南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 移动应用缺陷报告自动生成系统的设计与实现

作 者 姓 名 李沛源

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

2019 年 5 月 27 日

学　　号：**MF1732071**

论文答辩日期：**2019 年 5 月 20 日**

指 导 教 师：　　（签字）

The Design and Implementation of Automatic Generation System for Bug Report of Mobile Application

By

Peiyuan Li

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 2019

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： 移动应用缺陷报告自动生成系统的设计与实现

工程硕士（软件工程领域） 专业 2017 级硕士生姓名： 李沛源

指导教师（姓名、职称）： 陈振宇 教授

摘 要

移动应用开发周期越来越短，应用质量保障需要更高效的测试方式。自动化测试方法较传统人工测试具有速度快、可重复性强、多设备批量执行等优势，能够有效提高测试效率。然而，自动化测试日志可读性低，通常需要人工审查日志分析缺陷。受限于时间，人工分析难以充分挖掘多元日志的缺陷信息，缺乏完整的缺陷上下文，导致开发人员无法快速定位和修复缺陷。

本文设计与实现了一个移动应用缺陷报告自动生成系统，以解决自动化测试中日志可读性低与缺陷定位效率低的问题。本系统通过对自动化测试产生的多元日志进行挖掘，构建包含完整上下文信息的缺陷模型，包含操作序列、堆栈信息、缺陷截图等。利用分类算法（决策树、朴素贝叶斯等）对缺陷进行分类和去重。系统为每个缺陷推荐修复方案供开发人员参考。通过前端页面将缺陷数据可视化，生成可读性高和指导性强的缺陷报告。本系统基于Dubbo框架实现微服务架构，通过远程过程调用进行微服务交互，使用Zookeeper作为服务注册与发现中心。本系统服务端基于Spring Boot框架开发，前端基于Angular框架开发，前后端通过Nginx实现负载均衡。本系统使用MongoDB作为数据库进行数据持久化，使用Redis进行数据缓存，利用Docker将服务打包为镜像便于快速部署并使用Jenkins进行持续集成。

本系统已经部署并稳定支撑自动化测试平台的缺陷报告生成。本系统的可用性评估实验选取了8款开源移动应用在20台移动设备上分别进行自动化测试并生成缺陷报告。实验结果经过人工验证，报告平均准确率达到93%。本系统使开发人员无需直接面对自动化测试工具产生的海量日志，呈现完整的缺陷上下文信息，提高缺陷分析效率。

关键词： 移动应用自动化测试，缺陷报告，微服务，分类算法

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 Automatic Generation System for Bug Report of Mobile Application

SPECIALIZATION: Software Engineering

POSTGRADUATE: Peiyuan Li

MENTOR: Professor Zhenyu Chen

Abstract

As mobile application development cycle is getting shorter and shorter, application quality assurance requires more efficient testing methods. Compared with traditional manual testing, automated testing methods have the advantages of high speed, repeatability, and multi-device batch execution, which can effectively improve testing efficiency. However, the logs generated by automated testing have low readability and often require manual review to locate bugs. Due to time constraints, it is difficult to fully mine the bug information of multiple logs by manual analysis and lack of complete bug context, which results in developers unable to locate and fix bugs quickly.

This thesis designs and implements an automatic generation system for bug report of mobile applications to solve the problems of low readability of logs and low efficiency of bug location in automated testing. By mining multiple logs generated by automated testing, this system builds a bug model containing complete context information, including operation sequence, stack information, bug screenshots, etc. The classification and deduplication of bugs are performed using classification algorithms (Decision Trees, Naive Bayes, etc.). The system recommends a fix for each bug for developers to refer to. Bug data is visualized by front-end pages to generate highly readable and instructive bug reports. The system implements the microservices architecture based on the Dubbo framework, and the microservices interact through remote procedure calls, using Zookeeper as the service registration and discovery center. The system server is developed based on the Spring Boot framework. The front end is developed based on the Angular framework, and the front and back ends are load balanced by Nginx. The system uses MongoDB as a database for data persistence, Redis

for data caching, and Docker to package services as images for rapid deployment and continuous integration using Jenkins.

The system has deployed and stably supported the bug report generation function of the automated testing platform. The system's availability assessment experiment selected 8 open source mobile applications to perform automated testing on 20 mobile devices and generate bug reports. The experimental results have been manually verified, and the average accuracy of bug reports reaches 93%. The system eliminates the need for developers to directly face the massive logs generated by automated testing tools and improves the efficiency of bug analysis by presenting complete bug context information.

Keywords: Mobile Application Automated Testing, Bug Report, Microservice, Classification Algorithm

目　录

表　目　录

图　目　录

第一章　引言

1.1　项目背景

随着移动互联网和智能手机越来越融入人们生活中，移动应用软件也在充当越来越重要的作用，其软件质量保障需求也越来越高。传统手工测试的低效已经无法满足移动应用快速迭代中对质量保障的需求，因此自动化测试逐渐成为移动应用主流测试方式。移动应用自动化测试可以通过执行测试脚本同时在多台设备上进行测试，并可以重复执行测试脚本，减少执行过程中人为疏忽与错误，大幅提高测试效率 [1]。但是自动化测试仍存在测试日志可读性低，需要人工审核定位缺陷，以及缺乏关于缺陷的完整上下文信息，仅依靠堆栈等信息难以快速定位和修复缺陷等不足。针对上述问题，本文设计并实现移动应用缺陷报告自动生成系统，依托于公司云测试平台，在保留自动化测试优势基础上，通过对测试框架产出的多元日志进行深度挖掘，生成可读性高、指导性强的缺陷报告，提高开发人员定位缺陷、修复缺陷的效率。

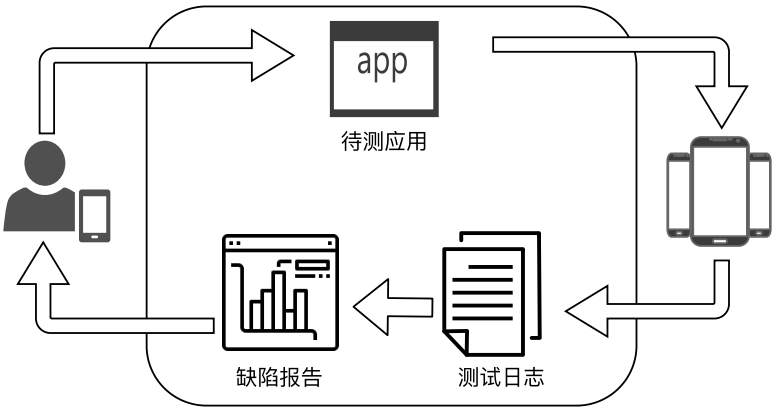

图 1.1: 移动应用缺陷报告自动生成流程

移动应用缺陷报告自动生成流程如图 1.1所示，系统依托于公司云测试平台，集成自动化测试工具，开发人员只需将待测应用安装包文件上传至云测试平台即可进行自动化测试，并可以查看系统生成的高可读性报告以快速定位应用缺陷。自动化测试过程中由系统按需分配测试资源，对待测应用进行自动化测试，将测试产出的多元日志进行数据预处理，关联缺陷发生时的页面截图、发生前后的操作序列等，构建包含完整上下文信息的缺陷模型。利用分类算法对缺陷进行分类，对短时间内大量重复出现的缺陷进行去重以防止开发人员重

复审核相同缺陷，并对每个缺陷提供修复方案供测试人员参考。通过前端页面生成可读性高、指导性强的缺陷报告。系统还提供了将自动化测试产生的缺陷转为众包测试进行验证的功能。如果测试人员对自动化测试产生的某些缺陷存在疑虑，可以选择缺陷转为云测试平台提供的众包测试，并进行在线验证。本系统使开发人员无需关心自动化测试环境与测试日志，只需查看系统生成高可读性缺陷报告即可快速定位应用缺陷。

1.2　国内（外）研究现状

1.2.1　移动应用在线自动化测试平台发展现状

移动应用研发生命周期中，测试是非常关键的环节。传统的手工测试由人工根据测试用例执行测试步骤并记录测试结果，对于机械重复的测试用例，手工测试效率低而且容易出错。自动化测试则是根据测试用例编写测试脚本，由自动化框架执行测试过程并输出测试结果。相比于手工测试，自动化测试的优势在于测试速度快且准确率高，并能够反复执行，从而节省大量的人力资源，提高测试效率。目前移动应用有着丰富的自动化测试框架，在脚本编写、处理机制、测试方法、处理流程等方面各有不同。

但是自动化测试需要编写测试脚本、设计场景、配置环境等较为复杂的准备流程，对测试人员的要求较高。为了解决这一问题，基于云测试平台的在线自动化测试解决方案应运而生 [2]。通过云测试平台，提供以测试即服务TasS（Test as a Service）为模型的基础新型软件测试模式 [3]。基于云测试共享基础架构的能力，测试人员可以通过云测试平台按需获取共享的移动应用测试软硬件资源 [4]，如不同机型的测试设备、自动化测试的环境和测试脚本等，这样既解决了手工测试效率低下的问题，又解决了将自动化测试应用在移动应用上学习成本和硬件成本高的问题 [5]。国外云测试平台Cigniti[1]终端云允许用户将自己的自动化移动测试框架应用在远程的真实终端上进行测试。Apkudo[2]为了度量终端用户体验，提供终端认证的功能,利用应用在终端上的表现来测试终端的综合性能。DeviceAnywhere[3]基于SaaS的商业模式，开发人员可以远程实时访问真实移动终端，提供功能测试、性能测试等测试服务，同时可以集成现有软件生命周期管理软件。TestDroid[4]使用了AppCrawler和Recorder 分别用于测试移动应用的兼容性和生成可重用的测试用例 [6]。

[1]https://www.cigniti.com
[2]https://apkudo.com
[3]http://www.qatestingtools.com
[4]https://cloud.testdroid.com

目前国内主流的移动应用云测试平台有腾讯WeTest[5]、阿里MQC[6]、百度MTC[7]、TestBird[8]、Testin 云测[9]等，主要提供安卓应用的功能、性能、兼容性测试等测试服务。国内主流平台的主要优势在于对机型和系统版本的覆盖性较高，但是存在共同的不足之处是对测试的结果和缺陷的分析并不深入和直观，更多的停留在展示原始报错日志的层面上，可读性不高，这也是本文着重需要解决的问题。

本文所阐述的缺陷报告自动生成系统同样提供了在线自动化测试功能，相较传统测试的优势在于用户只需要上传移动应用的安装包至测试平台，并选择需要执行测试的移动设备即可进行自动化测试，将复杂的自动化测试环境和配置进行封装，并提供了测试任务的一站式管理。通过简单的操作便可以将应用同时在多个机型上进行自动化测试，能够反复执行测试脚本，同时利用云测试平台共享测试设备，降低自动化测试成本。

1.2.2　移动应用测试缺陷报告发展现状

移动应用的开发人员通常依靠两种来源查看应用的缺陷情况，一种是来自结构松散的自然语言，如应用论坛中的用户评论，另一种是来自日志中应用报错的堆栈信息，缺陷上下文信息匮乏是开发人员无法复现或修复缺陷的主要原因 [7]。依据Zimmermann等人对对软件开发人员的调查，已经证明只有大约50％的开发人员认为缺陷报告是足够完整的，其中操作序列与截图等信息在开发人员复现缺陷时与缺陷堆栈信息同样重要，开发人员需要耗费较多时间用于人工书写缺陷报告 [8]。因此本系统旨在自动生成拥有缺陷完整上下文信息的缺陷报告。

为了解决应用开发人员复现和修复缺陷中的难题，使开发者能够有效地复现缺陷报告的操作过程，Kevin Moran提出了ODBR（On-Device Bug Reporting）工具 [9]，该工具能够在独立的Android设备上运行。为了准确描述Android应用程序测试中发生的缺陷，该工具利用Uiautomator 框架和底层事件流捕获来录制和重放缺陷发生时的一系列输入手势和传感器事件。这些记录信息会发送到Java Web应用程序，显示详细且可操作的缺陷报告，其中包括屏幕截图以及可在目标设备上重放的一系列用户事件，这种精确的收集方法可以捕获开发人员复现和修复缺陷所需关键信息。

[5]https://wetest.qq.com
[6]https://mqc.aliyun.com
[7]http://mtc.baidu.com
[8]https://www.testbird.com
[9]https://testin.cn

CRASHSCOPE作为一种新型自动化工具 [10]，将待测应用使用系统生成的测试算法进行测试,生成使用自然语言描述具有明确复现步骤的缺陷报告。此工具仅需要待测应用安装包文件和Android设备即可运行。整个CRASHSCOPE工作流程完全自动化，开发人员仅需阅读工具生成的缺陷报告。系统执行不同测试策略，旨在检测Android应用程序的缺陷并生成缺陷报告，其中包含屏幕截图，详细的缺陷复现步骤，堆栈信息以及可自动复现缺陷的可重放脚本。

