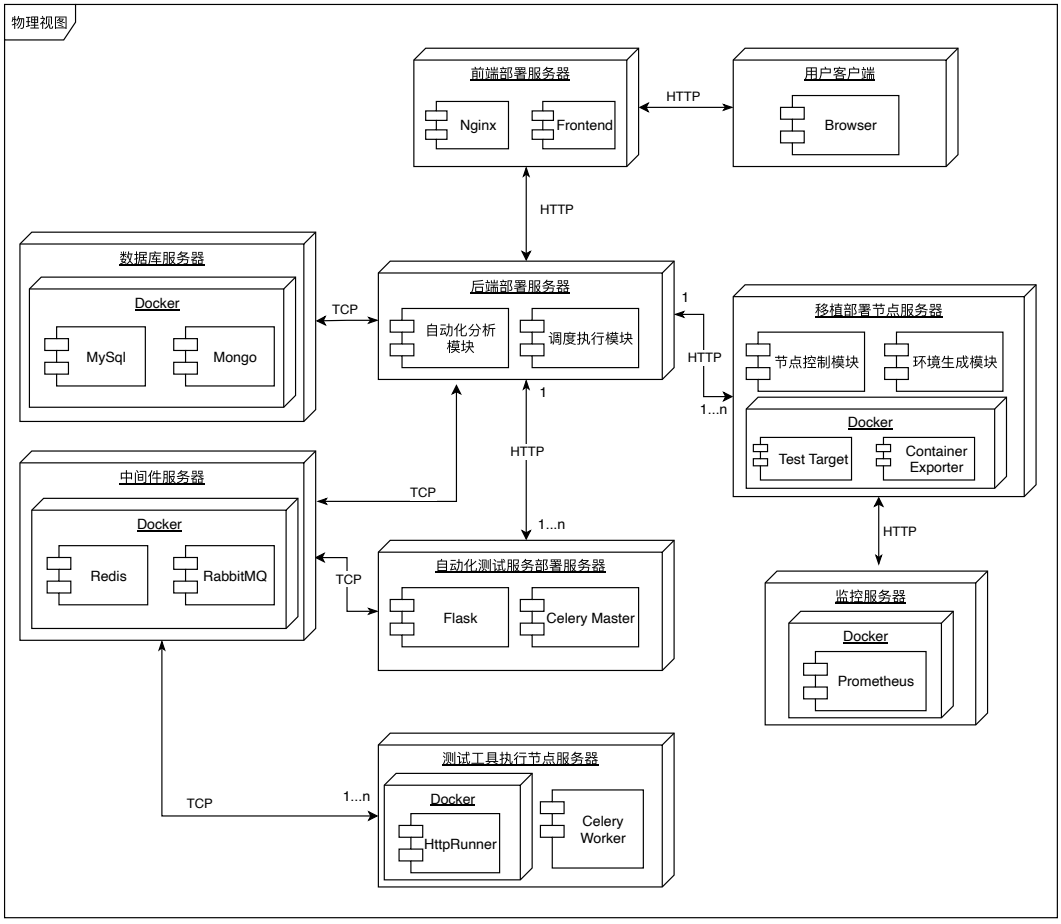

图 3-7: 系统物理视图

部署和通信机制进行说明。本系统的物理视图如图 3-7所示。

用户的个人电脑等设备作为客户端，使用浏览器访问 Java Web 应用可移植性测试平台的前端页面。前端部署在 Nginx 服务器上，浏览器发出的所有请求均由 Nginx 进行转发到后端服务器。系统后端功能服务直接运行于服务器上，各个微服务之间均通过 HTTP 请求相互调用进行通信，异步通信通过 RabbitMQ 实现。

本系统使用到的数据库、中间件、监控组件均采用 Docker 部署，能够减少安装配置过程，快速启动。

节点控制模块和环境生成模块以微服务形式部署于服务器集群每个节点上，该集群用于移植部署待测应用，每个节点上还部署了 Container Exporter 组件，用于监控节点上运行的 Docker 容器。自动化测试服务分为两部分部署，由 Python 编写的 Flask 应用和 Celery 运行于一台服务器上，对外暴露接口提供服务；另外 Celery Worker 采用集群部署，用于执行测试工具。

3.3.3 可移植性自动化测试任务流程设计

图 3-8表示 Java Web 应用可移植性自动化测试的流程设计，描述了一次测试任务的完整过程。宏观上可以将整个流程分为任务发布分解、执行移植部署、自动化测试、自动化分析四个大步骤。

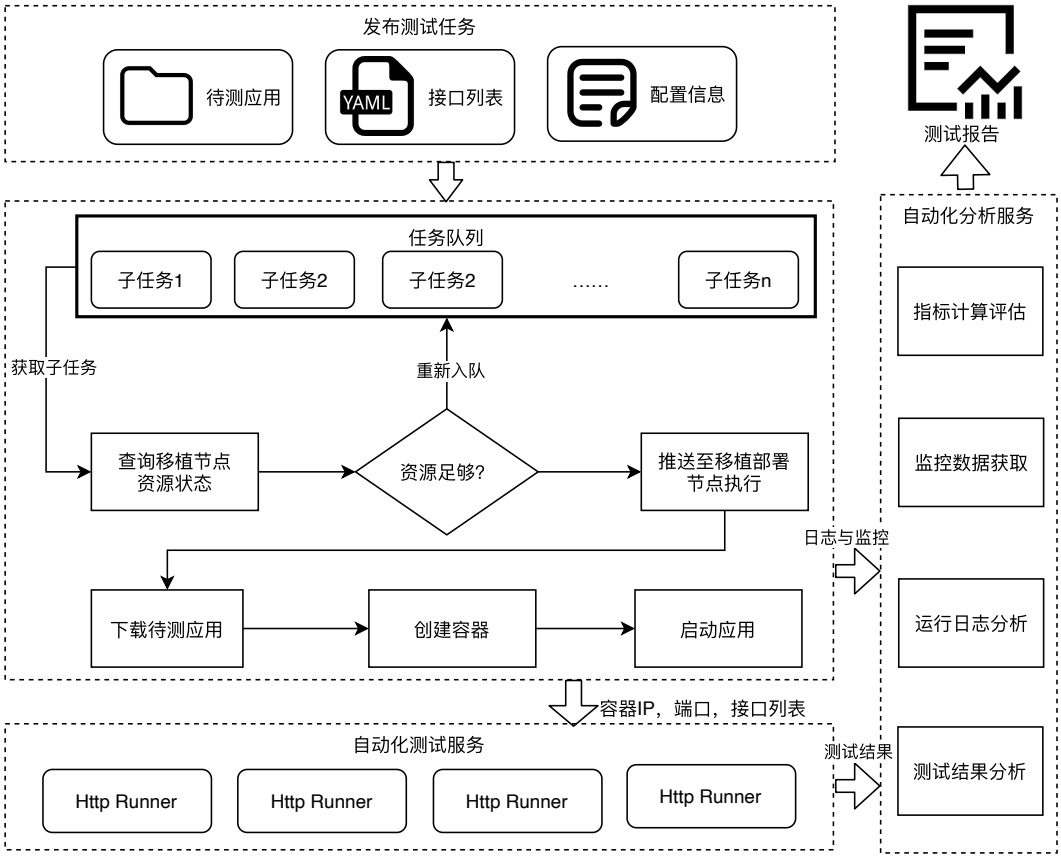

图 3-8: 可移植性自动化测试任务流程图

用户发布的可移植性测试任务包含待测 Java Web 应用，应用的接口列表和环境配置等信息。由于一次测试任务中可以包含一个或多个测试环境配置，因此首先需要对测试任务进行拆解，根据环境配置的不同组合拆分为多个子任务，并将子任务存入队列进行排队执行。然后从队列中取出当前要执行的子任务，根据其对应的环境配置，查询是否有移植部署节点的资源状态能否满足，如果当前资源不足，则将该子任务重新入队，若满足则将该子任务交由对应节点进行移植部署。移植节点该子任务后，需要下载测试目标的安装包，然后创建符合配置要求的 Docker 容器，并将安装包挂载到容器中执行，同时收集应用的运行日志和容器监控数据。

应用运行成功后，将容器的 IP、端口、接口列表等信息作为参数交给自动化测试服务进行测试，自动化测试服务启动 HttpRuner，对运行中的应用进行功能、性能测试。当一个测试任务的所有子任务均运行并测试完成后，自动化分析服务对应用的日志、监控数据、测试工具输出结果进行综合分析，输出测试报告。至此，一次可移植性测试任务运行结束。

3.4 持久化模型设计

根据系统需求和功能设计，系统需要对 Java Web 应用可移植性自动化测试过程中的数据进行持久化存储。本节对数据库持久化存储模型进行设计，系统使用 MySQL 存储系统业务数据，如测试目标、任务信息、子任务信息、测试环境配置信息、异常经验数据等；另外，系统使用 MongoDB 存储测试报告结果数据，利用其文档结构的特性，能够方便快速地对数据量较大的测试报告结果进行存储和查询。

3.4.1 系统业务数据库设计

系统业务数据主要由关系型数据构成，存储在 MySQL 中；通过从表字段、类型、含义和备注说明来对系统核心业务数据表就行阐述。

Target 表用于存放测试目标信息记录，包括名称、安装包路径、下载链接等。其具体属性详细设计如表 3-10所示。

表 3-10: 测试目标-Target 对象属性表

字段	类型	含义	备注
id	bigint(20)	测试目标 ID	全局唯一的测试目标 ID 标识
name	varchar(255)	名称	可理解的测试目标名称
path	varchar(255)	存储路径	测试目标安装包的存储路径
url	varchar(255)	下载链接	测试目标安装包的下载链接
create_time	datetime	创建时间	该条记录的创建时间
update_time	datetime	更新时间	该条记录的更新时间
is_deleted	tinyint(1)	是否删除	逻辑上标识该记录的删除

TestEnvConfig 表用于存放可移植性自动化测试系统支持的测试环境配置选项，包括操作系统类型、计算资源、软件依赖等。其具体属性详细设计如表 3-11所示。

表 3-11: 测试环境配置-TestEnvConfig 对象属性表

字段	类型	含义	备注
id	bigint	配置 ID	全局唯一的环境配置 ID 标识
name	varchar(20)	配置名称	该配置展示给用户的名称
value	varchar(255)	配置值	该配置对应系统能理解的实际值
type	varchar(20)	配置类型	OPERATION_SYSTEM，RESOURCE，SOFTWARE_DEPENDENCY

ExceptionInfo 表用于构建 Java Web 应用异常经验库，其中存储的异常经验数据为自动化分析服务提供支持。其具体属性详细设计如表 3-12所示。

表 3-12: 异常经验库-ExceptionInfo 对象属性表

字段	类型	含义	备注
id	bigint	异常经验 ID	全局唯一的异常经验 ID 标识
exname	varchar(20)	异常名称	Java Web 应用抛出的异常名称
extype	varchar(255)	异常类型	从可移植性角度判定该异常的类型
recommend	varchar(255)	异常解释	对异常进行解释，说明其可能原因

Task 表保存可移植性测试任务的基本信息和在系统中的状态，是系统的核心数据对象。其具体属性详细设计如表 3-13所示。

表 3-13: 测试任务-Task 对象属性表

字段	类型	含义	备注
id	bigint(20)	测试任务 ID	全局唯一的测试任务 ID 标识
name	varchar(255)	名称	可理解的测试任务名称
target_id	bigint(20)	测试目标 ID	测试任务关联的测试目标
yaml_url	varchar(255)	YAML 下载链接	HttpRunner 执行所需 YAML 格式接口列表
status	varchar(20)	任务状态	WAITING，RUNNING，ERROR，TERMINATE
create_time	datetime	创建时间	该条记录的创建时间
update_time	datetime	更新时间	该条记录的更新时间
is_deleted	tinyint(1)	是否删除	逻辑上标识该记录的删除

SubTask 对象是由测试任务拆解而来，包含各个测试环境中任务的配置和状态信息，这些数据存储在 SubTask 表中，其具体属性详细设计如表 3-14所示。

DeployNode 表是对移植部署集群中服务器节点的抽象，存放节点服务器的

表 3-14: 子任务-SubTask 对象属性表

字段	类型	含义	备注
id	bigint	子任务 ID	全局唯一的子任务 ID 标识
task_id	bigint(20)	任务 ID	子任务所属的测试任务
status	varchar(20)	子任务状态	WAITING，RUNNING，ERROR，TERMINATE
container_id	varchar(255)	容器 ID	子任务对应的 Docker 容器 ID
cpu_config	int(5)	CPU 配置	子任务运行配置的 CPU 核心数
mem_config	int(5)	内存配置	子任务运行配置的内存大小，单位 GB
os_config_id	bigint(20)	操作系统配置	对应 Config 表中配置
image_id	varchar(20)	容器镜像	子任务运行所需的 Docker 镜像
terminate_time	timestamp	运行结束时间	子任务运行结束的时间点
is_deleted	tinyint(1)	是否删除	逻辑上标识该记录的删除
begin_time	timestamp	开始运行时间	子任务开始执行的时间点

资源与状态信息，同时硬件国产情况，方便为信创适配性测试提供扩展支持。其具体属性设计如表 3-15所示。

表 3-15: 部署节点-DeployNode 对象属性表

字段	类型	含义	备注
id	bigint	节点 ID	全局唯一的节点 ID 标识
ip	varchar(20)	IP 地址	部署节点所在服务器的 IP 地址
status	tinyint(1)	节点状态	0：off，1：online
cpu	int(5)	CPU 核心数	注册的 CPU 资源信息
mem	int(5)	内存大小	注册的内存资源信息，单位 GB
available_cpu	int(5)	可用 CPU 核心数	当前空闲的 CPU 资源
available_mem	int(5)	可用内存大小	当前空闲的内存资源，单位 GB
os	varchar(255)	操作系统	该节点服务器的操作系统版本
is_domestic	tinyint(1)	是否国产	该服务器节点是否为国产硬件

3.4.2 测试报告实体设计

可移植性测试报告本质为上为存放大量任务执行信息与结果的文档，由于系统可能会扩展自动化测试工具或增加新的监控手段，因此文档结构并非一成不变的，所以选用 MongoDB 存储测试报告数据。

本系统根据功能逻辑和数据结构，定义了 TaskResult 集合和 RunDetails 集合。

TaskResult 集合对应可移植性测试任务，存放测试结果的各项指标、开始结束时间，以及其关联的子任务列表。其具体属性设计如表 3-16所示，图 3-9表示了 TaskResult 集合在 MongoDB 中实际存储的数据结构。

表 3-16: 测试任务结果-TaskResult 对象集合

字段	类型	含义	备注
taskId	Long	测试任务 ID	绑定对应的可移植性测试任务
startTime	Timestamp	开始时间	任务开始的时间戳
endTime	Timestamp	结束时间	任务结束的时间戳
functionScore	Integer	功能得分	测试目标可移植性的功能表现
performanceScore	Integer	性能得分	测试目标可移植性的性能表现
efficiencyScore	Integer	移植效率得分	测试目标可移植性的移植效率
adaptabilityScore	Integer	适应性得分	测试目标可移植性的适应性表现
subTasks	List<Long>	子任务列表	子任务运行结果记录列表

```
{
    "_id" : ObjectId("..."),
    "taskId" : 25,
    "startTime" : "1607312168000",
    "endTime" :"1607311808000",
    "functionScore" : 98,
    "performanceScore" : 85,
    "efficiencyScore": 92,
    "adaptabilityScore": 90,
    "subTasks" : [
        {
            "subTaskId": 50, //业务系统中的子任务 ID
            "objectId":   ObjectId("...") //MongoDB 中生成的对象 ID
        },
        ...
    ]
}
```

图 3-9: 测试任务结果-TaskResult 对象集合存储结构

RunDetails 结合对应可移植性测试任务拆解出的子任务，存放待测应用在各个测试环境中运行和测试的详细情况，包括运行日志、异常信息、监控时序数据，以及 HttpRunner 测试结果等。其具体属性设计如表 3-17所示。图 3-10表示了 RunDetails 集合在 MongoDB 中实际存储的数据结构。

```
{
    "_id": ObjectId("..."),
    "subTaskId": 10,
    "startTime": "1607312168000",
    "endTime": "1607311808000",
    "log": "...",
    "exceptions": [
        {
            "type": "网络连接异常",
            "error": "java.lang.IllegalArgumentException:",
            "detail": " Cannot Connect to public network!",
            "explain": "无法连接到公共网络！",
            "recommend": "检查代码中网络访问相关配置"
        },...],
    "memUsage": [[1615041806.026,"1305612288"],...],
    "cpuUsage": [[1615041806.026,"1305612288"],...],
    "toolResult": {{
            "success":true,
            "time":{"start_at":1613526102.2586281,"duration":0.4994847774505615},
            "platform":{...},
            "details":[{
                    "success":true,
                    "stat":{...},
                    "records":[{
                            "name":"/api/test/login",
                            "status":"success",
                            "attachment":"",
                            "meta_datas":{
                                "name":"/api/test/login",
                                "data":[
                                    {"request":{...},
                                     "response":{...}
                                    }],
                                "stat":{...},
                                "validators":{..}
                            },
                            "meta_datas_expanded":[{..}],
                            ...},...
]}]}}}
```


图 3-10: 子任务运行结果-RunDetails 对象集合存储结构

表 3-17: 子任务结果-RunDetails 对象集合

字段	类型	含义	备注
subTaskId	Long	子任务 ID	绑定对应的子任务
startTime	Timestamp	开始时间	子任务开始的时间戳
endTime	Timestamp	结束时间	子任务结束的时间戳
log	String	运行日志	子任务 Docker 容器输出的日志
exceptions	List<ExceptionInfo>	异常信息列表	日志分析结果
memUsage	List<Data>	内存占用量	内存监控时序数据
cpuUsage	List<Data>	CPU 使用率	CPU 使用时序数据
toolResult	Object	测试工具结果	HttpRunner 输出的结果数据

3.5 本章小结

本章对系统的业务需求进行了分析，并完成对系统整体架构的概要设计。首先对系统的业务应用场景进行概述，并说明业务过程；其次对本系统的涉众情况进行分析，简要梳理出系统的功能性和非功能性两方面的需求；然后结合用例图和文字描述对系统需求进一步说明，明确了系统的用户使用场景；在需求分析的基础上，对系统的整体架构进行设计，基于 4+1 视图对系统体系结构进行详细介绍；最后对可移植性自动化测试的流程方案和系统持久化模型进行设计，为下一章详细设计与代码实现奠定基础。

第四章　系统详细设计与实现

4.1　测试环境生成模块详细设计与实现

4.1.1　测试环境生成模块架构设计

图 4-1表示本系统测试环境生成模块的架构设计。该模块底层运行环境主要依赖于开源的 Docker 容器技术，运行 Java Web 应用的测试环境由 Docker 容器构成，Container Exporter 组件会监控这些容器的运行状态，将 CPU、内存等使用信息上传到 Prometheus 的时序数据库中。

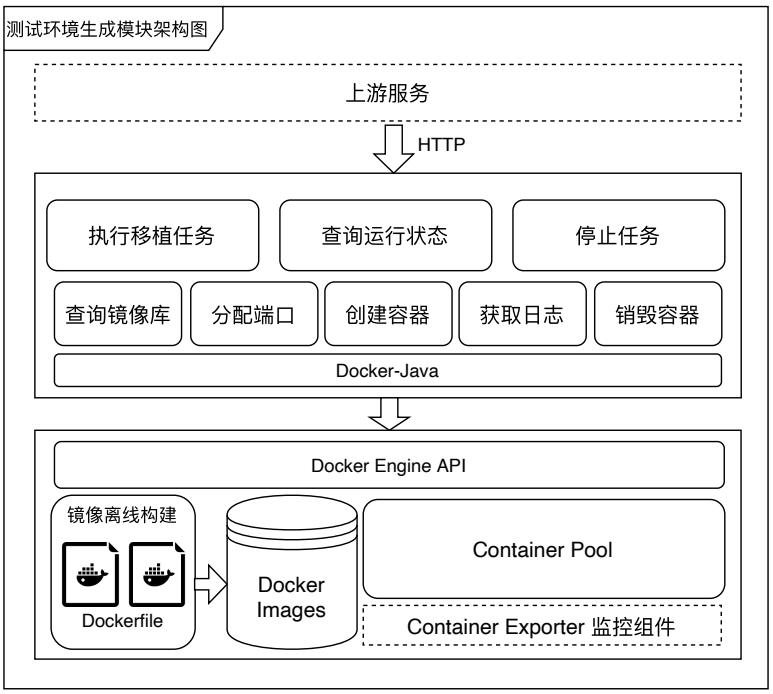

图 4-1: 测试环境生成模块架构图

系统支持的测试环境软件相关配置，如操作系统、Java 版本等，都通过离线构建 Docker 镜像实现，通过编写 Dockerfile 能够扩展系统对测试环境的支持。Docker-Java SDK 对 Docker Engine API 进行了良好的抽象封装，本模块通过使用 Docker-Java 控制 Docker 容器，从而实现分配容器端口、创建 Docker 容

器、获取容器运行日志和销毁容器等操作。由于环境生成模块运行在移植部署集群的各个节点上，因此对外部提供 HTTP 接口，实现执行测试子任务、停止任务等功能。

4.1.2 测试环境生成模块核心类图设计

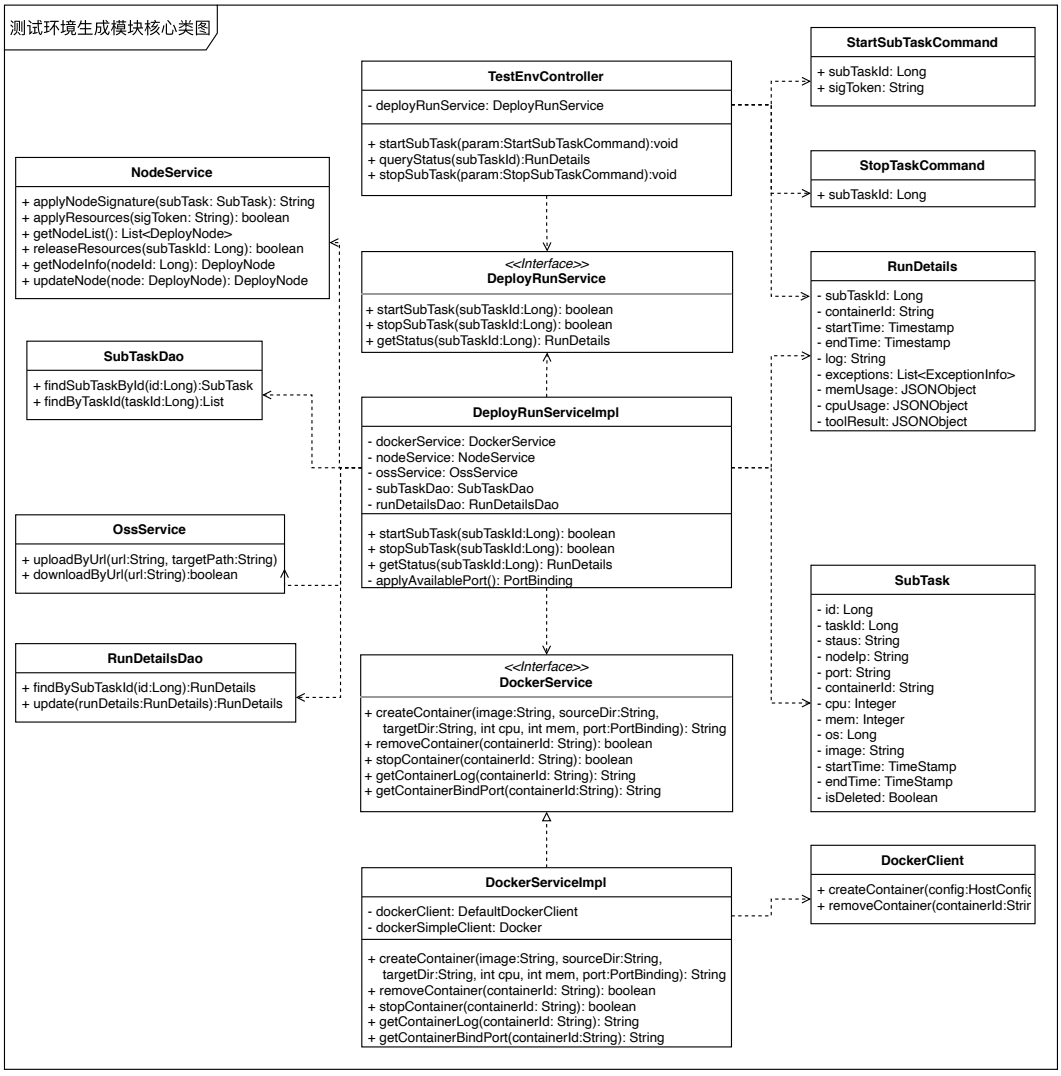

图 4-2: 测试环境生成模块核心类图

测试环境生成模块的核心类图设计如图 4-2所示。TestEnvController 是对外提供 RESTful 接口和路由的控制类，其接收封装成 Command 类的参数，对请求进行校验和转发，并调用 DeployRunService 类封装的抽象接口进行处理。

DeployRunServiceImpl 类是 DeployRunService 的实现类，其负责控制子任务执行和测试环境生成的整个流程；它的方法主要接收 subTaskId 参数，通过

调用 SubTaskDao 来获取和更新 SubTask 的配置信息和状态，通过 NodeService 和 OssService 来更新节点资源，下载测试目标安装包；通过调用 DockerService 控制容器的创建、销毁等流程，当子任务结束后将运行数据通过 RunDetailsDao 更新到 MongoDB 中。DockerService 和 DockerServiceImpl 是本模块最重要的核心类，它们分别定义和实现创建容器、停止容器、销毁容器、获取日志等方法，测试环境的生成主要依赖于其提供的 createContainer() 方法。

4.1.3 测试环境生成模块顺序图

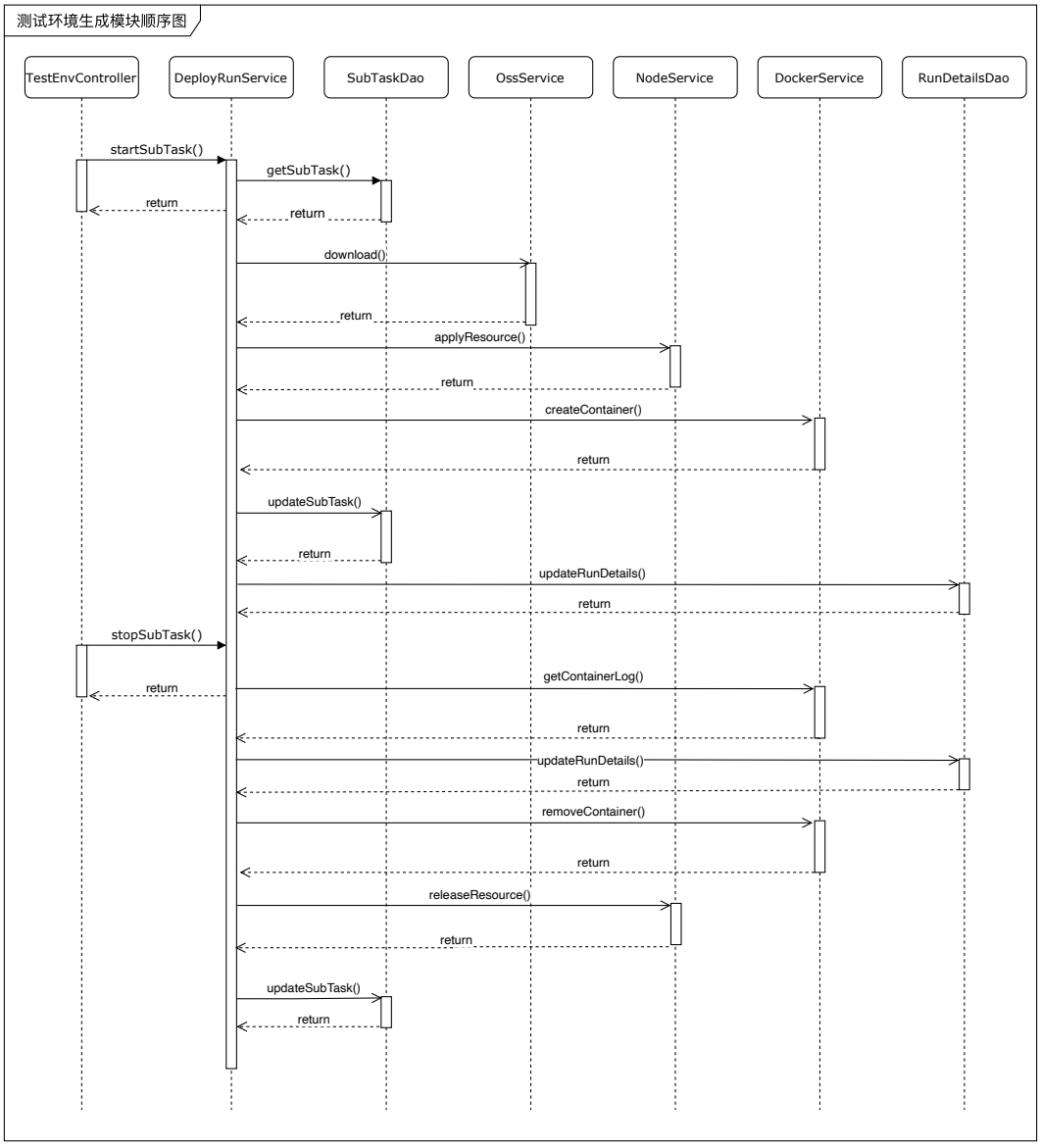

图 4-3: 测试环境生成模块顺序图

测试环境生成模块的顺序图如图 4-3所示。当接收开始测试子任务开始的请求时，TestEnvController 会调用 DeployRunService 的 startSubTask() 方法执行子任务运行的流程。在生成环境之前，先调用 SubTaskDao 获取子任务的配置信息，使用 OssService 下载测试目标安装包；然后通过调用 NodeService 的 applyResouce() 方法更新当前节点资源状态，之后调用 DockerService 的 createContainer() 方法创建容器生成测试环境，并通过 SubTaskDao 更新子任务的状态，并将容器的运行信息更新到 MongoDB 的 RunDetails 集合。当子任务测试完成后，DeployService 执行 stopSubTask() 方法执行任务停止流程，然后调用 DockerService 的 getContainerLog() 方法获取完整运行日志，并将日志信息更新到 MongoDB 中，销毁容器后释放资源，最后更新子任务的运行状态。

4.1.4 测试环境生成模块关键代码

图 4-4是环境生成模块执行测试子任务方法的主要代码，其接收参数为子任务标识 subTaskId，测试目标安装包下载链接 targetUrl 和应用运行命令 runCmd。

```
@Async
@Override
public void startSubTask(Long subTaskId, String targetUrl, String runCmd, String sig) {
    Optional<SubTask> subTaskOptional = subTaskDao.findById(subTaskId);
    if (!subTaskOptional.isPresent() || subTaskOptional.get().isDeleted()) {
        throw new NotFoundException(String.format("该子任务不存在, ID: %s",
subTaskId));
    }
    //下载目标，分配资源
    SubTask subTask = subTaskOptional.get();
    String path = ossService.downloadByUrl(targetUrl);
    if (!nodeService.applyResources(sig) {
        throw new BadRequestException("无效的资源申请签名");
    }
    //新建线程执行
    subTaskThreadPool.getExecutor().execute(
            new DeploySubTaskExecutor(dockerService, subTask, path,
                    rumCmd, configuration));
    subTask.setStatus(SubTaskStatus.RUNING);
    //更新状态
    subTaskDao.save(subTask);
}
```

图 4-4: 开始执行子任务代码

```
public class DockerServiceImpl implements DockerService {

    private final Long CPU_COUNT = 1000000000L; //Docker 中一个 CPU 核心数换算
    private final Long MEM_GB = 1024*1024*1024L; //Docker 中 1GB 内存换算
    private DefaultDockerClient dockerClient;
    private Docker dockerSimpleClient;

    @PostConstruct
    private void initDockerClient() {
        dockerClient = new DefaultDockerClient("unix:///var/run/docker.sock");
        dockerSimpleClient = new UnixDocker(new File("/var/run/docker.sock"));
    }

    @PreDestroy
    private void closeDockerClient() {dockerClient.close();}

    @Override
    public String createContainer(String imageName, String sourceDir, String targetDir,
                                  int cpu, int mem, PortBinding portBinding) {
        Map<String, List<PortBinding>> portBindings = new HashMap<>();
        List<PortBinding> hostPorts = new ArrayList<>();
        hostPorts.add(portBinding);
        portBindings.put("8080/tcp", hostPorts); //绑定端口
        HostConfig hostConfig = HostConfig.builder() //生成 Docker 运行配置
                .appendBinds(HostConfig.Bind.from(sourceDir)
                .to(targetDir).readOnly(false).build())
                .portBindings(portBindings).nanoCpus(CPU_COUNT*cpu)
                .memory(MEM_GB*mem).build();

        ContainerConfig containerConfig = ContainerConfig.builder()
                .hostConfig(hostConfig).image(imageName).build();
        ContainerCreation container = null;
        try {
            container = dockerClient.createContainer(containerConfig);
            dockerClient.startContainer(container.id());
            return container.id();
        } catch (Exception e) {
            e.printStackTrace();
            return container == null ? null : container.id();
        }
    }
    //省略其他代码
}
```