基于上述调研，本文阐述的缺陷报告自动生成系统与ODBR工具和CRASHSCOPE工具思路一致，均致力于将缺陷前后的操作序列、截图日志等缺陷上下文信息通过缺陷报告呈现给开发人员，提高其复现和修复缺陷的效率。除此之外，本系统还提供基于测试设备机群的自动化测试，同一应用可以同时在拥有不同系统版本、分辨率、终端品牌的测试设备机群中同时进行测试，对捕获的缺陷进行分类和去重，避免开发人员需要重复审核相同缺陷，并呈现各类缺陷在不同类型设备的分布情况。

1.3　本文的主要研究的工作

本文阐述如何设计并实现移动应用缺陷报告自动生成系统。本系统在集成公司自研移动应用自动化测试工具基础上，通过云测试平台提供在线自动化测试功能，利用缺陷报告生成服务深度挖掘自动化测试产出的多元日志，构建具有完整上下文信息的缺陷并通过生成高可读性缺陷报告进行数据可视化。系统使用基于Dubbo框架的微服务架构，将缺陷报告生成服务核心业务与云测试平台业务进行解耦，两者均作为独立微服务通过远程过程调用（Remote Procedure Call，RPC）进行交互。使用微服务架构便于各服务独立测试和部署，通过Zookeeper负责服务注册与发现，云测试平台作为服务调用方调用缺陷报告生成服务的接口时不需要硬编码服务IP 地址，降低系统维护成本。

移动应用缺陷报告自动生成系统分为测试任务创建与分发、缺陷分类器建模、日志解析与缺陷构建、缺陷报告前端渲染四个模块，其中任务创建与分发以及缺陷报告前端渲染模块主要逻辑由云测试平台服务端负责实现，缺陷分类器建模和日志解析与缺陷构建模块主要逻辑由缺陷报告生成服务负责实现。

测试任务创建与分发模块中，为了解决测试设备与测试任务的多状态匹配问题，采用状态机设计模式维护测试设备与测试任务的状态，根据状态转换执行相应操作，状态匹配时将任务分发自动化测试工具。

缺陷报告生成服务中，将自动化测试工具中产出的多种日志（应用日志、操作日志、截图日志等）进行关联，解析出缺陷发生时完整的上下文信息，提高了缺陷验证时复现缺陷和定位缺陷的效率。通过对缺陷建模，利用监督式分

类算法（决策树与朴素贝叶斯算法）对缺陷进行分类，基于已标注的训练集构建分类器模型，从而对缺陷的类型进行预测。对于每一类缺陷提出缺陷修复方案供开发人员参考，使测试报告具备更高的指导价值。为了丰富系统的缺陷修复方案库和缺陷训练集，提供了缺陷认证的功能，开发人员可以通过对系统推荐方案点赞或提交个人对该缺陷分类的方式将缺陷进行标注，并填写缺陷发生原因和修复方案，审核通过后将新的缺陷加入训练集，将修复方案加入系统缺陷修复方案库。系统经过缺陷报告生成服务生成缺陷报告所需数据，由前端Web 页面对解析数据可视化，生成可读性较高、便于开发人员定位缺陷的的高质量缺陷报告。测试报告可以进行编辑交互，用户可以选择报告中的缺陷转为众包测试在云平台进行在线真机验证。

云测试平台服务端与缺陷报告生成服务均基于Spring Boot框架进行开发，同时结合Maven和Docker进行项目的快速构建和可移植性部署。缺陷报告的前端渲染基于Angular框架进行开发，有效地将渲染和交互逻辑进行分离，并结合Echarts生成直观的视图对分类结果等数据进行可视化。

1.4　本文的组织结构

本文的组织结构如下：

第一章 引言部分。介绍了移动应用缺陷报告自动生成系统的项目背景，对比了云测试平台的国内外发展现状，并阐述了本文的主要研究工作。

第二章 技术综述。将本系统实现过程中使用的技术、算法和框架的优势和用途进行简要的介绍。

第三章 系统需求分析与概要设计。对项目的功能性需求和非功能性需求结合需求列表与系统用例图进行说明。对系统的总体设计、数据结构设计、模块设计及模块间的交互设计等结合图表的方式进行阐述。

第四章 系统详细设计与实现。对测试任务创建与分发、缺陷分类器建模、日志解析与缺陷构建、缺陷报告前端渲染四个模块的设计与实现进行了详细的阐述，主要包含了各模块的流程设计、数据与类设计、算法评估、实现代码、界面截图等方面。

第五章 总结与展望。总结论文主要工作，并展望了移动应用缺陷报告自动生成系统的未来可扩展方向。

第二章　技术综述

2.1　Dubbo

Dubbo是阿里巴巴公司开源的一款优秀的分布式服务框架，采用透明化和高性能的RPC远程服务调用方案，实现系统调用远程方法就像调用本地方法一样简单的效果，与Spring Boot框架集成也十分方便。Dubbo框架架构图如图2.1所示。其中较为核心的三个角色分别为Provider （服务提供方），Consumer（服务消费方）以及Registry （注册中心）。

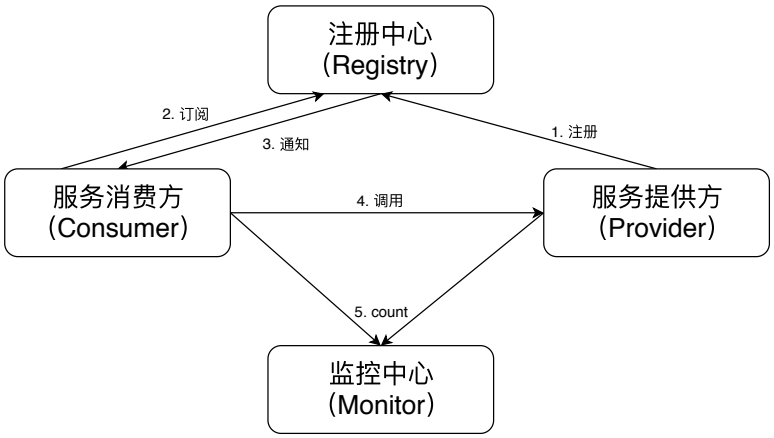

图 2.1: Dubbo框架架构图

工作原理为：定义好统一的服务接口，并由服务消费方和提供方均引入依赖。服务提供方负责实现定义的统一接口，启动成功时，向注册中心注册对外提供的服务。服务消费方启动成功时向注册中心订阅自己需要调用的远程服务，由注册中心负责将服务提供方地址列表给服务消费方，一旦提供方有变更，注册中心将基于长连接推送变更数据给消费方 [11]。服务消费方得到服务提供方的地址列表后，利用软负载均衡算法选择一台提供者进行服务调用，若有多个服务提供方，调用第一个失败后会自动选择其他提供方进行调用 [12]。

Dubbo官方文档中推荐使用Zookeeper作为服务注册与发现的注册中心。Zookeeper是一款在分布式中处理一致性的开源框架。ZooKeeper具有负载均衡的特性 [13]，单独一个注册中心可能无法承载过高的流量，负载均衡就是在流量到达超过阈值时进行分流而存在的，Web应用通过与ZooKeeper 群配合就能够达到负载均衡。

本文所阐述的系统采用Dubbo框架主要基于以下优势：

（1）本系统缺陷报告生成服务与云测试平台服务端分别作为服务的提供方和消费方，基于Dubbo框架中注册中心提供的服务注册与发现功能，云测试平台调用缺陷报告生成服务的接口不再需要维护所有的服务实例对应的IP，调用缺陷报告生成服务的方法就像调用本地方法一样简单，通过远程过程调用透明化降低维护成本。

（2）Dubbo框架对Spring Boot支持较好，由于本系统后端均基于Spring Boot框架进行发开，集成Dubbo只增加简单的配置和注解，如服务提供者只需要在对外提供的服务上添加@Service注解，服务消费者在调用的远程服务添加@Reference注解即可，无需添加侵入式的API代码。

（3）提供软负载均衡及容错机制，减少单点，提高系统的稳定性 [14]。通过将复杂的单体应用分解为多个微服务，每个服务都具有明确的边界 [15]，通过RPC接口相互交互，每个微服务独立测试部署，便于开发和维护 [16]。

2.2　Spring Boot

Spring Boot是由Pivotal团队开发的框架，其设计目的在于简化Spring框架的搭建与开发过程，其核心就是自动配置 [17]。作为一款十分适用于构建微服务的基础框架，Spring Boot具有以下优势：

（1）为了满足开发者快速构建应用的需求，Spring Boot提供自动化配置，极大地简化了Spring原本样本化的复杂配置 [18]。

（2）支持嵌入式的Tomcat容器，可以直接通过简单的java -jar命令启动标准化的web应用，并提供了运行时的监控，极大地提高了开发和部署效率。

（3）将Spring相关的技术解决方案通过一系列Starter POMs依赖包进行整合，集成模块时不用再维护Maven的pom.xml中错综复杂的依赖关系 [19]。

（4）完整地支持和封装了Spring生态圈及其他工具链的整合（如本系统使用的Redis，Dubbo，MongoDB、Log4J2等）。

由于本系统的云测试平台服务端及缺陷报告生成服务均作为单独微服务，并需要集成Dubbo框架、访问MongoDB数据库、访问Redis缓存等依赖，Spring Boot均具有良好的支持，故采用Spring Boot框架进行开发。

2.3　Redis

Redis是一个基于ANSI C语言编写的既可基于内存亦可持久化的Key-Value、日志型数据库 [20]。由于Redis还可以提供String，List，Map，Sets等丰富数据

结构的存储，故也被称为数据结构服务器。本系统基于以下优势选用Redis：

（1）由于Redis直接在内存中运行，因此具有优异的读写性能，读写速度能够达到10万次/秒，因此首要的应用场景便是作为数据缓存组件。

（2）Redis所有操作都是原子性的，所有数据类型都是基于基本数据结构并对使用者透明，无需额外的抽象。

（3）Redis支持通知，发布/订阅，设置过期时间等特性，自动过期能够自动清理过期数据无需使用者主动操作，提高开发效率 [21]。

（4）提供Master-Slave配置，快速的主从复制确保了发生服务故障时便于及时切换服务，提高系统稳定性。

本文阐述的移动应用缺陷报告自动生成系统便利用Redis作为缓存中间件，其中缺陷分类器模型作为热点数据以Key-Value形式存储在Redis中。由于每个自动化测试任务产生的日志都需要进行解析并进行缺陷的分类，分类时需要先载入缺陷分类器模型，所以将模型放入Redis缓存中可以极大地提高模型载入的效率和日志解析的速度。在载入分类器模型时首先读取Redis，如果不存在再从数据库中读取并将模型存入Redis。更新分类器模型时，存入数据库更新至缓存，并将之前的模型缓存设为过期从而确保缓存中总是最新的分类器模型。

2.4 MongoDB

MongoDB是使用C++语言编写的一款开源的，高性能，模式自由的非关系型数据库。MongoDB是面向文档的，可以方便的管理类似JSON的文档集合。相比于关系型数据库中面向关系的表结构，文档模型与面向对象的数据表达方式更加相似，文档中嵌入子文档就像Java对象的成员变量一样。数据可以被嵌套到复杂的体系中并保持查询和索引能力，因此应用程序中能够以更加自然的方式来为数据建模 [22]。

除此之外，MongoDB还有模式自由的特性。MongoDB 没有Schema（模式、数据模型），数据库并不需要知道将要存入到集合中的文档的任何结构信息。基于关系型数据库进行持久化时，如果需要给程序中某个对象增加新的属性，需要变更从应用数据层到数据库表的一套流程 [23]。尽管可以利用工具将该过程进行自动化，这仍然是一个复杂的运维问题，更新生产环境的线上数据库会更加棘手。而如果基于MongoDB进行数据持久化，就无需对数据库进行变更，只用在应用层做必要的改动即可，极大地降低了系统运维成本。

MongoDB支持完全索引，包含内部对象的索引，支持丰富的查询表达式。通过使用JSON形式的查询指令，能够便捷地对文档中内嵌对象及数组进行查

询。MongoDB还拥有在运维和部署方面的优势，分片功能支持在一台机器不够存放数据时，能够在集群中进行分散存储，同时利用在各个结点间自动迁移数据的功能，分片可以使各节点的数据量保持均衡。使用复制集可以用冗余机器的成本来降低丢失数据的代价。

本系统中定义的缺陷模型是多层嵌套的，使用MongoDB数据库相较关系型数据库能够更加便捷地进行存取。服务端的Spring Boot框架通过在Maven项目的pom文件中引入sping-data-mongodb依赖即可实现对MongoDB 数据库的访问，由于缺陷报告生成系统的服务端均基于Spring Boot进行开发，所以能够很方便进行MongoDB数据库访问的代码开发。

2.5 Angular

Angular是Google公司维护的一个基于TypeScript语言开发的开源前端框架，符合MVVM（Model-View-ViewModel）设计模式，具有双向数据绑定、依赖注入、声明式模板、路由、模块化等特性 [24]。微软架构师John Gossman在2005年首次提出MVVM模式 [25], 其核心为提出视图模型（ViewModel），对视图（View）与模型（Model）进行解耦，避免因为变动View而重构其对应的Model产生的昂贵代价。ViewModel将Model数据再封装，对View 同步更新并响应其执行的操作，进行相应处理后通知Model层进行更改。