图 4-5: 测试环境生成模块关键代码

根据 subTaskId 使用 SubTaskDao 查询出子任务的具体信息，如果不存在则抛出异常；targetUrl 作为参数调用 OssService 进行下载，返回值为本地存储路径，用于容器运行的挂载目录参数；根据子任务配置所需要的 CPU 核心数和内存，调用 NodeService 更新节点资源状态，如果资源不足则抛出异常，然后通过新建一个线程并调用 DockerService 来执行创建容器、更新运行数据等流程；最后将子任务的状态更新为运行中。

创建测试环境 Docker 容器的代码如图 4-5所示。createContainer() 方法首先使用 HostConfig 的 build() 方法来构造容器运行配置 ContainerConfig 对象，用于绑定容器运行运行、挂载文件目录、限制容器运行的 CPU 和内存资源。利用 DockerClient 提供的接口创建并运行容器，实现测试环境的生成。

4.2 节点控制模块详细设计与实现

4.2.1 节点控制模块架构设计

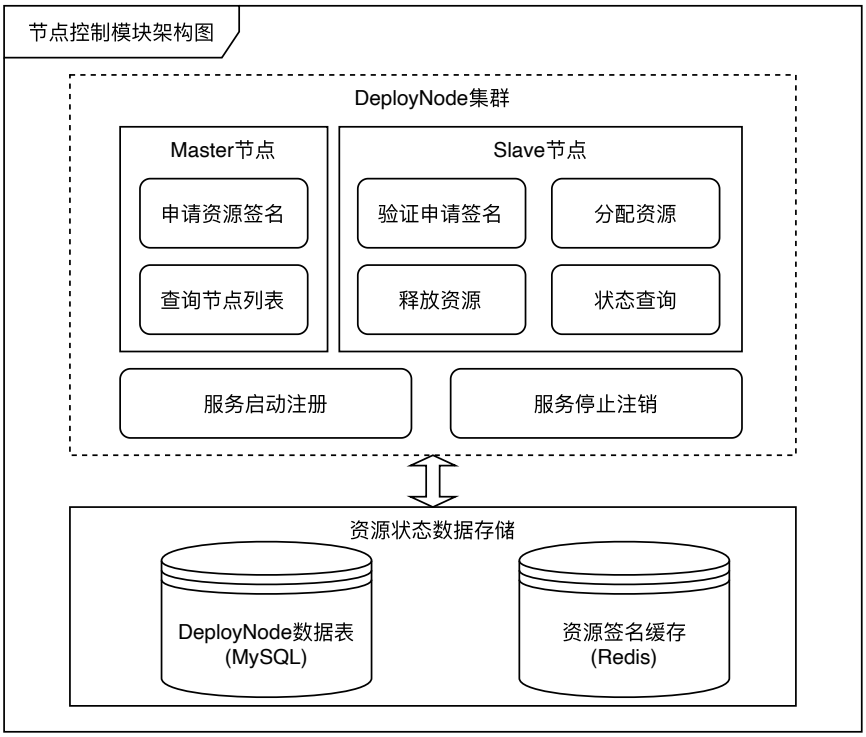

图 4-6: 节点控制模块架构图

图 4-6表示节点控制模块的架构设计。该模块运行在移植部署集群的各个节点上，集群至少有一个节点，并作为 Master 节点负责集群管理，提供申请资

源签名、查询节点列表等接口。每个节点上的服务启动时，会自动将 IP 地址、CPU、内存等资源信息注册到数据库中，当服务停止时会自动将该节点的资源状态注销。

当执行调度模块对具体子任务进行调度时，需要先向 Master 节点申请资源签名，资源签名的具体信息存在 Redis 缓存数据库中，并更新 DeployNode 表中对应 Node 资源状态。资源签名缓存过期时间为 30 秒，Master 节点监听过期事件，若过期时未使用该签名，则自动释放资源。各个 Slave 节点在执行子任务时，会对测试环境生成模块传入的资源签名进行验证，分配资源。另外还提供释放资源、状态查询等功能接口。

4.2.2 节点控制模块核心类图设计

节点控制模块核心类图设计如图 4-7所示。

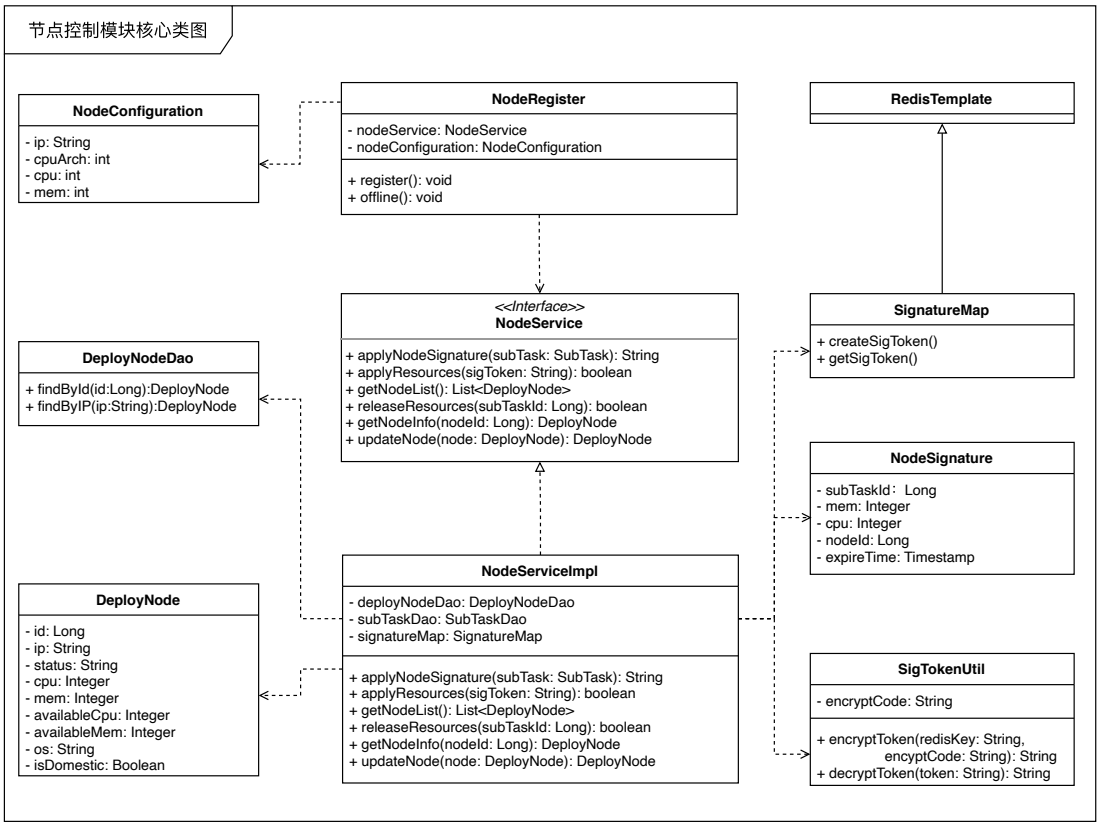

图 4-7: 节点控制模块核心类图

NodeService 接口类是对节点控制模块核心功能逻辑的抽象和封装，NodeServiceImpl 是其实现类，对外提供了一系列方法接口来实现移植部署集群节点

的状态维护和资源管理。

调度执行模块通过调用 applyNodeSignature() 方法来获取资源申请签名，测试环境生成模块通过调用 applyResources() 方法，并传入资源签名来验证并获取资源。另外，NodeServiceImpl 还提供了获取节点列表、释放资源和更新节点信息等功能。

NodeSignature 类是对资源申请签名的数据封装，其包含了子任务 ID、节点 ID，申请的资源大小和签名过期时间；SignatureMap 是根据业务功能对 RedisTemplate 的包装类，用于将资源申请签名作为缓存存入 Redis 数据库中；SigTokenUtil 类用于对 Redis 缓存的 Key 进行加密和解密，防止伪造资源签名信息，并保证数据传输的安全性；NodeRegister 类是用来于实现节点自动注册和注销的功能，在服务启动和停止的时候，根据本节点的配置信息，调用 NodeService 更新数据库表中 DeployNode 的状态。

4.2.3 节点控制模块顺序图

节点控制模块的顺序图如图 4-8所示。当移植部署集群中的节点启动时，会自动调用 NodeRegister 的 register() 方法，通过 NodeServcie 的 updateNode() 方法将本节点的资源数据注册到数据库中，并且将节点的状态设置为“在线”。

在可移植性自动化测试任务执行过程中，会先调用 NodeService 的 applyNodeSignature() 来获取可用资源，NodeService 会查询并选举出一个合适的节点，分配任务所需资源，更新节点的可用资源数量，并将分配的资源信息构造成签名，调用 SignatureMap 的 saveSignature() 方法存入 Redis 中；然后调用 SigTokenUtil 的 encryptToken() 方法对签名缓存的 Key 进行加密生成 Token，返回给外部服务。当测试子任务进行测试环境生成时，会调用 NodeService 的 applyResource() 方法，传入资源签名的 Token，NodeService 会调用 SigTokenUtil 的 decryptToken() 进行解密，获取签名缓存 Key，从 Redis 中获取资源申请签名信息，验证子任务和节点信息。测试子任务结束后会调用 NodeService 的 releaseResource() 方法将使用的资源释放掉，更新节点的可用资源数量。

移植部署集群中的节点停机或停止服务时，NodeRegister 的 offline() 方法会在服务停止前自动执行，将数据库中该节点的状态设置为“离线”，该节点在系统中不会再提供服务。

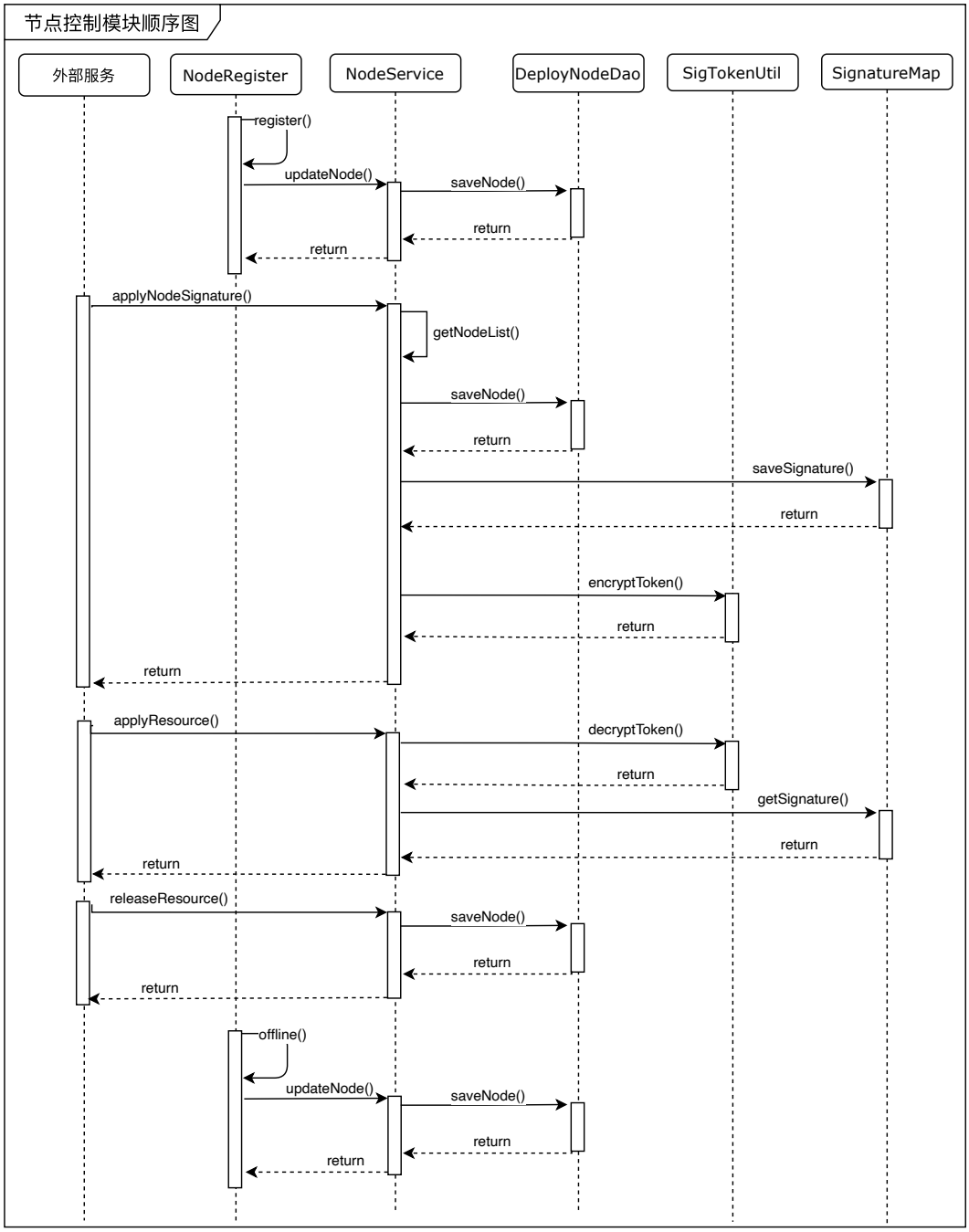

图 4-8: 节点控制模块顺序图

4.2.4 节点控制模块关键代码

图 4-9所示是节点自动注册与注销的代码。通过使用 @Configuration 注解，将 NodeRegister 注册为一个由 Spring 框架自动管理的 Bean，其在 Spring Boot 启动时会自动实例化为一个单例的对象；配合 @PostConstruct 和 @PreDe-

stroy 注解，可以实现在 NodeRegister 创建后和销毁前进行指定处理，本系统中通过定义 register() 方法，在服务启动后在数据库中添加或更新节点资源和状态信息；offline() 在服务停止前运行，负责将节点状态修改为“离线”。

```
@Configuration
public class NodeRegister {

    @Autowired
    private NodeConfiguration nodeConfiguration;
    @Autowired
    private NodeService nodeService;

    @PostConstruct
    public void register() {
        DeployNode local = new DeployNode();
        Optional<DeployNode> node = nodeService
                                    .findNodeByIp(nodeConfiguration.getIp());
        if (node.isPresent()) {
            local = node.get();
        }
        BeanUtils.copyProperties(nodeConfiguration, local);
        local.setStatus(NodeStatus.ONLINE);
        nodeService.update(local);
    }

    @PreDestroy
    public void offline() {
        Optional<DeployNode> local = nodeService
                                    .findNodeByIp(nodeConfiguration.getIp());
        local.setStatus(NodeStatus.OFFLINE);
        nodeService.update(local);
    }
}
```

图 4-9: 节点自动注册注销代码

图 4-10所示是申请节点资源签名方法的代码，其接收子任务 SubTask 作为参数，根据子任务所需资源从集群节点列表中筛选出符合要求的节点；由于可能存在多个节点的资源满足该子任务，因此根据选择策略选举出最合适的节点进行资源分配，更新数据库中该节点的可用资源；然后调用 SignatureMap 的 saveNodeSig() 方法，该方法会根据节点和子任务信息生成签名对象存入 Redis

缓存，并返回 Redis 中的 Key，最后调用静态工具类 SigTokenUtil 对 Key 进行加密生成 Token，返回给上游服务。

```
@Override
public String applyNodeSignature(SubTask subTask) {
    //筛选满足条件的节点
    List<DeployNode> nodeList = this.getNodeList().stream()
            .filter(node -> node.getAvailableMem() >= subTask.getMem()
                    && node.getAvailableCpu() >= subTask.getCpu())
            .collect(Collectors.toList());
    //所有节点资源均不满足，返回 NULL
    if (nodeList.size() < 1) {
        return null;
    }
    //根据选举策略从可选节点中选出最佳 Node
    DeployNode node = NodeChoiceStrategy.chooseNode(nodeList);
    //存入 Redis 缓存
    String redisKey = signatureMap.saveNodeSig(node, subTask);
    node.setAvailableMem(node.getAvailableMem()-subTask.getMem());
    node.setAvailableCpu(node.getAvailableCpu()-subTask.getCpu());
    this.updateNode(node)
    //对 Key 进行加密
    return SigTokenUtil.encryptToken(redisKey, Constants.ENCRYPT_CODE);
}
```

图 4-10: 申请资源签名代码

4.3 调度执行模块详细设计与实现

4.3.1 调度执行模块架构设计

图 4-11表示调度执行模块的架构设计，该模块主要负责对 Java Web 可移植性自动化测试任务的执行过程进行调度，通过与节点控制、测试环境生成、自动化测试等模块之间的交互，控制可移植性测试任务中移植部署调度执行、自动化测试调度执行和自动化分析调度执行等步骤，对外提供任务开始、停止、重新执行和状态查询等接口。

调度执行模块功能逻辑的实现主要基于队列机制，其中包含用于子任务等待排队的延时队列和用于自动化测试工具结果通知的消息队列。由于一个可移

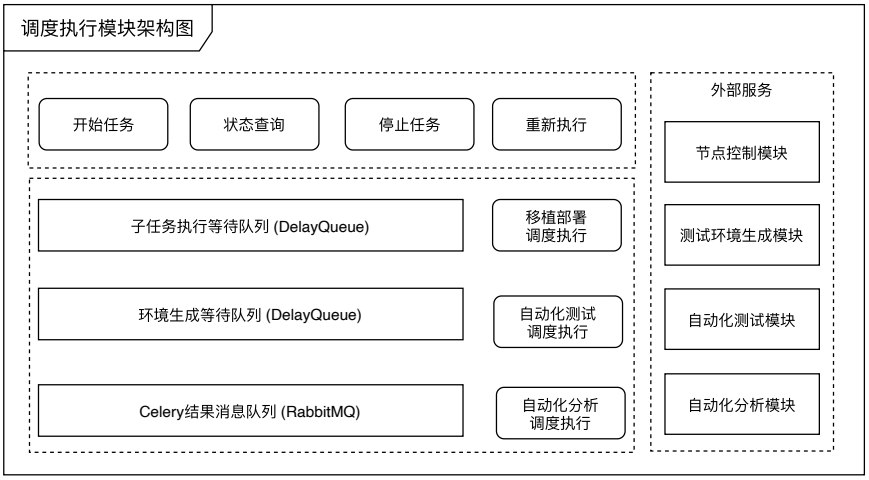

图 4-11: 调度执行模块架构图

植性测试任务包含多个子任务，因此执行可移植性测试任务时，调度执行模块会将其所有子任务封装成延时任务加入到基于延时队列的线程池中，由线程池负责调度子任务的执行，每个子任务执行时会先调用节点控制模块申请资源签名，如果申请失败，会将该子任务重新放回队列中等待调度执行；成功申请到资源并调用测试环境生成模块执行了部署运行的子任务，需要调用自动化测试模块对其进行测试，但由于需要 Java Web 应用运行启动需要一定时间，需要等待其启动完成再调用自动化测试工具，因此将子任务调用自动化测试模块的过程同样封装成延时任务加入延时队列等待调度执行。

自动化测试模块对每个子任务的测试完成后，会通过 RabbitMQ 消息队列发送测试完成的回调消息，调度执行模块通过监听相应队列，对回调消息进行处理，并判断可移植性测试任务的所有子任务是否执行完成，判断是否触发调用自动化分析模块。

4.3.2 调度执行模块核心类图设计

调度执行模块核心类图设计如图 4-12所示。TaskRunService 接口类是对调度执行模块主要功能逻辑的抽象，TaskRunServiceImpl 是其具体的实现类，对外提供了可移植性任务开始、停止、状态查询、获取子任务列表和维护子任务队列等接口。

TaskRunService 实现对可移植性测试任务运行的调度控制主要依赖于 ScheduledThreadPoolExecutor，它是一种基于延时队列 DelayQueue 实现的任务调度线程池，其支持指定等待时间并通过多线程的方式执行延时任务。本模

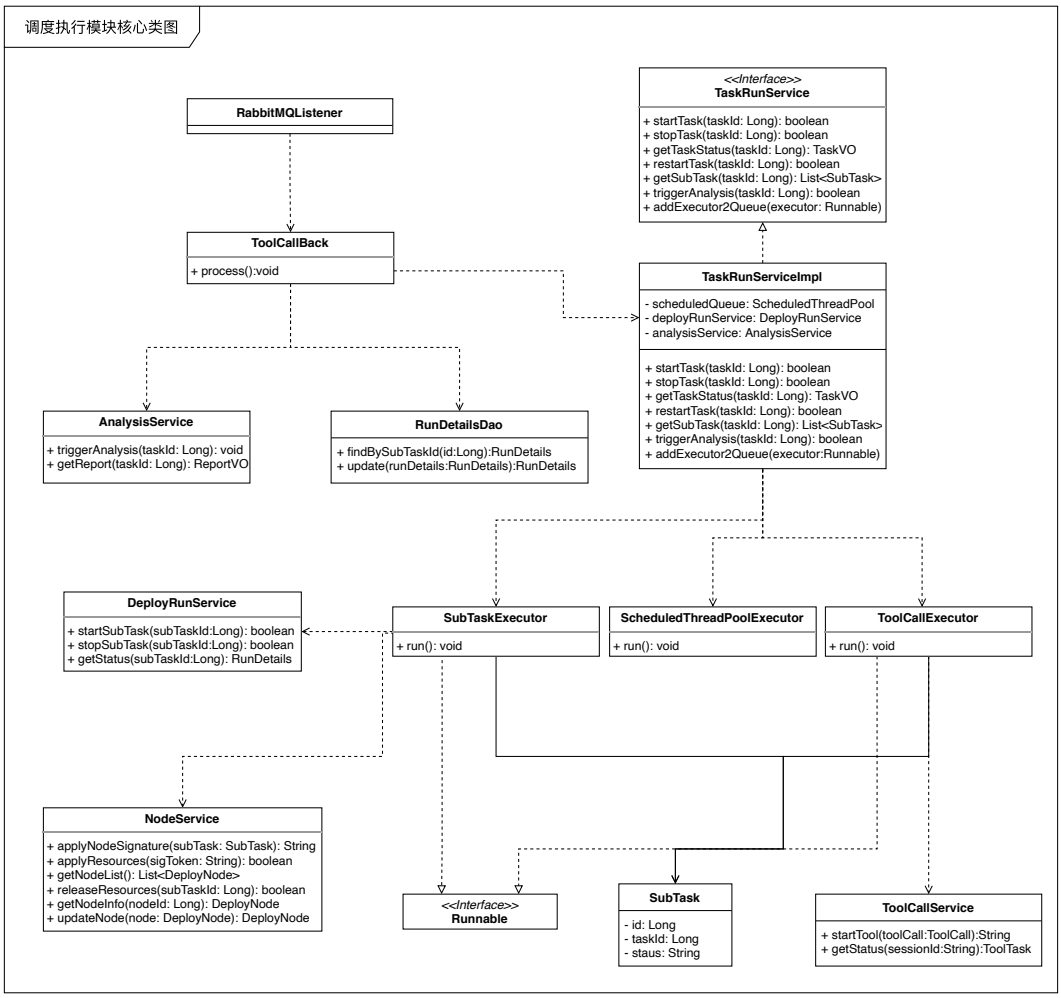

图 4-12: 调度执行模块核心类图

块将子任务调用测试环境生成模块和自动化测试模块的逻辑分别被封装成 SubTaskExecutor 和 ToolCallExecutor，这两个类均实现了 Runnable 接口，因此可以加入到 ScheduledThreadPoolExecutor 线程池中进行调度执行，通过设置 DelayTime 来指定延迟调度时间，可以实现资源不足时，子任务重新入队增加等待时长实现退火；并且可以延迟对自动化测试模块的调用，等待测试环境中待测应用启动完成。

SubTaskExecutor 类通过实现 run() 方法，负责子任务执行时调用节点控制模块和测试环境生成模块完成申请资源签名和触发子任务部署运行，并创建 ToolCallExecutor 对象加入到线程池中等待触发自动化测试模块。ToolCallExecutor 类负责在 SubTaskExecutor 对象调用测试环境生成模块后，等待测试环境中的 Java Web 应用启动，然后调用自动化测试模块接口，触发自动

化测试工具对子任务测试环境中的测试目标进行测试。

RabbitMQListener 类中配置了对 RabbitMQ 消息队列的监听，用于获取自动化测试模块发出的测试完成回调通知。ToolCallBack 负责对 RabbiMQListener 获取的通知进行具体处理，其会根据回调的结果，从 Redis 中取出测试结果存入 MongoDB 重的 RunDetails 中，并更新子任务状态，调用测试生模块停止子任务运行。当可移植性测试任务的所有子任务均测试结束后，则调用自动化分析模块进行结果分析。

4.3.3 调度执行模块顺序图

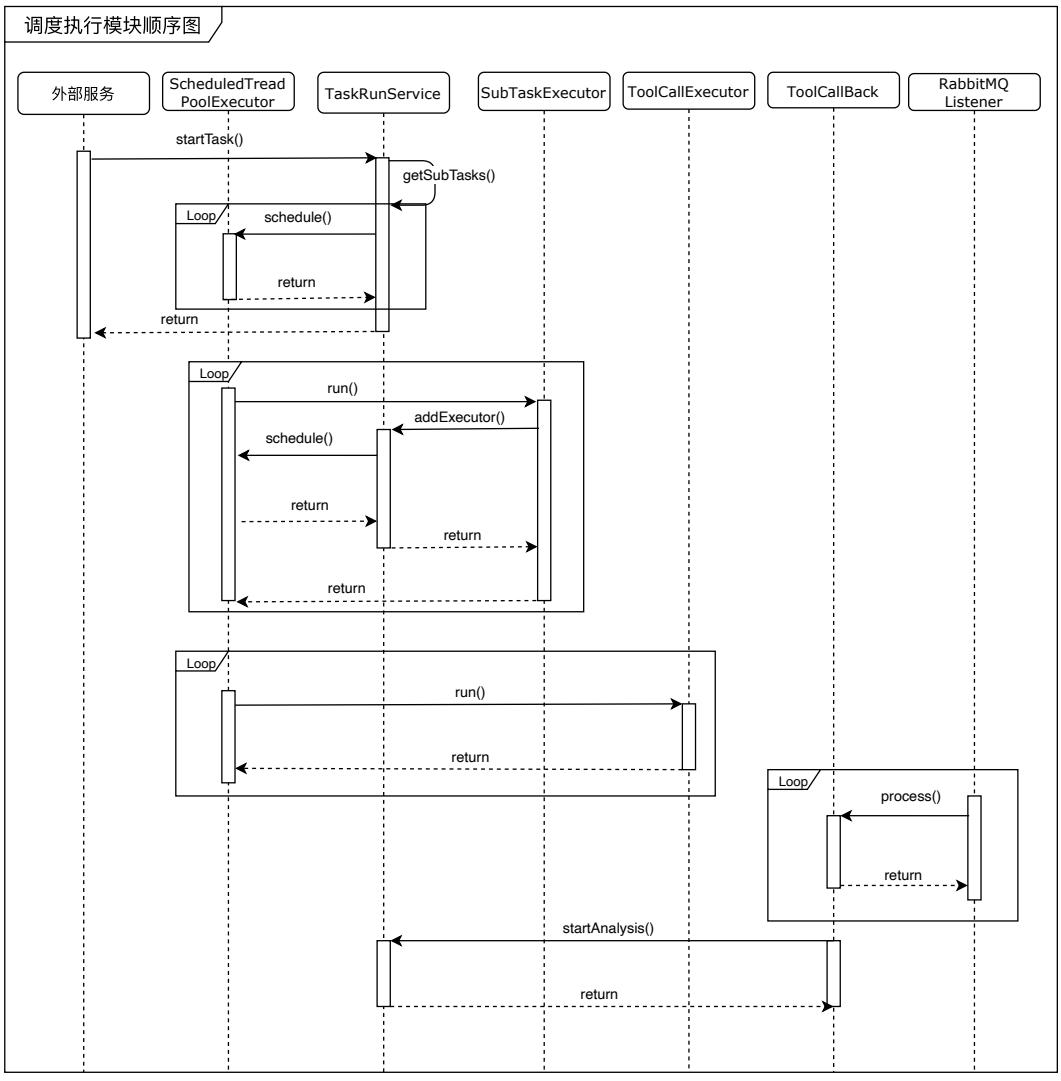

图 4-13: 调度执行模块顺序图

调度执行模块的顺序图如图 4-13所示。受限于篇幅，为了更好地展示核心业务逻辑，省略了与外部服务和其他模块交互的部分。

当上游或外部服务通过调用 TaskRunService 的 startTask() 方法启动执行一个可移植性自动化测试任务；TaskRunService 接收请求后对任务进行处理，首先获取该可移植性测试任务的子任务列表；然后循环遍历每个子任务，并将子任务封装成 SubTaskExecutor，调用 ScheduledThreadPoolExecutor 的 schedule() 方法将封装后的子任务加入到线程池中等待执行。

ScheduledThreadPoolExecutor 内置 DealyQueue 来维护等待执行的任务，当 SubTaskExecutor 达到延时时间，ScheduledThreadPoolExecutor 会分配线程来执行 SubTaskExecutor 中的 run() 方法，run() 方法中会先调用 NodeService 的 applyNodeSignature() 方法来获签名，然后使用封装好请求的 HttpUtil 工具类调用测试环境生成模块的接口触发 DeployService 执行 startSubTask() 方法，如果资源申请失败，则会将 SubTaskExecutor 中的 delayTime 加上 30 秒，重新加入到线程池中等待调度，增加 delayTime 是因为申请不到资源，增加等待时间从而减少任务密集性，达到任务调度退火的目的；如果运行成功，则会创建一个 ToolCallExecutor 对象，并调用 TaskRunService 的 addExecutor() 方法将 ToolCallExecutor 加入到线程池中，addExecutor() 方法会根据对象的 delayTime 来设置等待调度时间，ToolCallExecutor 的 delayTime 默认设置为 2 分钟，用于等待测试环境生成模块将待测应用启动成功；当 ToolCallExecutor 的 run() 方法被调度执行时，会调用自动化测试模块使用自动化测试工具 HttpRunner 对运行在测试环境中的待测应用进行测试。

自动化测试模块对每个子任务的测试完成后，都会通过 RabbitMQ 发送运行成功的通知消息。调度执行模块的 RabbitMQListener 监听到消息后，会调用 ToolCallBackd 的 proces() 方法进行处理，process() 方法会进行一系列数据处理和存储的操作，并判断当前可移植性测试任务的所有子任务是否都执行完成，若完成则调用 TaskRunServcie 的 startAnalysis() 方法触发对可移植性测试任务的结果分析。

4.3.4 调度执行模块关键代码

图 4-14所示是调度执行模块开始执行可移植性测试任务的部分相关代码。TaskRunServiceImpl 实现了 TaskRunService 接口，其维护一个 ScheduledThreadPoolExecutor 调度线程池，与节点控制模块自动注册注销同样，基于 Spring 框

架的 Bean 机制，以及 @PostConstruct、@PreDestroy 注解，实现在系统启动和停止时对线程池的初始化和销毁。

```
public class TaskRunServiceImpl implements TaskRunService{
    …… // 省略了属性、线程池创建、销毁和其他方法代码
    @Override
    public boolean startTask(Long taskId) {
        Optional<Task> taskOptional = taskDao.findById(taskId);
        if (!taskOptional.isPresent() || taskOptional.get().isDeleted()) {
            log.info("任务不存在或被删除, ID: {}", taskId);
            throw new NotFoundException("任务不存在或被删除!");
        }
        Task task = taskOptional.get();
        if (task.getStatus() == TaskStatus.RUNING) {
            throw new BadRequestException("任务运行中!");
        }
        List<SubTask> subTaskList = subTaskDao.findAllByTaskId(taskId);
        for (SubTask subTask: subTaskList) {
            SubTasksExecutor subTasksExecutor = new SubTasksExecutor(subTask
                    , nodeService, this);
            this.scheduledQueue.schedule(subTasksExecutor
                    , subTasksExecutor.getDelayTime(), TimeUnit.SECONDS);
        }
        return true;
    }
    @Override
    public void addExecutor2Queue(PortabExecutor executor) {
        scheduledQueue.schedule(executor, executor.getDelayTime()
                , TimeUnit.SECONDS);
    }
}
```