缺陷报告自动生成系统基于以下优势采用Angular框架用于前端项目开发：

（1）符合高内聚，低耦合以及高复用性的软件设计原则。Angular使用MVVM架构模式，其核心机制为通过双向绑定维护View与ViewModel中数据的一致性，View可以独立于Model变化和修改。视图逻辑封装于ViewModel中，同一个ViewModel可以被多个View复用。

（2）良好的团队支持。开发流程中界面设计（View）与业务逻辑和数据开发（ViewModel）进行解耦，均可以独立开发。同时Angular的模块化设计将项目分为多个功能模块，每个模块聚焦于一个特性区域，包含相应的模板、组件、指令等，便于团队成员分工同时进行各自模块的开发。

（3）丰富的组件库支持。Angular框架具有丰富的开源组件库支持，如PrimeNG[1]、NG Bootstrap[2]、NG-ZORRO[3]、Material2[4] 等，引入相应组件库依赖即按照项目需求进行个性化定制与开发，极大提高前端页面开发效率。

[1]https://github.com/primefaces/primeng
[2]https://github.com/ng-bootstrap/ng-bootstrap
[3]https://github.com/NG-ZORRO/ng-zorro-antd
[4]https://github.com/angular/material2

本系统的前端项目中将获取服务端数据的逻辑代码封装至自定义的服务类中，使用Angular提供的HttpClient类实现HTTP客户端功能，通过依赖注入的方式将服务类注入到组件中为组件提供数据，从而使Angular前端框架可以调用服务端接口进行数据获取。本系统前端项目部署于Nginx服务器中，通过Nginx 实现负载均衡与反向代理。前端项目通过打包命令（ng build）生成打包后的文件存于本项目dist文件夹中，将打包文件拷贝至Nginx服务器下，启动Nginx 服务器即可实现前端项目的部署。

2.6 Docker

Docker遵从Apache2.0协议的是一个开源的应用容器引擎，使用Go语言进行开发实现，属于Linux容器的一种封装，提供简单易用的容器使用接口，是目前最流行的Linux容器解决方案。

开发者的应用及其相关的依赖包均可以通过Docker打包到轻量级、可移植的容器中，并发布到各发行版本的Linux 服务器上。作为一种新兴的虚拟化方式，Docker与传统的虚拟化方式相比具有许多优势，与通过Hypervisor把底层设备虚拟化的虚拟机不同，Docker直接移植于Linux内核之上，利用LXC（Linux Container）来实现类似虚拟机的功能，内核命名空间来实现容器的隔离性，利用控制组实现资源的可度量及可配额，利用联合文件系统实现移动性 [26]。由于Docker通过Linux进程虚拟隔离底层设备，相比于虚拟机的系统性能损耗要低很多。同时，Docker应用容器具有非常高效的启停效率，能够支持大规模分布系统水平扩展。基于上述优点，Docker适用以下场景：

（1）在CI（持续集成）/CD（持续部署）环境中提供项目镜像构建和快速部署，连通生产环境与测试环境，保持各环境间高度一致性的功能。

（2）在微服务架构中作为独立服务的部署单元，Docker镜像中封装了该服务需要的所有依赖，简化环境配置，便于快速自动化打包以及部署发布。

（3）基于Docker容器随开随关的特性，能够提供弹性的云服务，支持动态扩容和缩容，提高系统可扩展性。

本文阐述的移动应用缺陷报告自动生成系统中云测试平台与缺陷报告生成服务均作为独立的微服务。各微服务通过Maven对项目源代码进行可执行文件打包后，利用DockerFile进行镜像打包，上传容器镜像至DockerHub用于管理镜像。生成Docker镜像后可以在不同的生产环境中进行快速打包部署，便于后续的持续集成 [27]，部署服务时可以直接从DockerHub拉取最新版本镜像进行部署，提高系统打包及部署速度，加速开发生命周期。

2.7 Jenkins

Jenkins是一款基于Java实现用于持续集成的开源可视化Web工具，可以使开发人员从繁杂的集成工作中解脱出来，能够集中更多的精力用于实现项目的业务逻辑，提高开发效率。持续集成的概念由Martin Fowler 提出 [28]，要求对软件项目进行持续化地自动化编译、打包、测试和发布，旨在尽早发现集成中的问题，及时反馈，提高自动化程度以提高软件开发效率。

本系统采用Jenkins作为持续集成工具，配置构建脚本，集成系统的代码版本管理工具GitLab，并结合上述Docker容器实现本项目各服务的自动化部署。当开发人员推送项目最新代码至GitLab远程仓库后，Jenkins 服务器在指定的仓库和分支上Hook代码更新操作，将代码下载至Jenkins服务器。Jenkins通过预先设置的任务依次进行可执行文件打包、容器镜像打包、上传容器镜像至DockerHub服务器等任务。发布时部署服务器从DockerHub拉取新版本镜像，删除原来的容器后重新运行新版本镜像。同时Jenkins能够对集成中存在的错误进行实时监控，具备详细的日志文件和提醒功能，并将项目构建趋势和稳定性通过图表形式形象地展示出来。

由于本系统所涉及的云测试平台是由团队中多人进行共同开发，为了更好的协同工作并确保软件开发的质量，采用持续集成这一软件开发实践，团队开发成员必须经常利用Jenkins集成自己的工作以便尽早发现和解决问题。

2.8 分类算法

分类算法是数据挖掘中广泛使用的技术，分类的目的为根据已有数据集构建分类模型，该模型能够将未知类别的新样本映射至某一类别，实现对样本类别的预测，属于监督学习。常用的分类算法有决策树算法、朴素贝叶斯算法以及支持向量机等，考虑到决策树和朴素贝叶斯算法的模型简洁，分类速度快，分类准确率高等优势，本系统采用这两种分类算法构造分类器模型用于对缺陷进行分类。缺陷报告生成服务基于抽象工厂方法的设计模式提供不同分类器，提高可扩展性，符合软件设计的开闭原则。本系统提供对决策树分类器和朴素贝叶斯分类器的分类效果评估对比，自动选择分类评估数据较优的分类器。

2.8.1 C4.5决策树算法介绍

决策树分类算法的目标是将具有p维特征的n个样本分到c个类别中去，将样本经过一种变换赋予一种类别标签。决策树作为一种分类预测模型，由决策节点、分支和叶节点三部分构成。决策节点代表待分类样本某个属性，每个分支

代表在该属性上的测试结果，是决策节点的不同取值。类别标签存放在叶节点中，代表一种分类结果 [29]，不同的决策树算法会根据不同特征选择方案进行分支选择。

本系统使用C4.5决策树算法构造缺陷分类器，该算法是用于生成决策树的一种经典算法，属于监督学习，由Quinlan于1993提出 [30]。使用流程为：基于已标注类别的缺陷训练数据集，每一个缺陷属于一组类别中的某一类。根据训练集构建决策树分类器模型，该模型描述从缺陷的属性值到类别的映射关系，并且能够用于对类型未知的新缺陷进行类别预测。分裂属性选择的评判标准是决策树算法之间的根本区别。C4.5 算法由ID3算法改进而来，ID3算法由Quinlan于1987年提出 [31] ，区别于ID3算法通过信息增益选择分裂属性，C4.5算法通过信息增益率选择分裂属性，属性取值数目越大，分裂信息值越大，从而部分抵消了属性取值数目带来的影响，因此可以避免ID3算法中通过信息增益倾向于选择具有大量值的属性作为分裂属性的不足 [32]。除此之外，该算法还具有能够处理离散型和连续型的属性类型，对生成树减枝，降低过拟合等优势 [33]。信息增益率计算方法如表 2.1 所示。

表 2.1: C4.5决策树算法关键公式

公式名称	公式
信息熵	$info(D) = -\sum_{i=1}^{m} p_i log_2(p_i)$
属性A的分裂信息	$splitInfo_A(D) = -\sum_{j=1}^{v} \frac{\|D_j\|}{\|D\|} * log_2 \frac{\|D_j\|}{\|D\|}$
信息增益	$infoGain(D, A) = info(D) - info_A(D)$
信息增益率	$infoGainRation(D, A) = \frac{infoGain(D, A)}{splitInfo_A(D)}$

2.8.2　朴素贝叶斯算法介绍

朴素贝叶斯分类算法是基于概率模型的分类算法，由Maron于1960年提出 [34]，Lewis于1998年阐述了朴素贝叶斯算法在文本分类与信息检索领域的应用 [35]。朴素贝叶斯算法基于位置独立性和条件独立性两个基本假设，这也是成为“朴素”的原因。朴素贝叶斯分类算法基于贝叶斯定理：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

其中$P(A)$为A的“先验概率”，即在B事件发生前对A事件概率的一个判断。$P(A|B)$为A的“后验概率”，即在B事件发生后对A事件概率的重新评估。$\frac{P(B|A)}{P(B)}$为“标准相似度，可以作为一个调整因子，使预估概率更接近真实概率。故朴素贝叶斯定理可以表述为后验概率=标准相似度*先验概率。

应用朴素贝叶斯定理的过程可以理解为先预估一个“先验概率”，然后加入实验结果，实验结果会增强或者削弱“先验概率”，使其数值不断更新至接近事实的“后验概率”。朴素贝叶斯定理用于分类算法的思路为求解出待分类项出现的条件下各个类别出现的概率，则待分类项属于概率最大的类别。

朴素贝叶斯分类算法的主要优点是源于古典数学理论，分类效率较为稳定，对小规模的数据表现很好，适合增量式训练，并能够处理多分类任务 [36]且常用于文本分类。缺点是由其分类的前提是假设属性之间相互独立，在实际应用中，如果属性之间的相关性较大时，分类效果则会不理想 [37]。本系统将朴素贝叶斯算法用于缺陷详情的文本分类，且在使用算法前会对分类文本进行分词、添加停用词、文本空间向量化等预处理，处理后各属性间相关性较低，故具有较好的分类效果。

2.9　本章小结

本章介绍了项目中使用的相关技术、算法与框架。Spring Boot作为服务端的开发框架，集成Dubbo框架并使用Zookeeper作为注册中心后构成微服务架构。缺陷报告生成服务中使用C4.5决策树或朴素贝叶斯算法构造缺陷分类器模型，建模后使用Redis缓存提高模型的读写速度，利用分类器模型对缺陷进行分类并使用MongoDB进行数据持久化。缺陷报告的前端页面通过Angular框架进行渲染。独立的微服务使用Docker镜像部署并使用Jenkins进行持续集成。

第三章　移动应用缺陷报告自动生成系统需求分析与概要设计

3.1　项目整体概述

随着各类移动应用在人们生活中充当越来越重要的作用，用户对应用质量保障的需求也越来越高。开发人员需要在移动应用开发周期越来越短的情况下保障产品质量，传统的人工测试已经无法承受这样负荷 [38]。本文阐述的移动应用缺陷报告自动生成系统依托于云测试平台，在提供自动化测试的基础上，针对自动化测试日志可读性低，人工填写缺陷报告效率低下，缺陷上下文信息匮乏等问题，通过自动生成缺陷报告以提高开发人员定位和修复缺陷的效率，更好地支撑公司云测试平台的在线自动化测试功能。

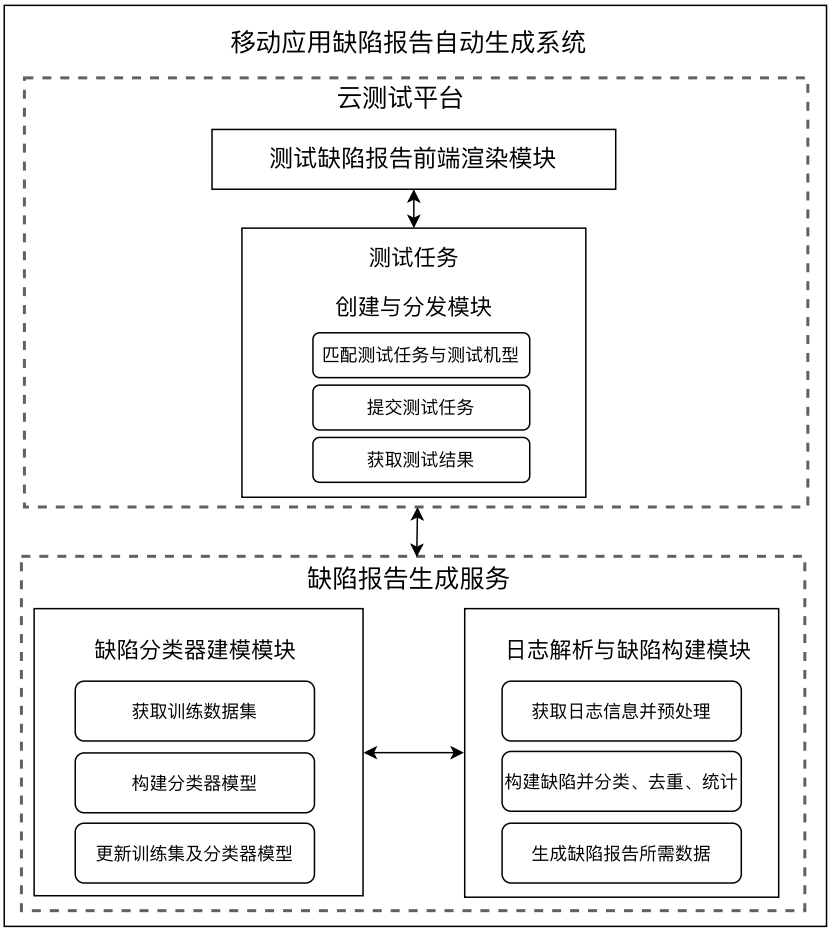

图 3.1: 移动应用缺陷报告自动生成系统模块图

本系统的模块图如图 3.1所示，依据业务流程和功能需求分为四个模块，分别为测试任务创建与分发、缺陷分类器建模、日志解析与缺陷构建和测试报告前端渲染模块。系统整体采用基于Dubbo的微服务架构，云测试平台服务端和缺陷报告生成服务分别作为服务消费方和服务提供方通过RPC接口进行交互。

移动应用自动化测试任务创建与分发模块，负责用户创建并提交移动应用自动化测试任务，平台管理员对测试任务进行审核，并负责通过审核的任务与空闲的测试机型进行状态匹配，匹配成功后转发测试任务给自动化测试工具。本模块中使用状态机的设计模式管理测试任务与测试设备的状态，将状态转移的代码从业务逻辑代码中解耦。

缺陷分类器建模模块，负责基于缺陷训练集进行缺陷分类器模型的训练，便于后续利用分类器对从测试日志中解析出的缺陷进行分类统计。目前提供了基于C4.5决策树算法和朴素贝叶斯算法的分类器构造服务，并提供了分类器分类效果的评估，默认会选择评估数据较好的分类器模型，将训练完成分类器模型存于Redis缓存中提高模型载入速度。

日志解析与缺陷构建模块负责从自动化测试工具获取应用日志、操作日志、界面截图等各类日志信息，通过解析日志构建具有完整上下文信息的缺陷模型，利用缺陷分类器模型对缺陷分类，将短时间内大量重复出现的缺陷去重，并生成缺陷报告需要的所有数据。此模块主体逻辑由缺陷报告生成服务实现，提供RPC接口给云测试平台获取缺陷报告相关数据。

缺陷报告前端渲染模块负责将云测试平台服务端获取的日志解析数据进行可视化，通过Angular框架构造前端页面渲染出测试任务的缺陷结果报告。并在报告页面中提供了将缺陷转为众包测试的功能，测试人员可以通过众包在线验证的功能对产生的缺陷进行二次验证。

3.2　移动应用缺陷报告自动生成系统需求分析

移动应用缺陷报告自动生成系统是在云平台原有功能基础上的完善和升级，本文重点对系统内在线自动化测试任务创建与分发、日志解析与测试报告的生成、将报告中缺陷转为众包测试等相关功能进行阐述。

3.2.1　功能性需求

移动应用缺陷报告自动生成系统创建与分发自动化测试任务的功能性需求列表如表 3.1所示。云测试平台的前端和服务端负责实现创建、审核、执行和停止自动化测试任务，将任务拆分后封装为数据传输对象由缺陷报告生成服务转发给自动化测试工具执行任务。

表 3.1: 测试任务创建与分发功能性需求列表

需求编号	需求名称	需求内容
R1	创建移动应用自动化测试任务	1. 普通用户创建新的移动应用自动化测试任务，上传后缀名为“.apk”的移动应用安装包，或选取已经上传的移动应用； 2. 选择自动化测试服务类型，并补充选择测试机型，填写测试名称及测试描述等配置，提交测试。
R2	审核移动应用自动化测试任务	1. 平台管理员对普通用户创建的所有移动应用自动化测试任务进行管理； 2. 可以进入任务详情页面查看并进行审核，判断是否允许执行自动化测试。
R3	执行已通过审核的移动应用自动化测试任务	1. 日志解析模块的服务不断向云测试平台更新机柜中的设备最新状态； 2. 平台管理员点击执行测试任务后，当测试任务中所需要的测试机型在机柜中为空闲可用状态时，匹配成功，测试机型状态为锁定，测试任务状态更新为匹配成功，并进入等待执行的队列。
R4	停止自动化测试任务	1. 平台管理员点击停止后可以对正在进行自动化测试的任务进行终止。
R5	轮询并提交匹配成功的移动应用自动化测试任务	1. 缺陷报告生成服务轮询地从云测试平台拉取状态为匹配成功的待执行任务； 2. 将获取到的任务信息存储到数据库后交由自动化测试工具进行测试。

表 3.2: 日志解析与缺陷构建功能性需求列表

需求编号	需求名称	需求内容
R6	获取自动化测试结果日志，并更新测试任务状态	1. 自动化测试工具将测试后的多种结果日志上传至缺陷报告生成服务所在服务器； 2. 全部上传完成后更新任务状态为已完成。
R7	通过训练集构建缺陷分类器	1. 缺陷报告生成服务依据现有数据集进行训练，获得分类器模型； 2. 默认选择评估参数较优的分类器，也提供用户自己选择的接口。
R8	从结果日志中根据缺陷模型提取结构化对象	1. 从多种测试结果日志的字符串信息中提取出结构化的缺陷对象； 2. 关联测试过程中页面截图。
R9	对缺陷进行分类、去重、统计	1. 利用构建好的分类器模型对缺陷进行分类，并利用文本相似度对缺陷进行去重； 2. 统计后对同一个测试任务中的多种机型产生的缺陷进行比对，将分析结果存入到数据库中。
R10	查看自动化测试任务的进度	1. 用户可以通过云测试平台查看当前测试任务进度。

移动应用缺陷报告自动生成系统测试日志解析与缺陷构建的功能性需求列表如表 3.2所示。主要包含从自动化测试工具获取结果日志、日志的预处理和解析、缺陷分类器的建模、缺陷的分类和去重等功能性需求，主要逻辑由缺陷报告生成服务负责实现，并将解析结果通过Dubbo 接口提供给云测试平台调用获取。用户可以通过云测试平台前端查看测试任务实时进展。

表 3.3: 测试缺陷报告交互功能性需求列表

需求编号	需求名称	需求内容
R11	云测试平台查看测试报告	1. 普通用户查看已完成测试任务的测试报告，云测试平台获取缺陷报告生成服务的结果； 2. 将结果交由前端页面进行测试报告的渲染。
R12	选择测试报告中的缺陷转为众包测试	1. 普通用户在查看测试报告后，可以进行批量的选择报告中展示的缺陷转为众包测试； 2. 其他普通用户可以作为测试人员参与众包测试对缺陷进行在线验证。
R13	认证为缺陷并提交至平台管理员审核	1. 普通用户对缺陷进行验证后，如果认证为缺陷，填写该缺陷的分类、产生原因以及解决方案，交由管理员进行审核。
R14	审核用户提交的缺陷，并重新构建缺陷分类器模型	1. 平台管理员对用户提交的缺陷进行审核，通过后由缺陷报告生成服务添加至缺陷的训练集，并重新构建缺陷分类器模型。

测试缺陷报告交互的功能性需求列表如表 3.3所示。云测试平台服务端从缺陷报告生成服务获取测试结果后，由云测试平台前端进行数据可视化，为用户提供高可读性的缺陷报告，并允许用户进行转为众包测试验证、为缺陷提供修复意见等交互功能。

3.2.2　非功能性需求

移动应用缺陷报告自动生成系统涉及的非功能性需求如表 3.4所示。表中的每项非功能性需求对系统质量的要求进行了详细的描述。其中系统安全性主要针对不同用户角色的权限问题，本系统采用集成Shiro安全框架的方式解决该问题，Shiro提供了授权、认证、会话管理等功能，将用户账号与角色多对多关联，每个角色赋予相应的权限。系统集成Log4J2日志框架，记录系统运行关键部分日志，便于快速定位缺陷。通过Spring Boot事务机制确保每一步操作的可靠性，发生异常的操作即时进行回滚。前端项目基于Angular框架开发提供合理的界面布局与流程指引，将复杂的测试流程进行黑盒处理，简化用户操作，使用户能够快速熟练使用系统。

表 3.4: 非功能性需求列表

安全性	自动化测试中，普通用户只能看到自己发布的测试任务，平台管理员可以看到所有任务，并且只有管理员拥有审核任务的权限。
	众包测试中，普通用户只能够看到自己发布或参加的任务详情，平台管理员可以看到所有任务。
可靠性	系统日志记录关键信息，系统出现故障可以通过日志快速定位缺陷。
	系统出现故障时保证数据完整性。
易用性	合理的界面布局与流程指引设计使用户能够快速熟悉并使用系统。
	将自动化测试封装为黑盒功能，用户只需要阅读处理后高可读性测试报告。

3.2.3　系统用例

本文阐述的移动应用缺陷报告自动生成系统的系统用例图如图 3.2所示。系统中主要涉及的角色有两个，分别为普通用户和平台管理员。普通用户拥有创建移动应用自动化测试任务、查看测试结果报告、通过选择报告中的缺陷转为众包测试、为已上传至云平台的应用添加用于功能回放测试的测试用例等权限。系统管理员除了拥有上述权限外，还拥有审核、执行、停止移动应用自动化测试任务的权限，只有系统管理员审核通过了该测试任务，测试任务才能继续执行下去。

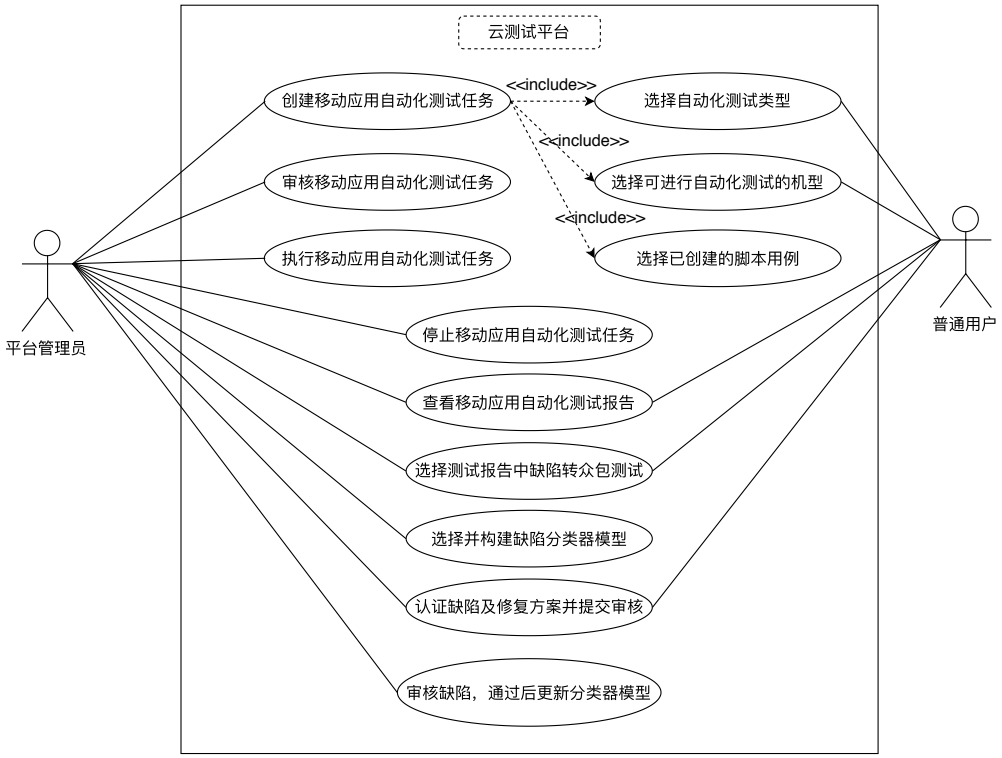

图 3.2: 系统用例图

系统用例表如表 3.5所示，为每一个用例添加了用例编号以及对应的用例名称，并关联了每个用例所涉及到的功能性需求的编号。

表 3.5: 系统用例列表

用例编号	用例名称	功能需求编号
UC1	创建移动应用自动化测试任务	R1
UC2	审核移动应用自动化测试任务	R2
UC3	执行移动应用自动化测试任务	R3、R5
UC4	停止移动应用自动化测试任务	R4
UC5	查看移动应用自动化测试报告	R6、R8、R9、R10、R11
UC6	选择测试报告中的缺陷转为众包测试	R12
UC7	选择并构建缺陷分类器模型	R7
UC8	认证缺陷类别及修复方案并提交审核	R13
UC9	审核缺陷，通过后更新分类器模型	R14

用例UC1：创建移动应用自动化测试任务的用例如表 3.6所示，该用例中主要涉及到普通用户和平台管理员的参与，是整个自动化测试流程的开始。两种角色的参与者均具有创建任务的权限，上传待测试移动应用安装包文件（apk文件）后填写相应配置即可创建成功，填写的配置内容主要包含了选择进行测试的机型，项目名称，执行时间等信息，如果创建功能回放测试还需要上传用于回放的测试脚本至云测试平台，便于自动化测试工具进行脚本回放。任务创建成功后在云测试平台后端即创建了状态为“待审核”的测试任务（Order）。