图 4-14: 可移植性任务开始执行的实现

startTask() 方法接收可移植性测试任务 ID 作为参数，用于启动一个可移植性测试任务，该方法首先对该任务的状态进行校验，如果任务不存在或在运行中则抛出异常，正常则继续流程，查询出该任务的所有子任务，并将每个子任务封装成 SubTaskExecutor。由于 Spring 框架中多线程任务无法直接使用 @Autowired 注解注入对象 Bean，因此此处将其所需要调用的 NodeService 和 TaskRunService 对象实例作为参数传入 SubTaskExecutor 的构造函数中；然后会调用任务调度线程池的 schedule() 方法将 SubTaskExecutor 加入到线程池的队列

中，通过参数传入延时时间，指定单位为秒。

图 4-15所示是子任务移植部署调度执行的代码，SubTaskExecutor 是对子任务的封装，其和 ToolCallExecutor 均继承了 PortabExecutor，PortabExecutor 实现 Runnable 接口，并且属性包含 SubTask 和延时时间。SubTaskExecutor 和 CallExecutor 重写 Runnable 接口的 run() 方法，负责具体业务逻辑的执行。当 SubTaskExecutor 被线程池调度执行时，运行 run() 方法，调用 NodeService 的 applyNodeSignature() 方法申请资源签名，如果申请失败，会将 SubTaskExecutor 的延时时间增加 30 秒后重新加入延时队列等待调度；资源申请成功后会解析签名缓存的 Redis Key 中包含的节点 IP 信息，然后使用 HttpUtil 工具类中封装好的接口，调用对应节点上的测试环境生成模块，生成子任务的测试环境，运行待测应用，最后创建一个 ToolCallExecutor 加入队列中，等待执行调用自动化测试模块。ToolCallExecutor 的实现逻辑与 SubTaskExecutor 基本一致，因此不再赘述。

```
//PortabExecutor 实现了 Runnable 接口
public class SubTasksExecutor extends PortabExecutor {
    …… //省略了属性和构造函数等代码
     @Override
     public void run() {
          log.info("开始执行子任务, SubTask ID: {}", this.getSubTask().getId());
          String sigToken = nodeService.applyNodeSignature(this.getSubTask());
          //资源不足，申请签名失败
          if (sigToken == null) {
               //将该子任务重新入队
               this.increaseDelayTime(30L);
               taskRunService.addExecutor2Queue(this);
               return;
          }
          //签名缓存 Key 中包含节点 IP 信息
          String redisKey = SigTokenUtils.decryptToken(sigToken);
          String nodeIp = SigTokenUtils.parseIp(redisKey);
          //发往指定节点进行部署
          HttpUtil.sendSubTaskToDeploy(nodeIp, this.getSubTask(), sigToken);
          taskRunService.addExecutor2Queue(new ToolCallExecutor(this.getSubTask()
                    , taskRunService));
     }
}
```

图 4-15: 子任务移植部署调度执行的实现代码

图 4-16所示是 ToolCallBack 类的代码，该类封装对自动化测试模块回调结果的处理逻辑，当 RabbitMQListener 监听到运行完成的回调消息后会调用 ToolCallBack 的 process() 方法，传入参数是已经经过反序列化后的 ToolResults 对象，其中包含子任务 ID、自动化测试工具输出结果在 Redis 中的 Key 等信息。ToolCallBack 首先会查询出对应的 SubTask，根据 nodeIp，使用 HttpUtil 调用测试环境生成模块的接口，停止子任务的运行，然后调用 SubTaskDao 更新子任务在数据库中的状态；然后将 Celery 输出到 Redis 中的自动化测试结果读取出来，调用 RunDetailsDao 更新到 MongoDB 中；最后获取所属可移植性测试任务的所有子任务状态，当全部子任务都执行完成时，调用 TaskRunService 的 triggerAnalysis() 方法触发对可移植性测试任务的自动化分析。

```
@Slf4j
@Component
public class ToolCallBack {
    …… //省略属性代码
    public void process(ToolResults toolResults) {
        log.info("子任务运行结束, ID: {}", toolResults.getSubTaskId());
        Long subTaskId = toolResults.getSubTaskId();
        SubTask subTask = subTaskDao.findById(subTaskId).get();
        //调用环境生成模块接口，停止子任务，并更新该子任务运行状态
        HttpUtil.stopSubTask(subTask.getNodeIp(), subTaskId)
        subTask.setStatus(TaskStatus.TERMINATE);
        subTaskDao.save(subTask);

        //从 Redis 中读取结果，并更新到 Mongo 中
        RunDetails runDetails = runDetailsDao.findBySubTaskId(subTaskId);
        runDetails.setToolResult(toolResultMap.getByKey(toolResults.getRedisKey()));
        runDetailsDao.save(runDetails);

        //判断所属可移植性测试任务是否运行结束
        boolean isTaskOver = subTaskDao.findAllByTaskId(subTask.getTaskId())
                .stream().allMatch(s -> s.getStatus().equals(TaskStatus.TERMINATE));
        if (isTaskOver) {
            //触发结果分析
            taskRunService.triggerAnalysis(subTask.getTaskId());
        }
    }
}
```

图 4-16: 自动化测试模块回调结果处理实现代码

4.4 自动化测试模块详细设计与实现

4.4.1 自动化测试模块架构设计

图 4-17表示本系测自动化测试模块的架构设计，该模块主要采用异步调度的方式，封装了对自动化测试工具的调用，对外提供测试工具开始执行、结束、重试和状态查询等功能。

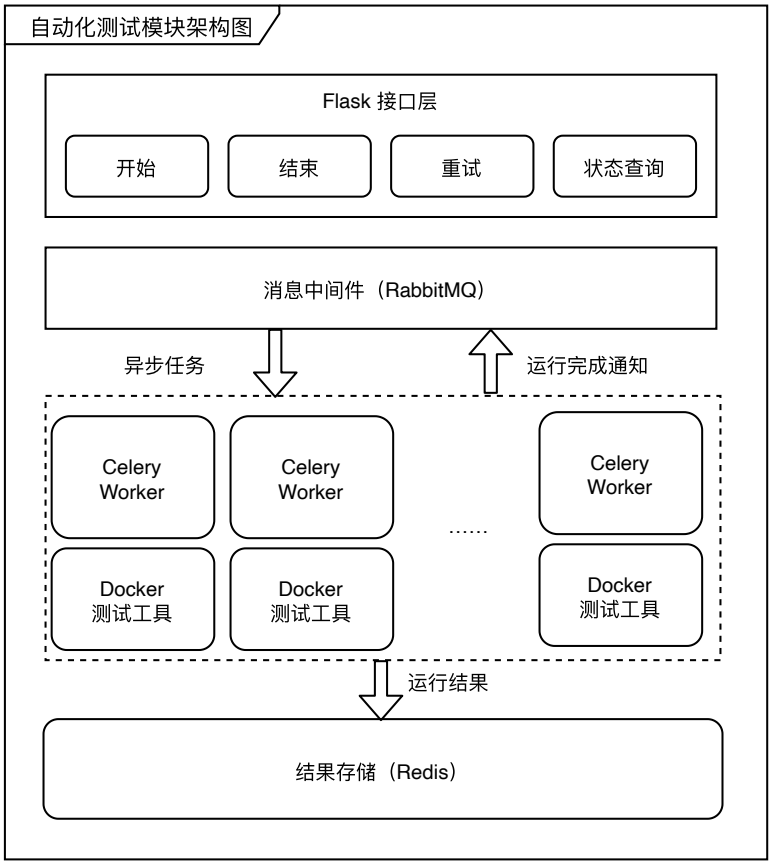

图 4-17: 自动化测试模块架构图

本模块使用 Python 开发，基于 Celery 框架实现异步任务调度，使用 RabbitMQ 作为 Celery 的 Broker，结果存储的 Backend 使用 Redis，Celery Worker 采用分布式的方式运行在服务器集群中，可根据服务压力动态伸缩调整。本系统中将自动化测试工具都封装成接口、参数统一标准的 Docker 形式，Celery Worker 执行时会根据传入的工具名称调用 Docker 命令启动容器运行工具，因此需要增加扩展新的自动化测试工具时只需要按照标准构建新工具的 Docker 镜像即可，系统无需修改代码。目前，自动化测试工具集成了 HttpRunner，用于

对运行在测试环境中的测试目标进行功能和性能测试。由于 Celery 能够很好的与 Python 的 Web 框架进行整合开发，本系统选用 Flask 进行接口层的开发，用于接收外部请求，将任务发布到 Celery。

4.4.2 自动化测试模块流程设计

自动化测试模块业务流程设计如图 4-18所示。

当接收到调用自动化测试工具的请求时，会先对请求的参数进行解析处理，组装成 Celery Worker 需要的参数格式，然后将测试任务发给 Celery，由 Celery 自动调度到 Worker 上进行执行。

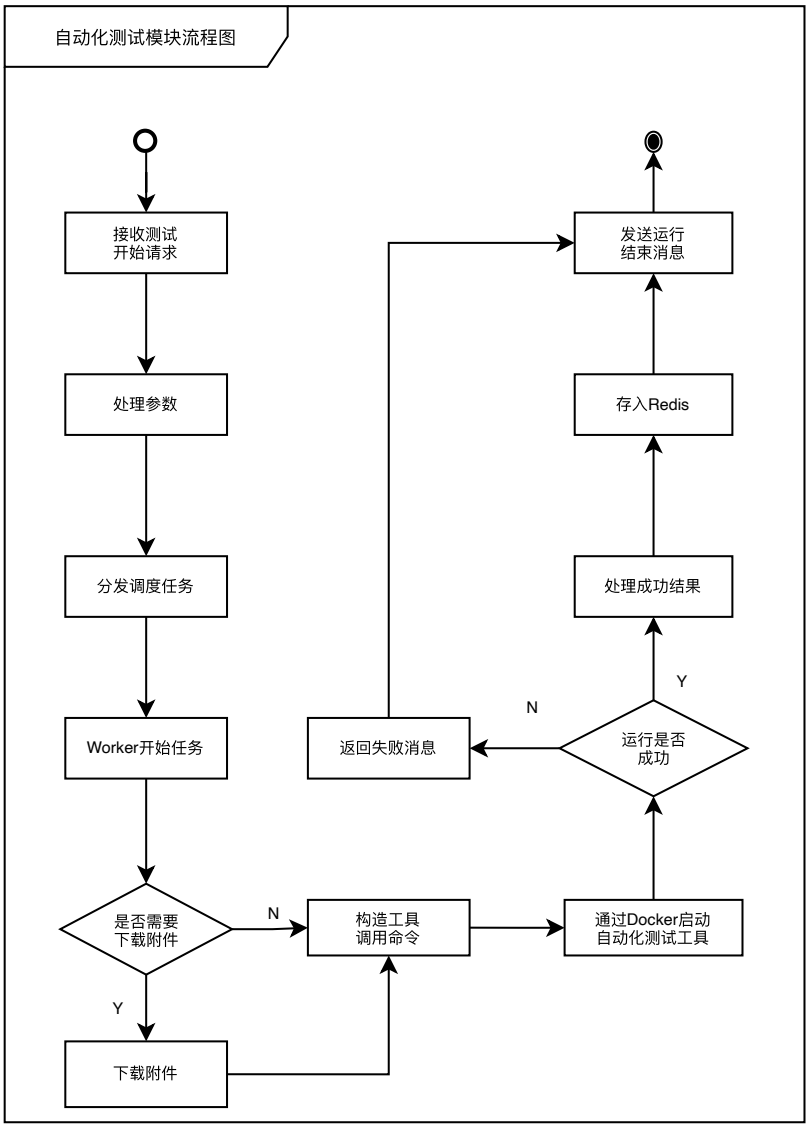

图 4-18: 自动化测试模块流程图

Worker 先判断是否需要下载文件附件，不需要则直接构造 Docker 运行自动化测试工具的命令，通过调用命令行启动自动化测试工具的容器；如果运行成功，容器运行结果通过标准输出返回，将其处理后存入 Redis 中，并发送消息到 RabbitMQ 中通知调用方测试已完成。

4.4.3 自动化测试模块关键代码

图 4-19所示是自动化测试模块工具调用接口的实现代码。@flask.route 装饰器声明了接口的路由和调用方法类型，当该接口被调用时，使用 flask 的 request 模块获取请求中的参数，如工具名 toolName、工具对应消息队列名称 queueName 等。由于该接口支持一次调用传入多个测试任务，因此将一次接口调用定义为一个 session，一个 session 中包含一个或多个 job，每个 job 对应执行一次自动化测试工具，所以通过循环处理请求中的 job，然后使用 Celery 中 Signature 模块将 job 发送到 Broker 中进行调度执行，最后返回 sessionId 等信息给接口调用方，可用于查询任务状态、停止、重试。

```
…… # 省略 import 和读取配置的代码
@flask.route('/celery/start', methods=['POST'])
def start_test():
    signatures = []
    session_tool_name = request.json['toolName']
    args = request.json['jobs']
    queue_name = request.json['queueName']
    for i in range(0, len(args)):
        if 'toolName' in args[i].keys() and args[i]['toolName'] != '':
            tool = args[i]['toolName']
        else:
            tool = session_tool_name
        signatures.append(Signature(task='clientWrapper.start', args=(tool,
                str(args[i]['params']), queue_name), queue=tool + 'Queue'))
    task_group = group(signatures, app=app)
    group_result = task_group()
    job_infos = []
    session_id_valid = False
    …… #省略了获取 session_id 的代码
    result = {'sessionId': final_result.id, 'jobs': job_infos, 'queueName': queue_name}
    return jsonify(result)
```

图 4-19: 自动化测试工具调用接口实现代码

图 4-20所示是 Celery Worker 中运行自动化测试工具的实现代码。通过使用 @app.task() 装饰器将方法定义为一个 Celery 可执行的任务，bind 参数表示将其定义为一个绑定任务，意味着方法的第一个参数总是任务实例本身 (self)，和 Python 绑定方法类似。

```
@app.task(bind=True)
def start(self, tool_name, args, queue_name):
    ··· #省略一些参数处理代码
    try:
        # prepare args
        args_dict = eval(args)
        os.mkdir(task_id)
        download_urls = args_dict['downloadURL']
        extra_args = args_dict['extraArgs']
        record_tool_name(task_id, tool_name)
        self.update_state(state='PROGRESS_DOWNLOADING')
        download_files(task_id, download_urls)
        self.update_state(state='PROGRESS_RUNNING')
        image_name = tool_name + ":latest"
        this_path = os.getcwd()
        path_map = this_path + '/' + task_id + ':/workspace'
        mvn_map = '/root/.m2:/root/.m2'
        command = 'docker run --rm --name %s -v %s -v %s %s start %s' \
                  % (task_id, path_map, mvn_map, image_name, extra_args)
        commands = command.split(' ')
        c = subprocess.Popen(commands, stdout=subprocess.PIPE, \
                             stderr=subprocess.PIPE)
        (stdout, stderr) = c.communicate()
    # 省略了结果返回与异常处理代码
```

图 4-20: Celery Worker 运行自动化测试工具实现代码

Celery Worker 执行一次任务时，就是处理上游发送的一个 job。任务方法首先对传入参数进行处理，调用 os 模块创建目录作为该任务的工作空间，并将该任务的状态置为“PROGRESS DOWNLOADING”，然后调用 download_files() 方法将任务包含的附件下载到工作空间；下载完成后将任务状态置为“PROGRESS RUNNING”，然后构建 Docker 运行自动化测试工具容器的命令，在命令中将工作目录和依赖目录挂载到容器，加上容器运行的其他所需参数，使用 subprocess 的 Popen() 方法启动进程执行构造好的命令，并获取作为运行结果的标准输出。在此过程中会捕获异常信息，最后对运行结果或异常进行处理，返回结果会自动存入 Redis，存储的 Key 为相应 job 的 ID。

4.5 自动化分析模块详细设计与实现

4.5.1 自动化分析模块架构设计

图 4-21表示自动化分析模块的架构设计，本模块主要负责对应用运行日志进行处理分析，根据自动化测试工具输出的结果计算 Java Web 应用的可移植性指标，包括功能表现、性能表现、移植效率和适应性四个指标，并获取 Java Web 应用在各个测试环境中运行的监控数据，整合出可移植性测试报告数据。