表 3.6: 创建移动应用自动化测试任务用例表

用例编号	UC1
用例名称	创建移动应用自动化测试任务
参与者	普通用户、平台管理员
前置条件	使用有效的管理员或普通用户账号成功登录系统。
后置条件	1. 普通用户可以在“我创建的任务”列表中看到该任务，任务状态为“待审核”； 2. 平台管理员可以在“待审核的任务”列表中看到用户提交的测试任务。
触发条件	进入创建移动应用自动化测试任务的页面。
正常流程	1. 点击发布任务中的app测试，进入创建测试任务的流程； 2. 点击上传应用选择本地后缀为“.apk”的应用安装包进行上传； 3. 选择自动化测试的服务类型（兼容性、深度遍历等）； 4. 选择可用机型，设置执行时间，填写项目名及项目描述后提交测试。
扩展流程	1. 可以在已上传应用列表中选择应用直接开始创建移动应用自动化测试任务流程； 2. 如果选择的测试类型为功能回放测试，需要提供进行回放的测试脚本文件。

用例UC2：审核移动应用自动化测试任务的用例如表 3.7示，该用例描述了在创建自动化任务后，只有平台管理员角色的账号具备审核任务的权限，管理

员可以在待审核的任务列表中查看已经创建成功的测试任务，并查看任务的详细配置信息，包括了用户提测的移动应用、选择的测试机型、以及测试时间等。审核通过后点击通过按钮，任务状态即更新为“等待执行”，进入自动化测试流程的下一步，为了更加准确的管理任务的状态，云测试平台后端会将测试任务（Order）依据选择的测试设备拆分为多个Job，每个Job对应一台测试设备。

表 3.7: 审核移动应用自动化测试任务用例表

用例编号	UC2
用例名称	审核移动应用自动化测试任务
参与者	平台管理员
前置条件	使用有效的管理员账号成功登录系统，并存在待审核的任务。
后置条件	审核通过，任务状态更新为“等待执行”；审核不通过，任务状态更新为“审核未通过”。
触发条件	进入管理任务中的审核任务页面。
正常流程	1. 查看所有任务列表，选择待审核的移动应用自动化测试任务，点击任务详情； 2. 进入任务详情页面查看任务的详细配置信息； 3. 点击审核通过，任务依据选择的测试设备进行分发，并等待执行。
扩展流程	审核不通过后填写拒绝原因，并返回给普通人员，按要求更改后可以重新提交任务。

用例UC3：执行移动应用自动化测试任务的用例描述如表 3.8所示，本用例中，参与者只包含具备平台管理员角色权限的账号，管理员可以点击开始执行状态为“等待执行”的测试任务，任务状态更新为“测试中”。测试任务开始执行后，云测试平台后端将用例UC2中拆分的Job 与测试设备状态进行匹配，当测试设备状态为空闲时，即将任务分发给自动化测试工具开始在测试设备上进行自动化测试。

表 3.8: 执行移动应用自动化测试任务用例表

用例编号	UC3
用例名称	执行移动应用自动化测试任务
参与者	平台管理员
前置条件	使用有效的管理员账号成功登录系统，并存在审核通过的任务.
后置条件	任务状态更新为“测试中”。
触发条件	进入任务的详情页面。
正常流程	1. 平台管理员选择要执行的任务的详情页面； 2. 点击执行，则该任务进入执行队列。
扩展流程	无

用例UC4：停止移动应用自动化测试任务的用例描述如表 3.9所示。在云测试平台中，只有平台管理员具有开始执行和停止执行自动化测试任务的权限，

当管理员点击停止后，即可将任务状态置为“任务已停止”，云测试平台后端会将该任务拆分多所有Job状态也进行改变为停止状态，从而通知自动化测试工具停止测试任务。

表 3.9: 停止移动应用自动化测试任务用例表

用例编号	UC4
用例名称	停止移动应用自动化测试任务
参与者	平台管理员
前置条件	使用有效的管理员账号成功登录系统，并存在已开始执行的任务。
后置条件	任务状态更新为“任务已停止”。
触发条件	进入任务的详情页面。
正常流程	1. 平台管理员选择要终止的任务的详情页面； 2. 点击停止按钮即终止该自动化测试任务。
扩展流程	无

用例UC5：查看移动应用自动化出测试报告的用例描述如表 3.10所示。在任务状态为“测试中”时，用例UC2拆分的每个Job执行完成测试并进行日志解析后，都会通知云测试平台，用户可以查看基于当前已完成的Job生成的测试报告，测试任务的发起者和平台管理员均有权限查看报告。缺陷报告生成服务作为服务提供方，云测试平台作为服务调用方通过Dubbo RPC从缺陷报告生成服务获取处理后的数据，并提供给前端框架进行页面渲染。当所有Job均完成时，测试任务的状态会更新为“测试已完成”。

表 3.10: 查看移动应用自动化测试报告用例表

用例编号	UC5
用例名称	查看移动应用自动化测试报告用例表
参与者	普通用户、平台管理员
前置条件	使用有效的管理员或普通用户账号成功登录系统，并存在状态为“测试中”的测试任务。
后置条件	进入测试报告界面。
触发条件	进入任务的详情页面，点击查看报告。
正常流程	1. 用户进入需要查看报告的任务详情页面点击查看报告； 2. 云测试平台会通过Dubbo接口调用缺陷报告生成服务获取解析数据； 3. 已完成的报告可以看到测试概况，缺陷列表，缺陷详情等信息。
扩展流程	日志解析未完成的任务会在报告页面显示解析进度。

用例UC6：选择测试报告中缺陷转为众包测试的用例描述如表 3.11所示。在本用例中，测试任务的发起者与平台管理员均有权限将缺陷转为众包测试进行二次验证。测试人员对于报告中存在疑虑的缺陷，可以单独或批量选择后，

并选择接收该缺陷验证的众包测试项目组，即可发起众包测试任务。其他测试人员可以加入该众包测试项目组对缺陷进行验证，云测试平台还提供了在线的远程真机调试功能方便缺陷验证。

表 3.11: 选择测试报告中的缺陷转为众包测试用例表

用例编号	UC6
用例名称	选择测试报告中的缺陷转为众包测试用例表
参与者	普通用户、平台管理员
前置条件	使用有效的管理员账号成功登录系统，并存在已解析完成的缺陷报告和接受众包验证的众包项目组。
后置条件	在众包测试任务列表中存在该缺陷验证任务。
触发条件	选择缺陷报告中的若干缺陷，并点击“转为众测”按钮。
正常流程	1. 查看已解析完成的缺陷报告中的缺陷列表； 2. 选择希望转为众包验证的缺陷； 3. 选择接收众包验证任务的众包项目组； 4. 填写配置项（项目名称，项目描述，开始时间等）； 5. 提交众包验证任务。
扩展流程	无

用例UC7：选择并构建缺陷分类器模型的用例描述如表 3.12所示。平台管理员拥有权限选择缺陷分类器模型的权限，管理员点击创建分类器模型后，缺陷报告生成服务会依据当前缺陷训练集将所有分类算法对应的模型均进行构建，并将各个模型的评估数据通过前端页面呈现给管理员，如果管理员不进行选择默认将分类效果最优的模型存入Redis缓存用于后续日志解析的缺陷分类。

表 3.12: 选择并构建缺陷分类器模型用例表

用例编号	UC7
用例名称	选择并构建缺陷分类器模型
参与者	平台管理员
前置条件	使用有效的管理员账号成功登录系统，且缺陷报告生成服务中已存在训练数据集
后置条件	在缺陷分类器模型列表中存在该分类器模型。
触发条件	进入训练分类器模型界面。
正常流程	1. 平台管理员点击训练缺陷分类器模型； 2. 缺陷报告生成服务基于训练集进行分类器训练，默认选择分类评估数据较优的模型并返回。
扩展流程	平台管理员手动选择构建特定分类算法的分类器模型。

用例UC8：认证缺陷并提交审核的用例描述如表 3.13所示。本用例的目的在于丰富缺陷的训练集，增强缺陷分类器模型的分类能力。测试任务发起者在查看缺陷报告时，可以查看缺陷的缺陷报告生成服务提供的缺陷类别、产生原

因和解决方案，如果认同即可点击同意并提交，如果不认同可以填写自己任务的分类、发生原因和解决方案等内容并提交由平台管理员审核。

表 3.13: 认证缺陷并提交审核用例表

用例编号	UC8
用例名称	用户认证为缺陷，填写缺陷原因、分类、解决方案，并提交审核
参与者	普通用户、平台管理员
前置条件	使用有效的普通用户或管理员账号成功登录系统，且存在解析已完成的缺陷报告。
后置条件	平台管理员的待审核自动化测试缺陷列表中存在该缺陷。
触发条件	进入报告的缺陷详情页面，并选择缺陷认证申请。
正常流程	1. 用户对于缺陷报告中分类不明确或系统推荐的产生原因存在分歧的缺陷，点击申请认证缺陷； 2. 填写自己对缺陷的分类，发生原因以及对应的解决方案并提交申请。
扩展流程	从众包测试的缺陷验证中也可以进入申请认证缺陷的流程。

用例UC9：平台管理员审核缺陷并更新分类器模型的用例描述如表 3.14 所示。本用例的参与角色只有平台管理员。管理员可以查看用户提交的待审核缺陷，如果内容属实，即可通过审核。此时会将该缺陷添加至缺陷训练集，并构建新的缺陷分类器模型，同时将Redis中旧模型置为过期并把最新的模型更新至Redis缓存中。

表 3.14: 平台管理员审核缺陷并更新分类器模型用例表

用例编号	UC9
用例名称	平台管理员审核缺陷并更新分类器模型
参与者	平台管理员
前置条件	使用有效的管理员账号成功登录系统，且在待审核缺陷列表中存在该缺陷。
后置条件	缺陷训练集中增加该缺陷并且分类器模型已更新。
触发条件	进入审核缺陷页面，点击通过审核。
正常流程	1. 查看该缺陷的详细信息，通过审核后，将该缺陷添加至缺陷报告生成服务的缺陷训练集； 2. 点击更新分类器模型，基于更新后的训练集构建新的分类器模型。
扩展流程	审核不通过填写原因返回给提交者。

3.3　移动应用缺陷报告自动生成系统总体设计

3.3.1　系统整体架构设计

移动应用缺陷报告自动生成系统架构图如图 3.3所示，描述了与用户直接交互的展示交互层、与前端和其他服务进行交互的核心业务层以及职责较为单一提供独立服务的基础业务层，并与硬件设备层进行交互。

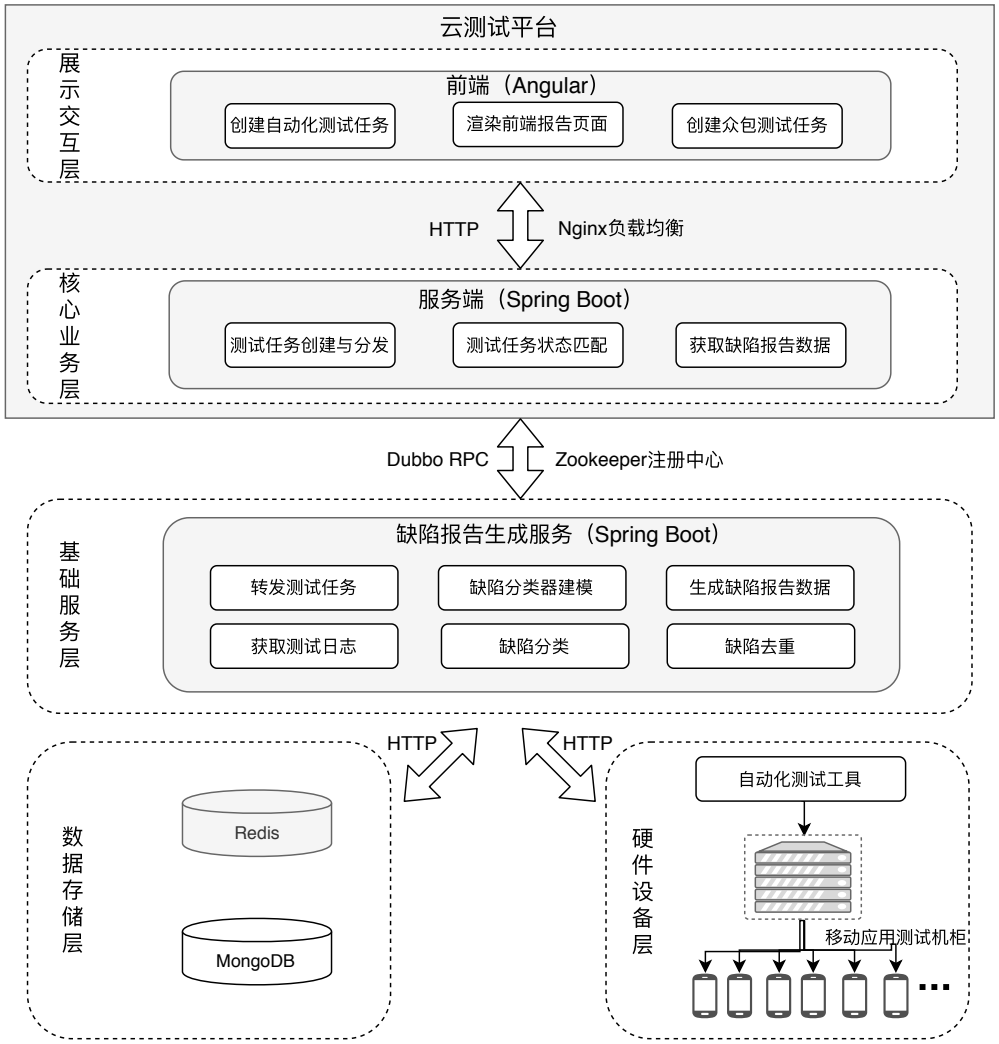

图 3.3: 移动应用缺陷报告自动生成系统架构图

云测试平台提供了移动应用在线自动化测试、众包测试本、在线真机验证等多项功能，本系统作为云测试平台的一个子系统，主要负责移动应用在线自动化测试相关功能，包括创建和分发移动应用自动化测试任务、解析自动化测试结果日志、查看由解析结果生成的自动化测试缺陷报告以及将报告中的缺陷转为众包测试等功能。