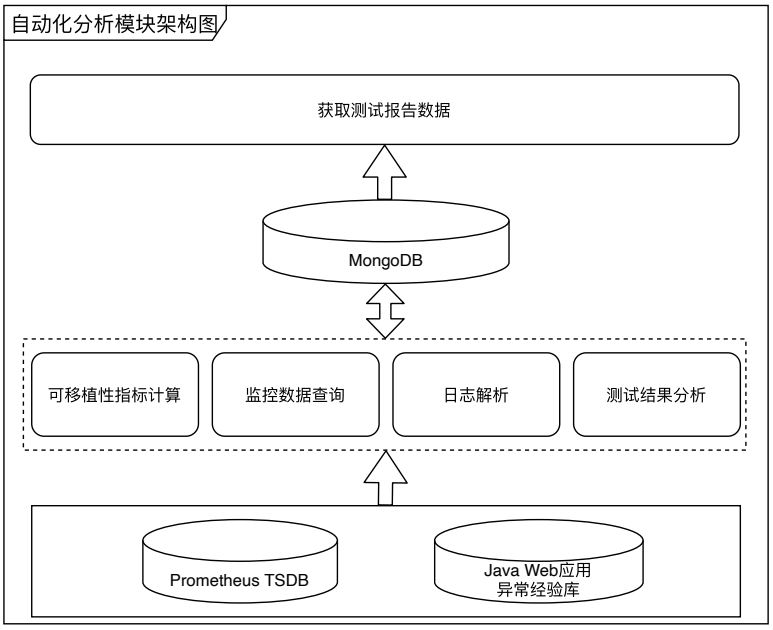

图 4-21: 自动化分析模块架构图

本模块的实现依赖于测试环境生成模块、自动化测试模块和监控服务在可移植性测试任务执行过程中记录下的数据。测试环境生成模块会记录下子任务运行的容器日志，调度运行模块会将自动化测试模块返回的测试结果更新到 MongoDB 中的 RunDetails 中，同时 Prometheus 会通过 Container Exporter 采集测试环境容器运行的监控数据存入 Prometheus 的时序数据库。基于上述这些数据，当自动化分析模块触发执行分析后，会根据可移植性测试任务的信息，从 MongoDb 中查询出每个子任务的日志和自动化工具测试结果，使用日志分析脚本解析日志并根据 Java Web 应用异常经验库分析出运行日志中包含的异常信息；调用 Prometheus 提供的查询 API，获取每个子任务对应容器运行期间的监控数据。最后对分析和查询到的结果数据进行整合成可移植性测试的最终结

果，写回 MongoDB 中，当调用获取测试报告接口时，直接从 MongoDB 中读取出来即可，无需重复计算。

4.5.2 自动化分析模块类图设计

自动化分析模块核心类图设计如图 4-22所示。AnalysisServcie 接口类是对可移植性测试任务结果数据进行自动化分析的相关功能逻辑的抽象和封装，AnalysisServiceImpl 是其具体实现类，对外主要提供触发分析的 triggerAnalysis() 方法和获取可移植性测试报告的 getReport() 方法，这两个接口的传入参数均为系统中可移植性测试任务的 ID。AnalysisServiceImpl 需要使用 TaskDao、SubTaskDao、TaskResultDao 和 RunDetailsDao 等持久化接口，从 MySQL 和 MongoDB 中读取出运行和测试过程中记录的数据，并在分析结束后将最终结果在回写到数据库。

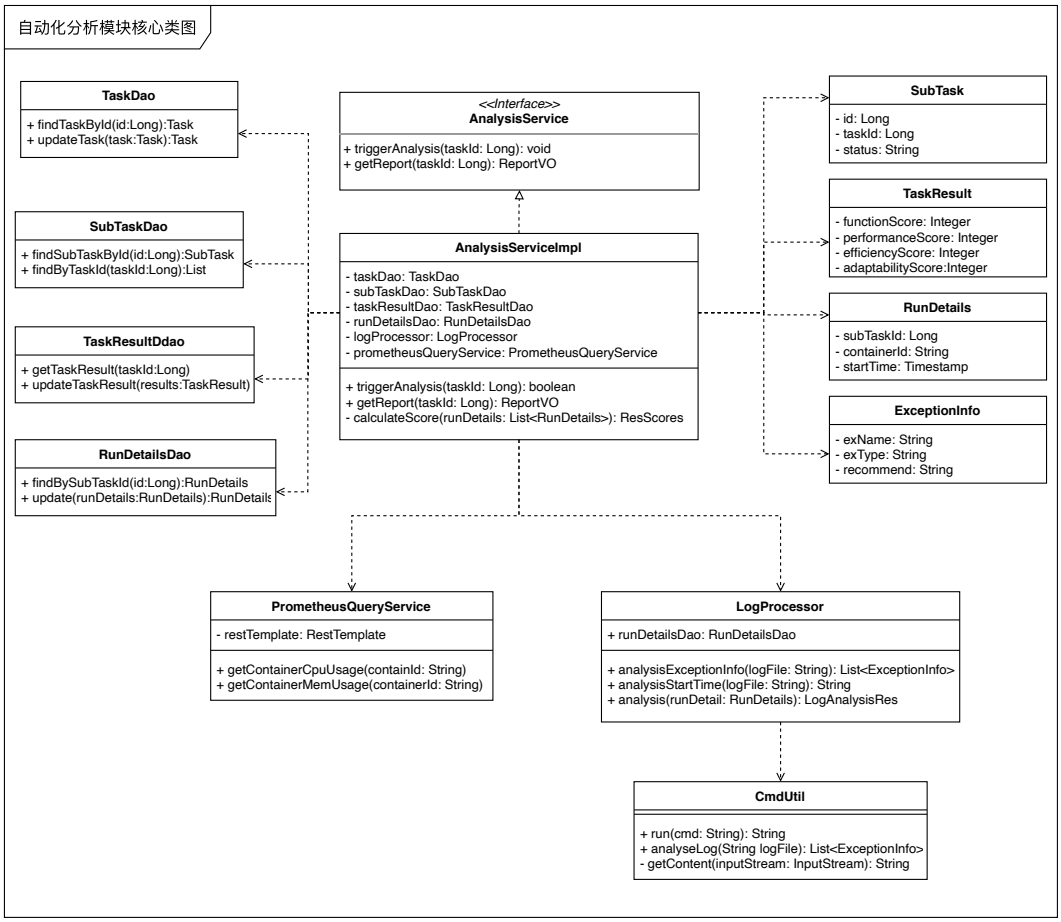

图 4-22: 自动化分析模块核心类图

Prometheus 提供了 RESTful 的 API 查询其 TSDB 中的监控时序数据，查询语句遵循 PromSQL 语法，因此 PrometheusQueryService 类对查询 Prometheus 的操作进行了封装，提供 getContainerCpuUsage() 和 getContainerMemUsage() 方法用于获取每个子任务测试环境运行时的 CPU 和内存状态。LogProcessor 类提供日志分析的接口，其会调用封装了执行命令行功能的工具类 CmdUtil 去运行 Python 脚本对运行日志分析，并对从标准输出返回的日志分析结果进行处理，日志分析主要包括通过日志解析应用实际启动的耗时，以及分析日志中的异常、异常类型和原因等信息。

4.5.3 自动化分析模块顺序图

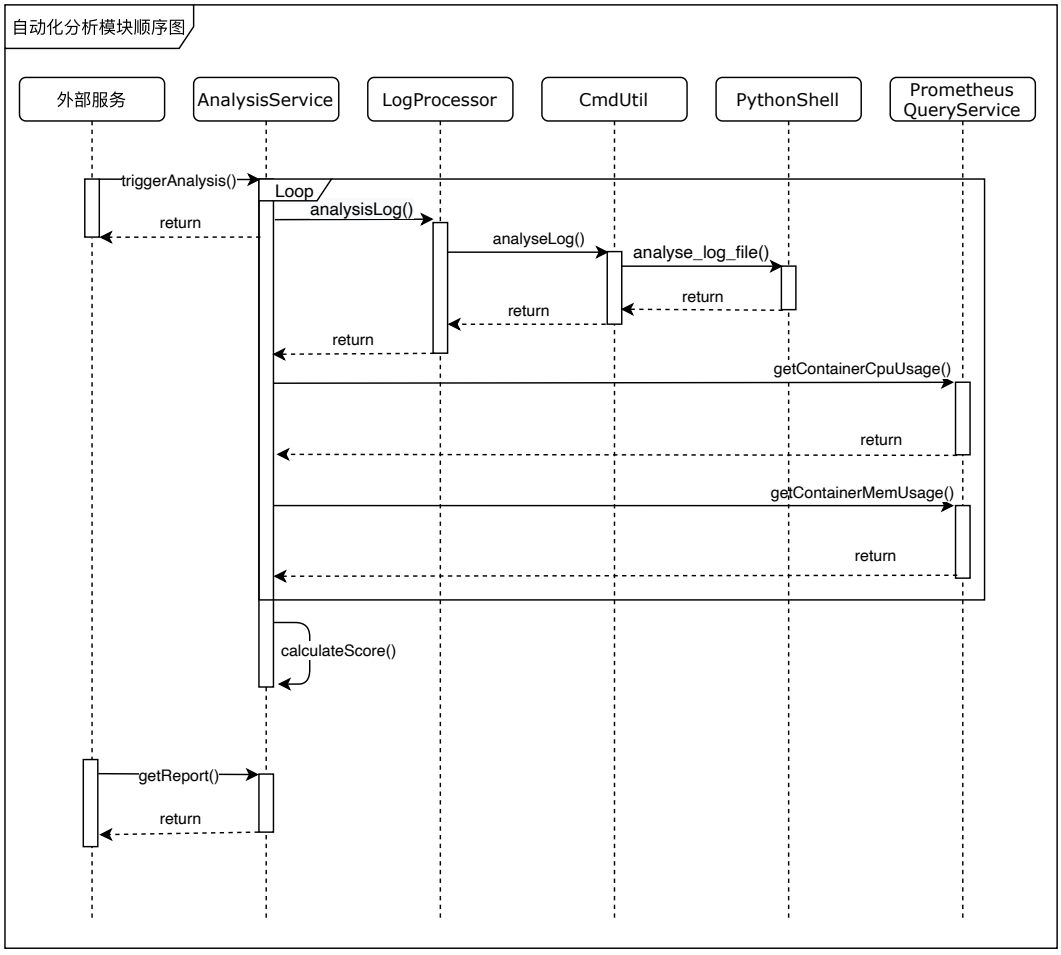

图 4-23: 自动化分析模块顺序图

自动化分析模块的顺序图如图 4-23所示。当调度执行模块识别到所有子任务均运行完成后，会调用 AnalysisServcie 的 triggerAnalysis() 接口触发对可移植

性测试任务的结果分析；

AnalysisService 会先查询出可移植性测试任务的信息，以及该任务的所有子任务，并从 MongoDB 中读取各个子任务的 RunDetails；接着调用 LogProcessor 的 analysisLog() 方法对 RunDetails 中记录的运行日志进行分析，LogProcessor 会调用 CmdUtil 运行 Python 脚本进行日志分析，Python 脚本会对日志内容重的异常进行识别提取，比对异常经验库分析识别到的异常类型、原因等，同时解析出日志中应用的启动耗时，返回给 LogProcessor；日志解析完成后，调用 PrometheusQueryServcie 的 getContainerCpuUsage() 和 getContainerMemUsage() 方法获取 CPU 和内存监控信息。每个子任务都需要日志分析和监控数据获取，因此该过程需要循环执行。

当所有子任务的日志和监控数据处理完成后，AnalysisServcie 会调用自身的 calculateScore() 方法，综合各个子任务的运行和分析结果，计算 Java Web 应用的可移植性指标。分析完成后，外部服务可调用 AnalysisService 的 getReport() 方法获取测试报告，getReport() 方法会从数据库中查询出任务信息和结果数据，组装成 ReportVO 返回，调用者根据需求进行渲染。

4.5.4 指标评估计算说明

本文参考目前关于软件系统可移植性的相关研究，结合 Java Web 应用的特性，主要计算功能表现、性能表现、移植效率和适应性四个指标，从而评估 Java Web 应用的可移植性，指标仅能给测试需求方提供参考，因此在测试报告中会给出测试过程中的日志、异常分析、监控数据和自动化工具测试结果，方便测试需求方进行进一步的评估与决策。四个指标的计算方法如下：

功能表现通过对 Java Web 应用的功能接口测试通过率来表示，HttpRunner 会根据任务中的接口用例列表进行自动化测试，根据测试结果计算所有子任务的平均接口用例通过率；

性能表现通过计算 HttpRunner 测试结果中的平均接口响应时间来表示，根据业内常见标准，Java Web 应用接口响应时间应小于等于 300 毫秒，超过 600 毫秒认为是不能接受的，因此将平均时间小于等于 300 毫秒定义为 100 分，600 毫秒定义为 0 分，对性能表现进行评估；

移植效率通过所有子任务测试环境中 Java Web 应用的平均启动耗时来进行评估；

适应性通过子任务运行日志中出现的异常数量来评估，为不同类型异常设

置不同的权重，对每个子任务的适应性进行评分，最后计算平均值来评估 Java Web 应用的适应性。

4.5.5 自动分析模块关键代码

图 4-24表示触发自动化分析的 triggerAnalysis() 方法的实现代码，对于要触发自动化分析的可移植性测试任务，首先会查询出所有的子任务，然后进行遍历操作，通过 runDetialsDao 从 MongoDB 查询出每个子任务对应的 RunDetails，然后调用 LogProcessor 对 Rundetails 中保存的运行日志进行分析处理，接着获取子任务测试环境的监控数据，将日志分析结果和监控数据保存到 RunDetails，并更新到 MongoDB；遍历结束后，调用 calculateScore() 方法计算四个指标数据，并保存到 MongoDB 的 TaskResult 集合。

```
public void triggerAnalysis(Long taskId) {
        List<SubTask> subTaskList = subTaskDao.findByTaskId(taskId);
        TaskResult taskResult = taskResultDao.findById(taskId);

        List<RunDetails> runDetailsList = new ArrayList<>();
        for (SubTask subTask : subTaskList) {
            RunDetails runDetail = runDetailsDao.findBySubTaskId(subTask.getId());
            //日志分析
            LogAnalysisRes logAnalysisRes = logProcessor.analysis(runDetail);
            runDetail.setExceptionInfoList(logAnalysisRes.getExceptions());
            runDetail.setDeployDuration(logAnalysisRes.getDuration());
            //获取监控数据
            runDetail.setCpuUsage(prometheusQueryService
                    .getContainerCpuUsage(subTask.getContainerId()));
            runDetail.setMemUsage(prometheusQueryService
                    .getContainerMemUsage(subTask.getContainerId()));
            runDetailsDao.save(runDetail);
            runDetailsList.add(runDetail);
        }
        //评估指标
        ResScores scores = this.calculateScore(runDetailsList);
        BeanUtils.copyProperties(scores, taskResult);
        taskResultDao.save(taskResult);
}
```

图 4-24: 触发分析实现代码

图 4-25所示是用于分析处理运行日志的 Python 脚本代码，其会对传入日

志中的异常和启动耗时信息进行提取，主要通过对 Java Web 应用的异常关键字构造正则表达式，进行检索匹配，如”Caused by”，截取出异常信息，然后查询异常经验库进行分析，获取异常的类型、原因等信息；最后将分析结果通过标准输出返回给脚本的调用者。

```
def analyse_logs_str(logs_raw):
    pattern_causred_by = "Caused by: (.*)"
    pattern_at = "at.*"
    translator = Translator(from_lang="english", to_lang="chinese")
    # 异常分析存储字典
    exce_dict = {}
    for i in range(len(logs_raw)):
        raw = logs_raw.iloc[i]["txt"]
        # 找到 Causred by 开头的列
        clue = re.search(pattern_causred_by, raw)
        if clue is not None:
            # 获取异常信息
            exce_info = clue.group(1)
            # 将异常类别和具体原因分离，存储为数组
            exce_info_list = exce_info.split(":", 1)
            j = i
            # 向上遍历，找到第一个开头不为 at 的行即为具错误信息
            message = re.search(pattern_at, logs_raw.iloc[j-1]["txt"])
            while(message is not None):
                j = j-1
                message = re.search(pattern_at, logs_raw.iloc[j]["txt"])
                message_temp = logs_raw.iloc[j]["txt"]
            exce_dict[message_temp] = exce_info_list
    # 省略了查询经验库代码和解析部署时间等代码
    ......
    analyse_json = json.dumps(result_list, ensure_ascii=False)
    print(analyse_json)
```

图 4-25: 运行日志分析脚本代码

4.6 系统实例展示

图 4-26所示是使用系统上传测试目标的页面，用户在该页面可上传 Java Web 应用的可运行包，支持 Jar 包和 War 包两种格式，并填写测试目标的名称，后续选择已有测试目标创建可移植性测试任务。