展示交互层中，普通用户可以直接在前端页面进行操作，进入创建移动自动化测试任务的流程，此流程主要包含三步操作：上传待测试的移动应用的apk文件，选择测试服务的类型（主要包含兼容性、深度性能、深度遍历、功能回放测试等），补充配置（主要配置信息包含任务的名称、选择进行测试的机型、任务描述等）并提交测试。提交成功后平台管理员可以在展示交互层中的管理任务页面中看到待审核的任务，并可以查看任务的具体配置信息，由管

理员决定是否通过该任务，审核通过后可以选择执行、停止任务。任务完成后，创建者可以在任务详情中查看测试的结果报告，并可以选择报告中展示的缺陷转为众包测试，并提供在线验证的功能。

核心业务层中，平台采用前后端分离的架构，自动化测试主要提供了后端业务逻辑的实现，通过HTTP调用对前端提供接口，通过Dubbo接口与其他微服务进行交互。云测试平台服务端与本系统相关的功能是接受用户在前端创建的移动应用自动化测试任务，并不断接收来自日志解析推送的移动应用测试机柜中测试设备的最新状态。当平台管理员审核通过并点击执行任务后，后端的业务逻辑会通过状态机管理测试任务的状态与设备状态，当状态匹配成功后，任务的状态更新为匹配成功，等待执行测试。

基础服务层中，缺陷报告生成服务是本系统中较为核心的部分。缺陷报告生成服务作为独立部署的微服务，负责从云测试平台轮询地拉取状态为匹配成功的测试任务，将任务的配置等基本信息存入MongoDB后，将任务转发给自动化测试工具，由自动化测试工具中的自动化测试框架对任务中所包含的移动应用进行测试。测试完成后，自动化测试工具将测试结果日志（主要包含了Logcat日志、操作日志、界面截图等）上传至缺陷报告生成服务所在服务器，在所有日志上传完成后，通知缺陷报告生成服务可以开始进行解析任务。解析过程中需要对缺陷分类，首先尝试从Redis缓存中加载模型，如果缓存中没有，则从数据库中获取同时将其更新至缓存中，提高之后日志解析速度。解析完成后的结果也存入MongoDB中，并通知云测试平台任务完成进展，并对云测试平台提供Dubbo接口用来调用获取解析结果。缺陷报告生成服务之所以要做成微服务，是因为该服务需要在多个自动化测试机柜中进行部署，每个机柜可能都有不同的IP地址，如果不使用微服务的Dubbo框架，则需要在云测试平台维护所有的缺陷报告生成服务实例的IP地址，极大的增加了维护成本。缺陷报告生成服务设计为微服务后，服务的发现与注册由注册中心Zookeeper 来维护，并且能够将复杂的日志处理逻辑以及对其他服务的调用等从平台中抽离，便于进行快速开发、测试和部署，也符合软件设计中的高内聚、低耦合的原则。

3.3.2　4+1视图

本节通过“4+1”视图模型对移动应用缺陷报告自动生成系统进行阐述，其中包含了逻辑视图、过程视图、物理视图（部署图）、开发视图和场景视图。其中场景视图对应于UML描述的用例图，如上文图 3.2所示。

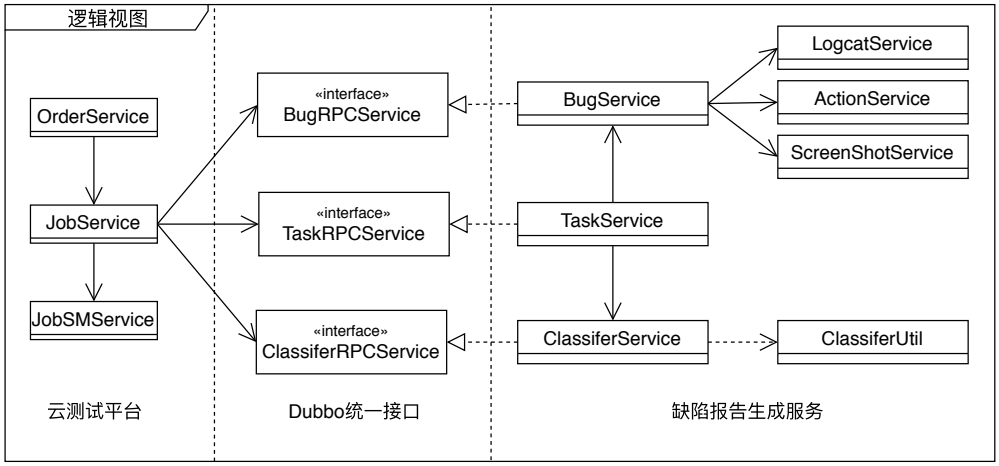

图 3.4: 逻辑视图

本系统的逻辑视图如图 3.4所示。逻辑视图面向最终用户视角，关注点为功能性需求，即系统为用户应该提供的服务，可以使用UML中的类图进行描述。系统可以分解为一系列关键的抽象，展示提供的服务。在本系统中，通过Dubbo统一接口定义了缺陷报告生成服务对外提供的服务，如图中的BugRPCService、TaskRPCService、ClassiferRPCService分别提供关于构建缺陷、日志解析任务、缺陷分类器模型的服务接口，缺陷报告生成服务负责实现这些接口，云测试平台引入统一接口的依赖即可调用其中定义的远程服务。云测试平台中的OrderService 负责测试任务的创建、审核等服务，JobService负责订单（Order）拆分后的每一个Job的状态匹配等服务，JobSMService管理每一个Job的状态机（具体管理方式详见本文第四章）。

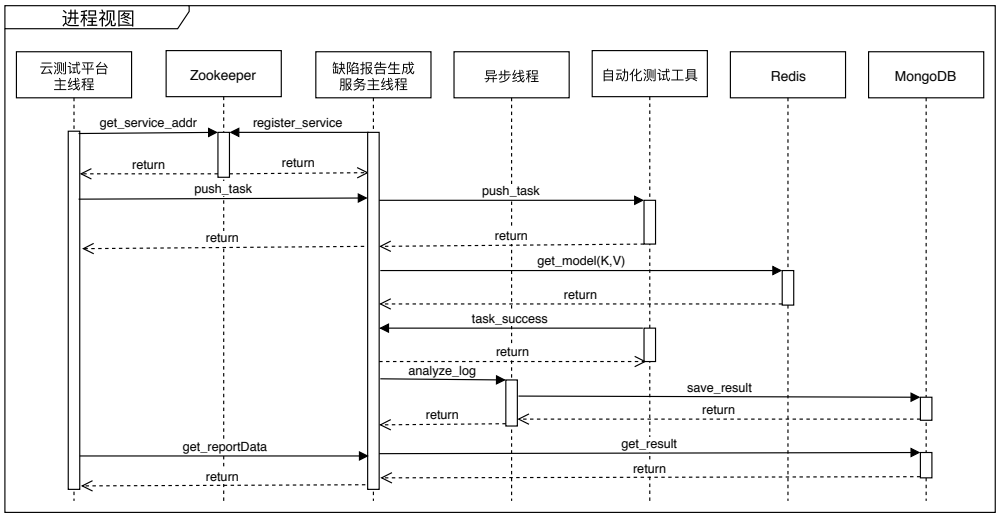

图 3.5: 进程视图

本系统的进程视图如图 3.5所示。软件架构设计中进程视图面向系统集成者视角，描述了系统主要部分的通信。在Dubbo框架下，缺陷报告生成服务作为服务提供方启动成功时向注册中心Zookeeper注册服务，云测试平台作为服务消费方启动成功时从Zookeeper获取提供者的列表，从而可以通过RPC 接口调用。缺陷报告生成服务从自动化测试工具获取所有日志后，启用异步线程进行日志解析，并从Redis缓存载入缺陷分类器模型进行缺陷的分类，并将结果存入MongoDB中。云测试平台通过Dubbo RPC接口从缺陷报告生成服务获取缺陷报告相关数据。

系统开发视图如图 3.6所示。开发视图面向程序员、产品经理等视角，主要关注点在软件开发的模块组织。由于系统为微服务架构，故服务端由云测试平台和缺陷报告生成服务两部分构成。前端有Angular框架负责对TypeScript、CSS、HTML、资源文件（图片等）进行打包。服务端均以Controller包、Logic包、Service包的结构进行组织。其中Controller包负责接收前端或者其他服务的接口调用，Logic包中的类由Controller直接调用，完成对应的业务逻辑，Service包中每个类负责内聚程度更高的服务，提供给Logic进行调用。

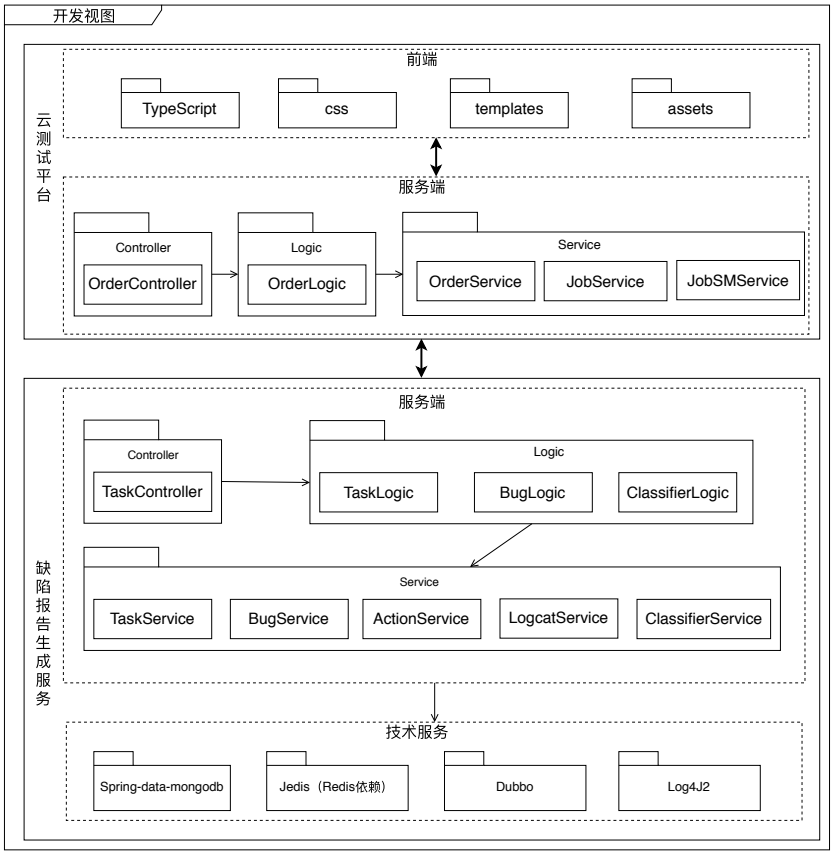

图 3.6: 开发视图

本系统相关的整体部署图如图 3.7所示，即为系统的物理视图。物理视图面向系统工程师视角，主要关注系统依赖于硬件的可用性、可靠性、可伸缩性等。云测试平台前端项目部署于Nginx服务器中，用户通过浏览器发起HTTP请求，经由Nginx进行负载均衡和反向代理后将请求转发给云测试平台服务端。本系统中，为了减轻云测试平台所在服务器的压力，缺陷报告生成服务作为微服务独立部署在服务器上。通过Dubbo 推荐的注册中心Zookeeper提供服务的发现与注册，缺陷报告生成服务作为服务的提供方，云平台作为服务的消费方，云平台与缺陷报告生成服务之间通过Dubbo RPC 接口进行交互。缺陷报告生成服务可以部署多个微服务实例，图中由于篇幅限制省略相应的绘制。由于开发中存在测试和线上两种环境，Zookeeper 采用集群模式进行搭建，不同的环境的服务到相应的注册中心进行注册。

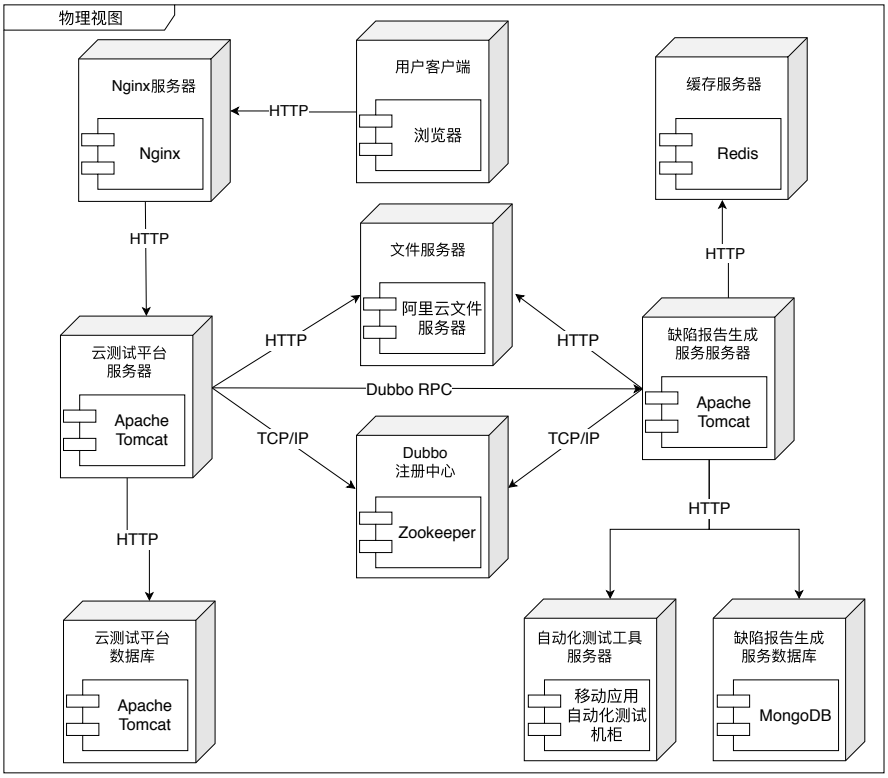

图 3.7: 物理视图

基于阿里云文件存储器的可靠性、安全性、访问速度高等优势，将待测应用的apk文件、应用图标等文件存储于阿里云文件服务器。缺陷报告生成服务所调用的移动应用自动化测试工具部署在移动应用自动化测试机柜，负责提供自动化测试的框架以及环境，与机柜中的测试机进行直接交互。云测试平台与缺陷报告生成服务的后端项目均使用Docker进行容器化，通过Jenkins持续集成。

3.4　测试任务创建与分发模块设计

3.4.1　架构设计

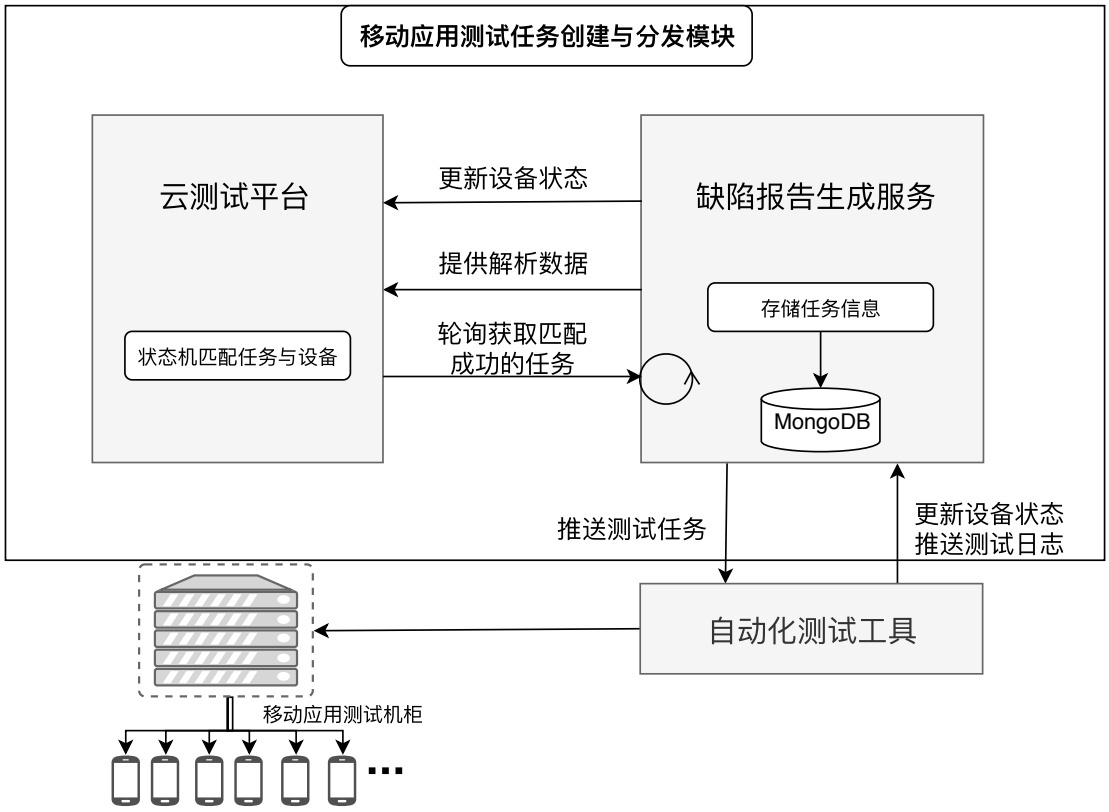

图 3.8: 移动应用测试任务创建与分发模块架构图

移动应用测试任务创建与分发模块的架构图如图 3.8所示。本模块主要负责创建并分发移动应用在线自动化测试后任务。自动化测试工具会不断的向缺陷报告生成服务更新测试机柜中在线设备的最新状态（包含已占用、空闲等），缺陷报告生成服务会将设备状态转交给云测试平台，由云测试平台的状态机负责管理。除此之外，云测试平台的状态机还维护着任务的状态（包含了待审核、待执行、待匹配、匹配成功、执行中等状态）。当平台管理员点击执行任务时，会根据状态机维护的状态，将空闲的设备与待匹配的任务进行匹配，触发状态的匹配成功事件，由事件监听器将任务的状态更新为匹配成功。缺陷报告生成服务会轮询地从云测试平台拉取状态为匹配成功的测试任务，并转发给自动化测试工具进行测试。测试完成后，缺陷报告生成服务会将解析结果存于数据库中，并提供Dubbo接口供云测试平台进行调用。

3.4.2　实体设计

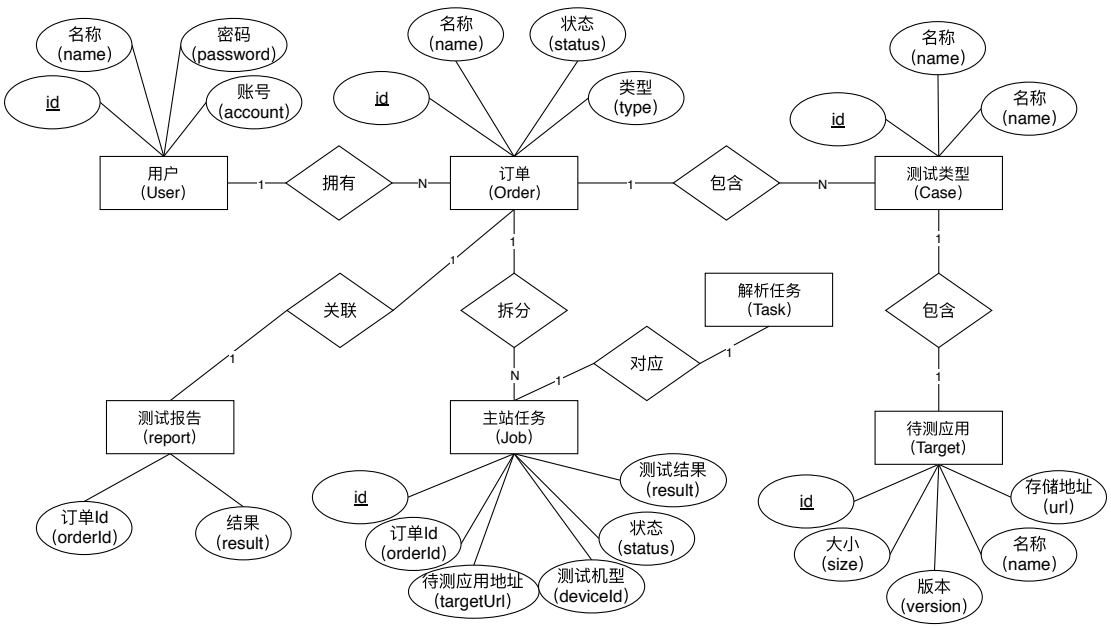

图 3.9: 任务创建与分发模块实体关系图

测试任务创建与分发模块相关的实体关系图如图 3.9所示。实体包括了用户、订单、测试类型、待测应用、主站任务、测试报告等。其中在分发自动化测试任务时，会将订单（Order）依据所选的测试设备数分为相同数目的主站任务（Job），并与缺陷报告生成服务中的日志解析任务（Task）一一对应。对Order进行拆分时为了能够更加准确的管理每个测试设备的任务进程。其中Task实体的具体描述由于篇幅限制仅在下文图 3.13中进行了阐述。

3.5　缺陷分类器建模模块设计

3.5.1　架构设计

缺陷分类器建模模块的架构图如图 3.10所示。数据库中存储的训练集是人工进行标注的数据集，选取多次自动化测试中产生的常见的缺陷，为其打上标签，存于MongoDB数据库中。目前提供基于决策树和朴素贝叶斯分类算法的分类器的构建，当平台管理员选择分类的算法后，缺陷报告生成服务会根据需求构建对应的分类器。考虑到系统的可扩展性，通过抽象工厂方法的设计模式来提供不同的分类器模型，如果增加新的类型的分类器，可以符合开放扩展，关闭修改的设计原则。

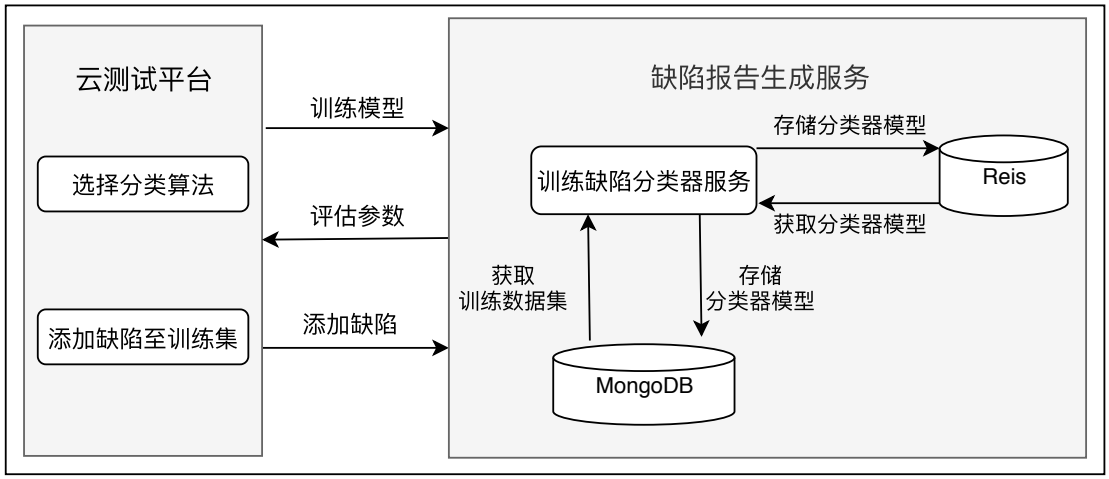

图 3.10: 缺陷分类器建模模块架构图

创建分类器模型的主要流程为缺陷报告生成服务通过Dao层从MongoDB中获取已经标注的训练集，分类器模型需要填充的属性包括训练集中的所有标签以及对应的缺陷属性，构建为格式化的训练集实例（Instances），然后利用构建分类器的工具类构建分类器并产出分类效果评估数据，评估数据包含了最经典的TPFN组合（False Negative，False Positive，True Negative，True Positive），查全率（recall）以及查准率（precesion），ROC曲线等。构建好的分类器存入MongoDB数据库中，同时将分类器模型存入Redis缓存中，当需要对缺陷进行分类时先从Redis 中获取，如果没有再从MongoDB中获取并更新至Redis。当云测试平台为训练集增加数据时，缺陷报告生成服务依据更新后的数据集重新构建分类器模型，并更新至Redis缓存和MongoDB数据库，将缓存中旧模型置为过期。

3.5.2　实体设计

缺陷分类器建模模块实体关系图如图 3.11所示。本模块中将分类器模型（Classifier）和分类器评估数据对象（Evaluation）均封装至ClassifiModel对象中，并存放该分类器的构造时间、所用算法等信息，并利用构建时间的时间戳作为版本，这样便于每次载入时使用最新版本的分类器模型。其中分类器评估数据每个属性值的含义参见表4.6分类器评估参数列表。在分类器（Classifier）中需要定义文本向量化的过滤器以及对应的分类算法。每个缺陷训练集对象（Traindata）包含了缺陷及其对应的类别，构造分类器模型时将训练集对象均载入内存中。