图 4-27所示是查看可移植性测试任务列表页面，该页面向用户展示了当前已经创建的可移植性测试任务，以及其当前状态，并提供根据状态进行筛选和关键字搜索功能，方便用户根据需要查看。

图 4-26: 上传测试目标截图

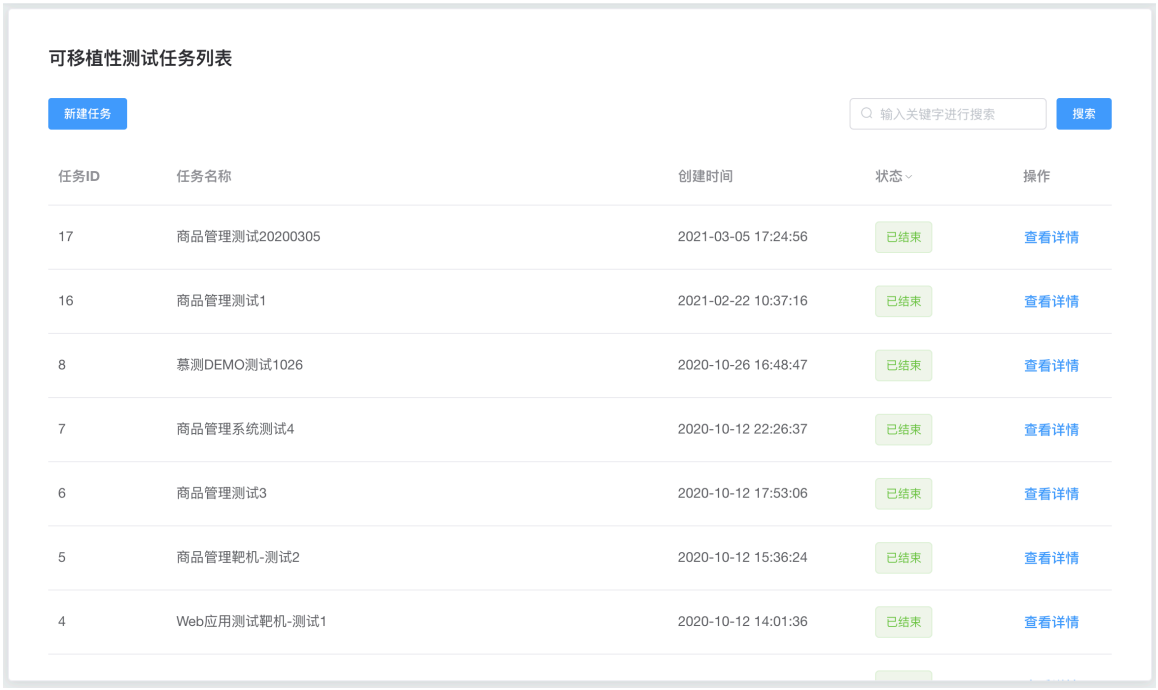

图 4-27: 测试任务列表页面截图

图 4-28所示是创建 Java Web 应用可移植性测试任务的页面。用户在该页面需要填写任务名称，选择已上传的测试目标，并且可根据需要指定 Jar 包的启动命令；用户可以配置一系列 HTTP 接口，用于进行功能和性能的验证测试，系统支持两种配置方式，一种是逐个接口手动填写路径、请求参数等信息，第二种是直接上传 HttpRunner 标准的 YAML 或 JSON 文件批量导入接口，

当用户在接口配置方式单选框选择批量上传时即可选择文件进行上传；用户可以根据系统提供的可选项选择组合不同的操作系统、计算资源和软件依赖，生成不同的测试环境进行可移植性测试。

图 4-28: 创建测试任务页面截图

图 4-29所示是可移植性测试任务详情页面展示的测试报告，该页面给出对 Java Web 应用可移植性指标的评分；并且展示各个测试环境中自动化测试工具输出的报告、CPU 和内存资源使用监控图、运行日志等信息，为提测方进一步

分析和决策提供参考。

图 4-29: 测试结果报告页面截图

4.7 本章小结

本章对系统环境生成模块、节点控制模块、调度执行模块、自动化测试模块和自动化分析模块的详细设计与实现进行介绍。通过架构图、UML 类图、顺序图等方式对各个模块的设计实现思路进行阐述，并给出每个模块关键功能的实现代码。最后通过实际运行页面截图和文字描述，对系统进行了实例展示。

第五章　系统测试

5.1　测试目标

为保证本系统在上线后能够提供稳定可用的服务，以及在未来的开发迭代中保证软件质量，需要对系统进行全方面的测试。基于前两章的工作，本文主要一下三个方面进行系统测试的设计和执行。

（1）单元测试。单元测试是对系统中最小功能粒度进行测试的方法，也被称作模块测试 [40]，在本系统中最小功能粒度即为函数方法。单元测试通常根据需求在开始过程中有开发者编写，能够从功能代码层面保证系统的质量。在实践中，通常采用 Mock 的方式，将被测方法对其他模块的依赖屏蔽掉，实现单元测试的独立性。

（2）接口测试。接口测试是针对系统对外提供功能 API 进行的一种黑盒测试，用于保证系统对外提供的服务质量和可靠性，验证其是否符合交付需求、性能需求 [41]。对于 Web 系统，通常通过设计测试用例调用接口，验证接口是否能够给出符合预期的响应。

（3）验收测试。验收测试向用户证明系统能够按照需求预期进行工作，常见的验收测试手段有 Beta 测试等。此项测试用于验证系统的产品成熟度和可交付能力，测试系统在真实场景中使用是否正常，验收测试通常脱离开发者，完全由使用者进行，能否覆盖常见待测应用，并且不应有系统崩溃、无响应等异常情况。

单元测试与接口测试用于保证系统的功能稳定性，在系统后续的开发和维护中，应采用持续集成与持续测试的方式，对项目的每一次构建都应执行一次单元测试和接口测试的回归，保证代码和功能的正确性，避免代码修改和功能扩展的过程中引入 Bug。验收测试保证本系统能够符合最终用户的预期，并且能够在生产环境中对真实 Java Web 应用进行测试。

5.2 测试环境

表 5-1展示系统测试环境中服务部署涉及到的服务器硬件资源，主要包括 CPU、内存、带宽和磁盘空间大小。本系统中，自动化测试模块是独立服务，因此将其独立部署，另外分别部署可移植性测试服务 Master 和 Slave，构成集群。

表 5-1: 系统测试硬件环境

服务	服务器规格	CPU	内存	带宽	磁盘
可移植性测试服务 Master	阿里云 ecs.s6-c1m4.xlarge	4 核	16G	4M	500G
可移植性测试服务 Slave	阿里云 ecs.s6-c1m4.xlarge	4 核	16G	1M	500G
自动化测试服务	阿里云 ecs.c5.xlarge	4 核	8G	1M	500G

表 5-2展示系统运行所需的软件环境依赖和版本，基础操作系统均使用 Ubuntu 16.04；开发语言环境主要为 Java 8 和 Python 3.6；数据库和中间件分别选用了 Redis 4.0.11、MySQL 5.7、RabbitMQ 3.7.4、Mongo 4.2.2；Docker 所用版本为 19.03.13；自动化测试工具 HttpRunner 版本为 2.5.7。

表 5-2: 系统测试软件环境

环境项	版本
操作系统	Ubuntu16.04
JDK	JAVA 1.8.0_191
Python	Python 3.6
Redis	Redis 4.0.11
MySQL	MySQL 5.7
RabbitMQ	RabbitMQ 3.7.4
MongoDB	Mongo 4.2.2
Docker	Docker 19.03.13, build 4484c46d9d
HttpRunner	HttpRunner 2.5.7

5.3 单元测试

单元测试作为软件测试的重要组成部分，在整个软件生命周期中也是不可缺少的一环。本系统开发主要使用 Java 语言，因此选取业内主流的测试框

架 JUnit 作为本系统单元测试框架，并同时结合 Mockito 和 PowerMockito 两个 Mock 框架，完成单元测试过程中 Mock 其他模块与数据的工作。

单元测试需要按照系统的架构模式进行，本系统采用 MVC 分层设计模式，共分为 Controller 层、Service 层和 Dao 层，根据各个层所负责的主要功能指责，在对不同层进行单元测试所关注的重点也不同 [42]。Controller 层主要负责对请求的转发控制和参数校验处理，因此在对 Controller 层进行单元测试时主要验证其对异常请求参数的校验；Service 层负责系统的业务逻辑处理，因此其单元测试重点在于验证业务流程的正确性，以及对异常流程的识别与响应；Dao 层主要负责与数据库的交互，其测试重点在于验证数据库操作执行的正确性；由于不同层次之间、各个功能模块之间都存在依赖，因此为保证单元测试更加内聚与被测方法本身，需要使用 Mockito 和 PowerMockito 去模拟对其他模块的调用与返回。

表 5-3: 单元测试用例

单元测试项	类型	说明	用例数量
ManageControllerTest	接口层	测试目标与任务管理接口	8
TestEnvControllerTest	接口层	测试环境生成模块接口	3
OssServiceTest	Service 层	Oss 服务	3
DeployRunServiceTest	Service 层	测试环境生成部署服务	4
DockerServiceTest	Service 层	Docker 调用服务	4
NodeServiceTest	Service 层	节点控制服务	6
TaskRunServiceTest	Service 层	调度执行服务	6
ToolCallServiceTest	Service 层	自动化测试调用服务	3
AnalysisServiceTest	Service 层	自动化分析服务	5
TaskDaoTest	Dao 层	可移植性测试任务 Dao	3
SubTaskDaoTest	Dao 层	子任务 Dao	3
TargetDaoTest	Dao 层	测试目标 Dao	3
DeployNodeDaoTest	Dao 层	集群节点 Dao	3
RunDetailsDaoTest	Dao 层	子任务结果 Dao	3
TaskResultDaoTest	Dao 层	可移植性测试任务结果 Dao	3
总计			60

根据上述思路，本系统的单元测试用例统计情况如表 5-3所示。共计 15 个测试项，包含 Controller 层 2 项，分别针对提供测试目标与任务管理相关接口的 ManageController 和测试环境生成模块接口 TestEnvController 进行测试；Service 层 7 项，主要对 OssService、DeployRunService、NodeService 等进行测

试，验证他们的业务逻辑以及异常处理响应；Dao 层 6 项，分别对 MySQL 读写查询和 MongoDB 读写查询的 Dao 进行测试，暴扣 TaskDao、RunDetailsDao 等；所有单元测试用例共计 60 个。

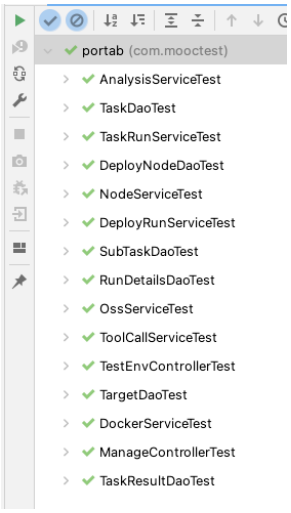

图 5-1: 单元测试结果

单元测试结果如图 5-1所示，通过率达到 100%（60/60）。在后续的迭代中，应不断维护扩充单元测试用例，并通过持续集成与测试的方法，在每一次构建发布前都运行全部单元测试用例，只有全部通过才能进行发布，从而保证功能的正确性。

5.4 接口测试

接口测试是对 Web 服务接口的功能正确性以及性能进行验证的一种黑盒测试，通常在 Web 系统内部可能由于迭代产生诸多变化，但是对外接口的变现往往稳定不变，因此通过对接口进行测试对服务整体质量具有重要意义，同时也是对服务进行自动化测试的重要一环。

本系统为面向 Java Web 应用可移植性的自动化测试系统，采用前后端分离模式开发部署，系统后端对外暴露了可移植性测试任务相关的接口能力，上游服务也可以是系统前端以外的其他调用者，因此需要对本系统后端接口进行测试，保证其对外提供正确可用的服务。本文进行接口测试侧重点在于接口的数据一致性和接口响应是否符合预期。

本系统接口测试工具使用 Postman，其为后端开发中常用的接口测试工

表 5-4: 接口测试用例

测试 ID	说明	输入	预期输出	结果
IT-1	启动测试正常流程	正确任务元信息对象	正确任务状态对象	通过
IT-2	启动测试异常流程	缺少参数的任务元信息对象	抛出 HTTP400 异常	通过
IT-3	获取任务状态正常流程	正确的任务 ID	任务状态对象	通过
IT-4	获取任务状态任务 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-5	停止任务正常流程	正确的任务 ID	true	通过
IT-6	停止任务 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-7	正常执行自动化测试流程	正确的任务 ID	true	通过
IT-8	执行自动化测试 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-9	正常查询自动化测试状态流程	正确的 SessionID	true	通过
IT-10	测试查询自动化测试 SessionID 不存在流程	错误的 SessionID	抛出 HTTP404 异常	通过

具，提供了方便且强大的接口调用能力。本系统接口测试用例如表 5-4所示，主要包含业务系统本身的接口和自动化测试模块 Flask 提供的接口。

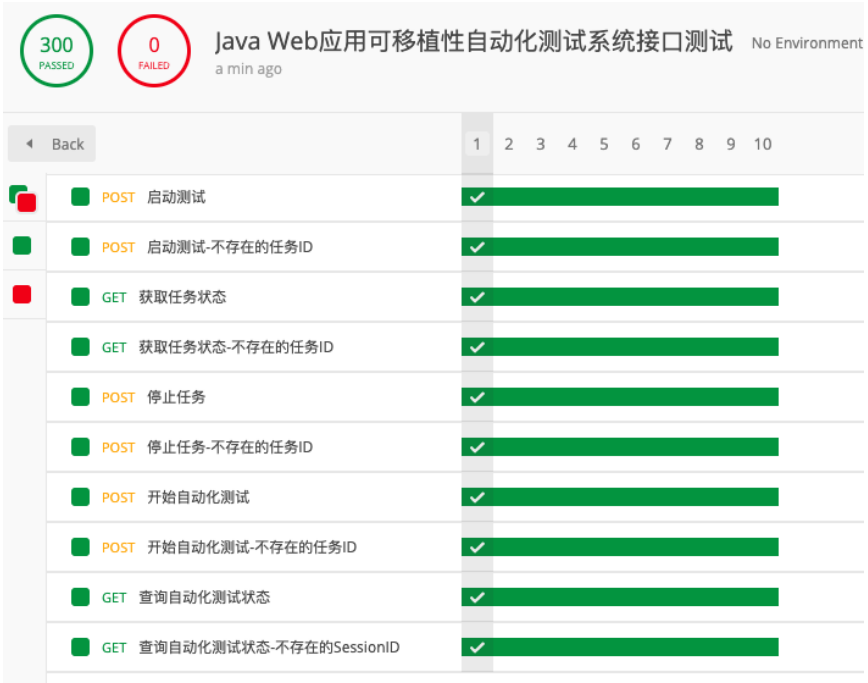

图 5-2: 接口测试结果

根据上述接口测试用例共有 10 个请求，其中 4 个 GET 请求和 6 个 POST

请求，执行时对这些请求返回的状态码、响应时间和响应体内容这个三个单元进行校验，共计 30 个测试单元；接口测试共迭代进行了 10 轮，共 300 次测试5-2所示，测试全部通过。

5.5 验收测试

验收测试是技术测试的最后一个阶段，通常在产品完成单元测试、功能测试等其他阶段之后，在产品交付之前进行，因此也被称为交付测试。为保证系统符合最终用户的预期，能够在生产环境中平稳运行，本系统上线后，组织真实用户对其进行验收测试。

表 5-5: 验收测试待测应用集

应用序号	应用名称	格式	大小	类型
1	商品销售管理系统	War 包	27M	购物类
2	失物招领平台	War 包	34M	生活类
3	大学女生话题论坛	War 包	17M	社交类
4	预约挂号后台管理中心	War 包	23M	工具类
5	房屋租赁系统	War 包	24M	生活类
6	机票预订系统控制台	War 包	37M	工具类
7	支付服务	Jar 包	69M	金融类
8	测试用例生成服务	Jar 包	63M	工具类
9	订单查询服务	Jar 包	50M	购物类
10	数据扩增服务	Jar 包	49M	工具类
11	Sushine-FactoryModel	Jar 包	113M	工具类
12	Sushine-Gateway	Jar 包	91M	工具类
13	Sushine-System	Jar 包	94M	工具类
14	Sushine-TrendTool	Jar 包	103M	工具类
15	Sushine-Eureka	Jar 包	40M	工具类
16	tx-manager	Jar 包	51M	工具类
17	公众号服务	Jar 包	55M	阅读类
18	WebIDE 后端系统	Jar 包	63M	学习类
19	WebIDE 推荐服务	Jar 包	50M	工具类
20	可移植性测试系统	Jar 包	49M	工具类

在本系统的验收测试中，测试执行人员从某公司收集了 20 个真实的 Web 应用进行测试，其中包含 6 个 War 包类型应用和 14 个 Jar 包类型应用；待测应用涵盖工具类、生活类、社交类等多种类型，基本能够覆盖常见的 Web 应用场

景，待测应用集的具体信息如表 5-5所示。

表 5-6: 验收测试结果

执行结果	应用数量
正常执行	19
非正常执行	1

基于上述应用集，本次验收测试根据系统能否成功运行测试任务并输出测试结果报告，作为待测应用是否可以正常执行测试的判断依据。实验结果如表 5-6所示，20 个应用中有 19 个均正常执行，测试通过率为 95%；有 1 个应用未正常执行，但其并未引起系统的死锁、无响应等异常问题。我们对该应用无法正常执行测试的原因进行分析，发现是由于该应用缺少了一些外置的静态文件，导致无法启动并且没有输出日志，该问题是由于应用自身导致的问题，但后续可对本系统进行改进，将这类特殊情况在结果中进行报告，以便于测试人员判别。

5.6 本章小结

本章从功能可用性的角度对系统进行代码层面进行单元测试，从功能接口层面进行测试，详细介绍了测试的设计和结果，可以极大程度保证系统的质量；另外，收集整理了 20 个真实的 Java Web 应用对本系统进行验收测试，保证了系统上线后能够正常提供服务。

第六章　总结与展望

6.1　总结

随着互联网软件技术的快速发展，软件产品越来越多样化，Web 应用作为一种无需安装、使用方便的软件类型逐渐受到广大用户和开发者的欢迎，在人们日常生活和工业生产等各个方面都发挥着重要作用。Java 作为一种主流的 Web 开发语言，在 Web 应用市场占据非常高的份额，Java Web 相关开发技术与生态也十分丰富；但随着需求的多样化和软件规模不断扩大，Java Web 应用的部署方式与环境也不断变化，另外由于云计算技术成熟，公网产品私有云化趋势，以及国产软硬件的快速发展带来适配行测试需求等因素，也对 Java Web 应用的可移植性能力提出了更高的要求。

目前面向 Java Web 应用进行的可移植性测试主要依赖于人工，需要测试人员手工搭建各种不同的测试环境，并对部署在不同环境中的应用进行功能验证和性能测试。这些过程需要耗费大量的时间与成本，且难以跟上如今频繁迭代，持续集成的高效开发模式。针对上述问题，一种可行的解决方案是通过技术手段将 Java Web 应用可移植性测试过程中人工成本较大的步骤自动化，代替人工完成环境搭建、应用部署和验证测试等该工作，并输出具有参考性的测试报告，提高整体测试效率。因此本文提出一种面向 Java Web 应用可移植性的自动化测试技术，并通过实现 Web 系统为测试人员提供服务。

本系统使用容器技术实现自动化生成部署所需测试环境，测试人员只需要简单地选择环境配置组合，系统会根据需求生成对应的测试环境，并自动运行 Java Web 应用。本系统基于 Celery 和 HttpRunner 搭建一套自动化测试服务框架，对于在各个测试环境中运行的 Java Web 应用，系统会调用该服务进行自动化测试，验证其功能与性能是否符合预期；另外在测试环境运行期间会对日志和监控数据进行采集，最后综合各项数据分析计算指标，并生成可移植性测试报告，简化测试人员进行 Java Web 应用可移植性测试的工作量，提高测试效率。

本文在第一、二章详细介绍了论文的背景，以及软件可移植性测试的相关

研究现状，并对系统开发过程中所依赖的相关理论和关键技术进行阐述。在第三章使用软件工程的方法对本系统的业务需求进行了分析，并对系统整体架构进行概要设计，首先对系统所面向的目标用户进行了梳理，以此为基础对系统非功能性和功能性需求，以及系统用例进行了详细分析与阐述；然后根据分析的结果对系统的整体架构和持久化模型进行概要设计，并使用 4+1 视图从不同角度对系统的架构体系与模块划分进行了展示。第四章通过使用架构图、类图和顺序图，分别对系统各个模块对详细设计与实现进行介绍，并通过关键代码展示系统实现细节，辅助理解，最后通过实例截图的方式展示了系统实际运行效果。为保证系统的质量和可用性，本文在第五章中介绍了对系统进行的测试工作，包括单元测试、接口测试，并收集了 20 个真实应用对系统进行了验收测试。

6.2 展望

目前本系统已完成部署，并上线投入使用，但还存在一些问题和不足，可在后续迭代维护中进行改进与优化，主要包括以下几个方面。

（1）系统自动化测试模块为工具接入提供了良好的扩展性，目前仅集成了 HttpRunner 测试工具，后续可以接入更多用于 Web 应用可移植性测试的工具，丰富测试的手段，但是缺少测试流程编排的能力，之后可以通过扩展 Celery 对工作流的支持，开发测试流程编排功能。

（2）目前系统自动化分析模块中的日志分析功能是基于手工构建异常经验库实现，经验库的扩充较为困难，并且分析结果仅能反映代码层面的含义，准确性不够高。该模块保留了扩展接口，之后可以通过采用一些更加智能的分析方法进行替代升级。

（3）目前系统提供的测试配置可选项包括操作系统、CPU 和内存资源、软件依赖版本四类，后续可以考虑扩展配置可选项，并提供更多的版本可选。

（4）目前可移植性指标的分析计算较为简单，且只有四个维度，仅能提供参考，仍然需要测试人员根据报告中的详细数据进行分析和判断，后续可以考虑收集更多的数据，采用更加科学的分析计算方法，提供更加准确丰富的可移植性指标。

致　　谢

时光荏苒，如白驹过隙，研究生的生活悄然走到了尾声；回想从大三保研来到南京大学 iSE 实验室至今已近三年，在这里我度过了难忘的时光，遇到了很多优秀的人，在他们的帮助关心下，我收获了很多宝贵的知识、技能和经验，并且顺利完成了毕业设计的系统开发和论文编写。

首先我想感谢我的导师陈振宇教授，在 iSE 实验室的几年中，陈老师给予了我莫大的指导和帮助；刚进实验室时陈老师根据我的个人意愿让我进入了慕测研发组，提供了充足的平台和资源帮助我学习和进步，这段开发实践经历让我在技术上得到了充分的成长，也让我更加深入地思考了未来的职业发展方向；在毕业论文编写阶段，陈老师为我提供了思路和方向，在写作方面进行了指导和督促，提出了很多宝贵的意见。还有 iSE 实验室的房春荣老师和赵源博士，这几年我也受到了他们诸多照顾，他们都是良师，也是益友。正因为在他们的帮助和引导下，我的研究生生活才能如此充实快乐地度过，顺利完成毕业论文。其次我还要感谢慕测科技的黄勇老师，他在软件研发、职业发展和人生规划等方面分享了很多宝贵的经验，尤其是在技术上的指导，给了我很大的帮助。还要感谢研发组的门铎学长、袁阳阳学姐和韩奇学姐，在我刚进组时热情地为我答疑解惑，并在学习和生活上给了我很多照顾；感谢杨郁芩学妹在我的毕设研发过程中，为我提供了许多前端上的指导和帮助；另外还要感谢孙加辉、李文龙、段梦洋、徐佳炜等研发组的其他同学，以及我的三位室友，许子桓、杨笑和闫超威，感谢他们这几年的陪伴和帮助，让我度过了惬意快乐的校园生活；有他们一起，在学习、生活和求职等方面互相帮助和鼓励，让我的研究生生活更加多姿多彩。最后我要特别感谢我的女朋友闵轩，四年里我们从分隔两地到一起进入南京大学读研，正是彼此的关心照顾和鼓励，我才能不断进取，想要成为一个更加优秀的人；还有我的家人们，他们都是我的坚强后盾，让我不断有自信和动力去为未来奋斗拼搏。

我时常感到自己的幸运，让我遇到如此多优秀且善良的人，庆幸有他们作为我的师长、榜样、朋友。衷心祝愿所有的人，都能有幸福美好的生活和光明的未来，让我们一起努力奋斗！

参考文献

[1] JORGENSEN P C. Software testing: a craftsman's approach[J], 2018 : 3 – 5.

[2] DOĞAN S, BETIN-CAN A, GAROUSI V. Web application testing: A systematic literature review[J]. Journal of Systems and Software, 2014, 91 : 174 – 201.

[3] 查修齐, 高元钧, 吴荣泉. C/S 到 B/S 模式转换的技术研究 [J]. 计算机工程, 2014, 40(01) : 263 – 267.

[4] AL-ROUSAN T, ABUALESE H. The Importance of Process Improvement in Web-Based Projects[G] // Research Anthology on Recent Trends, Tools, and Implications of Computer Programming. Hershey, Pennsylvania, USA : IGI Global, 2021 : 1770 – 1784.

[5] YOURDON E. Java, the Web, and software development[J]. Computer, 1996, 29(8) : 25 – 30.

[6] GROUP T, OTHERS. TIOBE Index for ranking the popularity of Programming languages[J], 2020, 01 : 01 – 02.

[7] 张倩, 袁玉宇, 张炀炀. 《系统与软件可移植性》标准中可移植性定义的研究 [J]. 信息技术与标准化, 2009(10) : 50 – 54.

[8] GUPTA N, YADAV V, SINGH M. Automated regression test case generation for web application: A survey[J]. ACM Computing Surveys (CSUR), 2018, 51(4) : 1 – 25.

[9] MOONEY J D. Developing portable software[G] // Information Technology. Boston, MA : Springer, 2004 : 55 – 84.

[10] GAREN K. Software portability: Weighing options, making choices[J]. The CPA Journal, 2007, 77(11) : 10.

[11] SALONEN V. Automatic portability testing[M]. 2012 : 11 – 17.

[12] WEISSHARDT F, KETT J, ARAUJO T D F O, et al. Enhancing software portability with a testing and evaluation platform[C] // ISR/Robotik 2014; 41st International Symposium on Robotics. 2014 : 1 – 6.

[13] MALEKI N G, RAMSIN R. Agile Web development methodologies: a survey and evaluation[C] // International Conference on Software Engineering Research, Management and Applications. 2017 : 1 – 25.

[14] POOLE P C, WAITE W M. Portability and adaptabilty[G] // Software Engineering. [S.l.] : Springer, 1975 : 183 – 277.

[15] TANENBAUM A S, KLINT P, BOHM W. Guidelines for software portability[J]. Software: Practice and Experience, 1978, 8(6) : 681 – 698.

[16] SILAKOV D V, KHOROSHILOV A V. Ensuring portability of software[J]. Programming and Computer Software, 2011, 37(1) : 41 – 47.

[17] GHANDORH H, NOORWALI A, NASSIF A B, et al. A Systematic Literature Review for Software Portability Measurement: Preliminary Results[C] // Proceedings of the 2020 9th International Conference on Software and Computer Applications. 2020 : 152 – 157.

[18] TANAKA M. A study of portability problems and evaluation[C] // Proceedings Conference on Software Maintenance 1992. 1992 : 90 – 91.

[19] MOONEY J D. The CTRON approach to operating system support for software portability[J]. ACM SIGOPS operating systems review, 1992, 26(4) : 90 – 97.

[20] POULIN J S. Measuring software reusability[C] // Proceedings of 1994 3rd International Conference on Software Reuse. 1994 : 126 – 138.

[21] HAKUTA M, OHMINAMI M. A study of software portability evaluation[J]. Journal of Systems and Software, 1997, 38(2) : 145 – 154.

[22] WASHIZAKI H, YAMAMOTO H, FUKAZAWA Y. A metrics suite for measuring reusability of software components[C] // Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717). 2004 : 211 – 223.

[23] MATINLASSI M. Evaluating the portability and maintainability of software product family architecture: terminal software case study[C] // Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004). 2004 : 295 – 298.

[24] JÖNSSON P, KÅGSTRÖM S, OTHERS. Software quality attributes and tradeoffs[J]. Blekinge Institute of Technology, 2005, 97(98) : 19.

[25] GAFNI R. Framework for quality metrics in mobile-wireless information systems[J]. Interdisciplinary Journal of Information, Knowledge & Management, 2008, 3(1) : 24 – 38.

[26] DI MARTINO B, CRETELLA G, ESPOSITO A. Mapping design patterns to cloud patterns to support application portability: a preliminary study[C] // Proceedings of the 12th ACM International Conference on Computing Frontiers. 2015 : 1 – 8.

[27] DE ANGELIS F, POLINI A. Evaluation of Cloud Portability of legacy applications[C] // 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). 2018 : 232 – 237.

[28] SURYOTRISONGKO H, JAYANTO D P, TJAHYANTO A. Design and development of backend application for public complaint systems using microservice spring boot[J]. Procedia Computer Science, 2017, 124 : 736 – 743.

[29] REDDY K S P. Web Applications with Spring Boot[G] // Beginning Spring Boot 2. Apress, Berkeley, CA : Springer, 2017 : 107 – 132.

[30] GUTIERREZ F. Introduction to Spring Boot[G] // Pro Spring Boot 2. [S.l.] : Springer, 2019 : 31 – 44.

[31] 刘一田, 刘士进. 多租户高可用并行任务调度框架 [J]. 计算机系统应用, 2016, 25(12) : 280 – 284.

[32] BOETTIGER C. An introduction to Docker for reproducible research[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1) : 71 – 79.

[33] 武志学. 云计算虚拟化技术的发展与趋势 [J]. 计算机应用, 2017, 37(4): 915 – 923.

[34] RITI P. Introduction to Continuous Integration and Delivery[G] // Pro DevOps with Google Cloud Platform. Apress, Berkeley, CA : Springer, 2018 : 37 – 62.

[35] 迟学斌, 和荣, 卢莎莎, et al. 面向高性能计算环境的微服务运维平台设计与实现 [J]. 计算机应用研究, 2020, 37 : 190 – 192.

[36] ROSTANSKI M, GROCHLA K, SEMAN A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ[C] // 2014 federated conference on computer science and information systems. 2014 : 879 – 884.

[37] BARON C A, OTHERS. NoSQL key-value DBs riak and redis[J]. Database Systems Journal, 2016, 6(4) : 3 – 10.

[38] HEULE S, NUNKESSER M, HALL A. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm[C] // Proceedings of the 16th International Conference on Extending Database Technology. 2013 : 683 – 692.

[39] KRUCHTEN P B. The 4+ 1 view model of architecture[J]. IEEE software, 1995, 12(6) : 42 – 50.

[40] ZHU H, HALL P A, MAY J H. Software unit test coverage and adequacy[J]. Acm computing surveys (csur), 1997, 29(4) : 366 – 427.

[41] REICHERT A. Testing APIs protects applications and reputations[J]. Dostopno na: https://searchsoftwarequality. techtarget. com/tip/Testing-APIs-protects-applicationsand-reputations [29.7.2019], 2015.

[42] WU Q, HU Y, WANG Y. Unit testing and action-level security solution of struts web applications based on MVC[C] // 2010 International Conference on Biomedical Engineering and Computer Science. 2010 : 1 – 4.

简历与科研成果

基本信息

薛晓波，男，汉族，1997 年 2 月出生，安徽省天长市人。

教育背景

2019 年 9 月 — 2021 年 6 月 南京大学软件学院 硕士

2015 年 9 月 — 2019 年 6 月 大连海事大学信息科学技术学院 本科

攻读工程硕士学位期间完成的学术成果

1. Wen Wu, **Xiaobo Xue**, Ya Li, Peng Gu, and Jianfeng Xu. “Code Similarity Detection using AST and Textual Information”. International Journal of Performability Engineering 15, no. 10 (2019): 2683.

《学位论文出版授权书》

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

作者签名：薛骏波

2021 年 5 月 20 日

论文题名	Java Web 应用可移植性自动化测试系统的设计与实现				
研究生学号	MF1932216	所在院系	软件学院	学位年度	2021
论文级别	☐ 硕士　☑ 硕士专业学位 ☐ 博士　☐ 博士专业学位 （请在方框内画勾）				
作者 Email	xuexb@smail.nju.edu.cn				
导师姓名	陈振宇 教授				

论文涉密情况：

☑ 不保密

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

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。