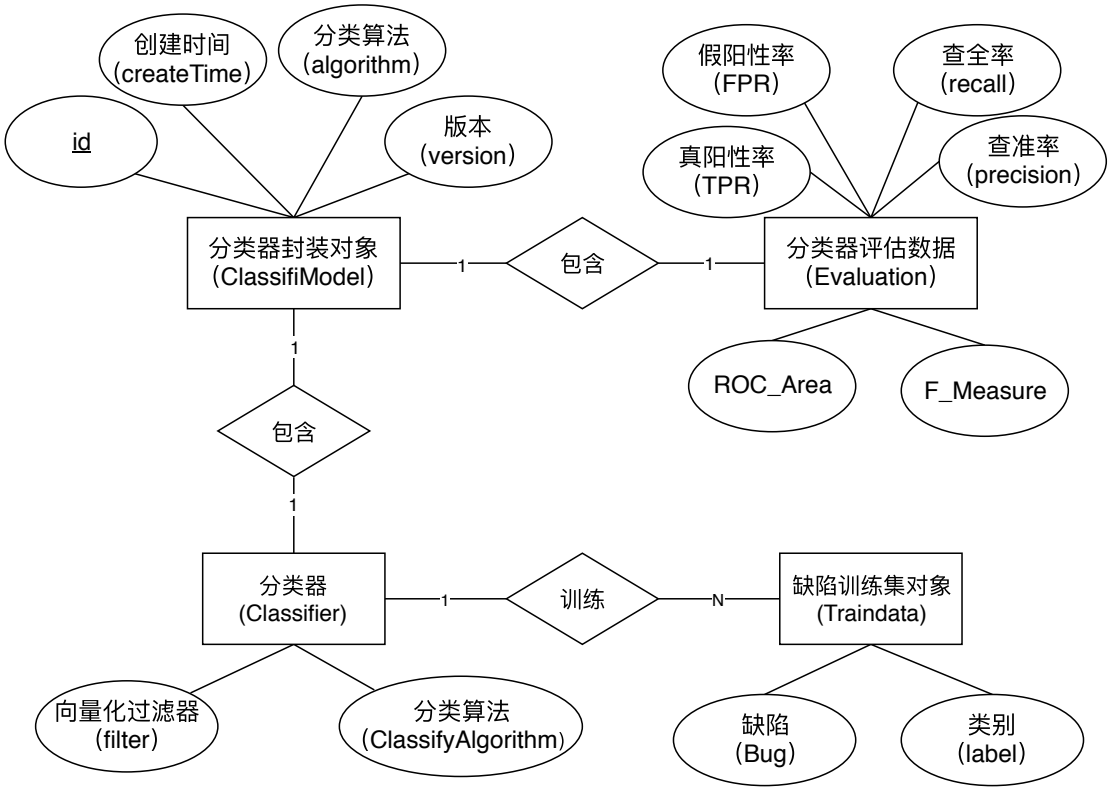

图 3.11: 缺陷分类器建模模块实体关系图

3.6　日志解析与缺陷构建模块设计

3.6.1　架构设计

日志解析与缺陷构建模块的架构图如 3.12所示。自动化测试工具执行完成测试任务后，将Logcat日志、操作日志、截图日志、性能相关日志等测试结果日志上传至缺陷报告生成服务所在服务器，全部上传完成后通知解析服务可以开始进行日志解析工作。

本模块构造Logcat对象、生成操作序列、获取界面截图、生成测试路径功能等都具有相应服务类进行解析，具体解析过程详见第四章中的实现部分，各类日志解析完成后由BugService将缺陷发生时页面截图、发生前后的操作链、操作页面节点、设备性能信息等上下文信息进行关联，结构化的缺陷模型构造完成后从Redis 缓存取出并加载缺陷分类器模型，利用分类器对缺陷进行分类，对短时间内多次重复出现的缺陷进行去重并统计后将解析数据存储与MongoDB数据库进行数据持久化。

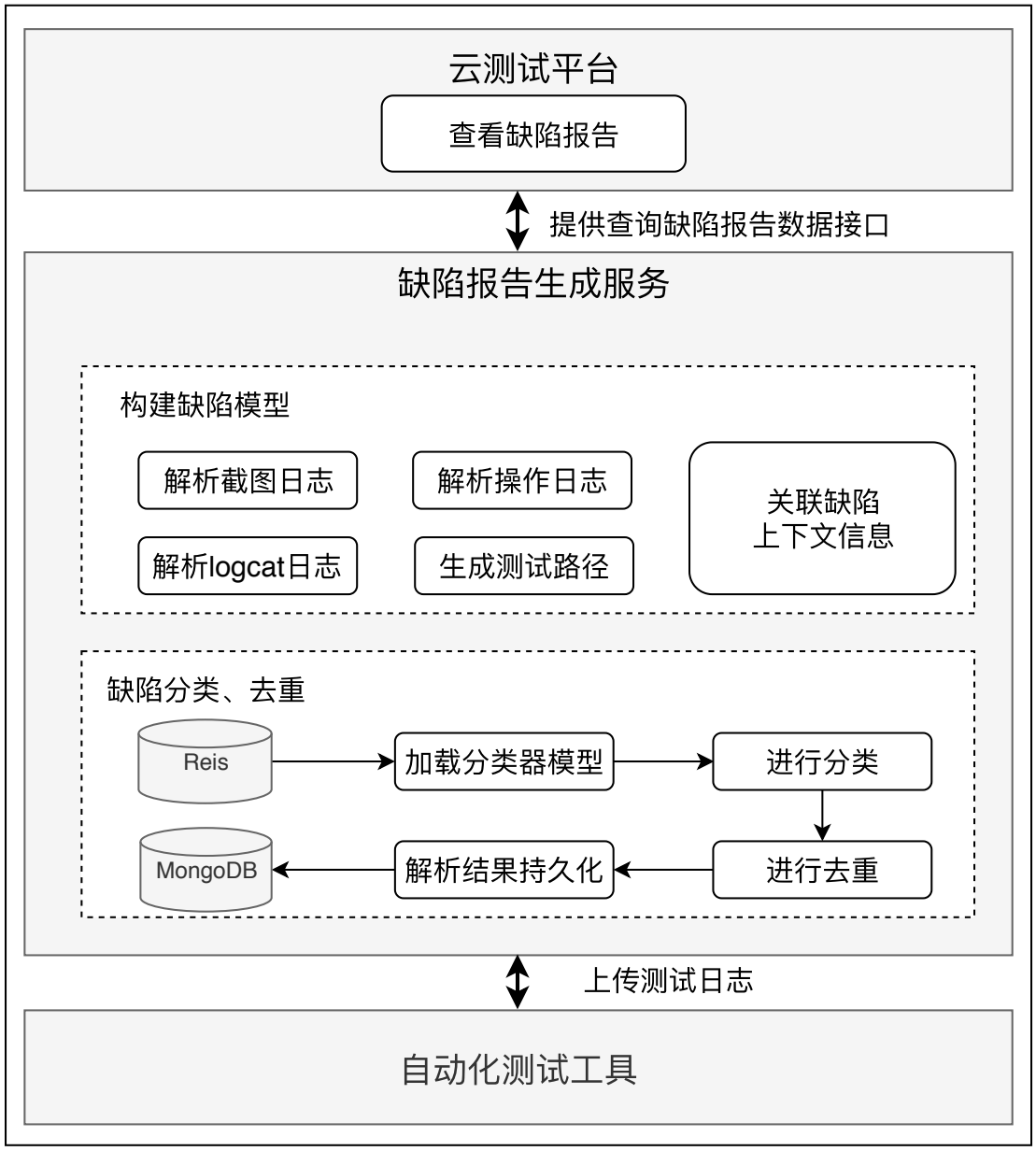

图 3.12: 日志解析与缺陷构建模块架构图

云测试平台与缺陷报告生成服务共同依赖多个统一的服务接口以及数据传输对象（DTO），这些接口定义了缺陷报告生成服务提供给云测试平台的服务，数据传输对象定义了服务交互的数据格式。缺陷报告生成服务负责这些接口的具体实现，通过Dubbo框架与云测试平台进行关联，这样云测试平台在展示自动化测试缺陷报告时可以像调用本地服务一样调用远程的缺陷报告生成服务获取缺陷报告相关数据。

3.6.2　实体设计

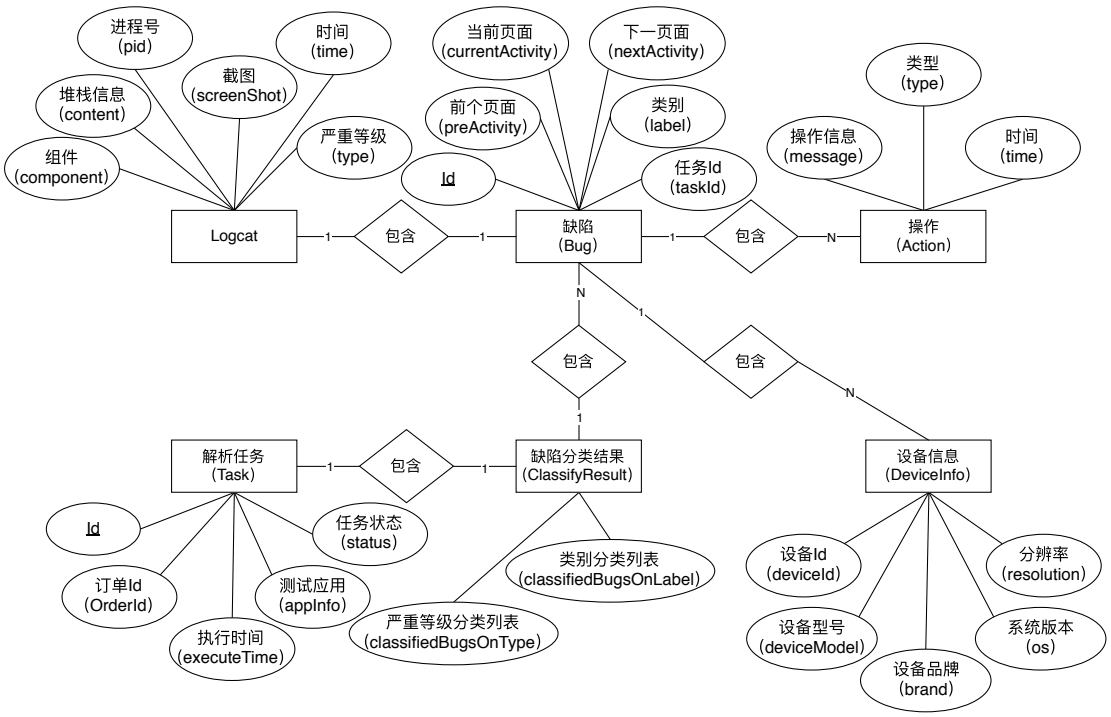

图 3.13: 日志解析与缺陷构建模块实体关系图

日志解析与缺陷构建模块实体关系如图3.13所示。本模块中主要包含日志解析任务、缺陷分类结果、缺陷、操作、Logcat对象、操作对象、设备信息等实体。缺陷报告生成服务从Logcat日志中解析出Logcat对象，其中包含了缺陷的进程号、堆栈信息、严重等级等基本信息。从Action操作日志中解析出操作对象，并构建测试路径与缺陷相关联。根据分类和解析结果，对每一个日志解析任务生成一个对应的分类结果。基于订单所拆分的所有日志解析任务的解析结果，生成一个与订单一一对应的订单结果报告。

订单（Order）对应的所有日志解析任务完成后，缺陷报告生成服务会向云测试平台进行汇报，更新订单的状态，当用户选择查看缺陷报告时，云测试平台通过Dubbo接口从缺陷报告生成服务中获取报告相关的数据，从而能够进行前端页面的渲染。云测试平台与缺陷报告生成服务之间交互的对象使用统一的数据传输对象（DTO）进行传递，获取后根据服务自身需求在利用对应的装饰器（Wrapper）进行自定义的包装。使用DTO的好处是将缺陷报告生成服务与云测试平台的数据结构进行解耦，DTO中只需要定义服务间交互需要的属性，而不用将服务内部的领域对象暴露给对方。

3.7　缺陷报告前端渲染模块设计

3.7.1　前端页面组件设计

表 3.15: 缺陷报告前端组件列表

父组件	子组件
查看结果报告（auto-report）	测试应用基本信息（app-baseInfo）
	测试概况、缺陷列表、设备详情切换页（p-tabPanel）
测试概况（test-overview）	设备运行概况（device-summary）
	问题概述（bug-summary）
	问题机型聚类（bug-cluster）
缺陷列表（bug-list）	缺陷表格（p-dataTable）
设备列表（device-list）	设备表格（p-dataTable）
	设备详情（device-detail）
缺陷详情（auto-report-bugDetail）	缺陷概述（bug-info）
	缺陷上下文信息（bug-context）
设备详情（report-deviceDetail）	当前设备缺陷列表（p-dataTable）
	缺陷类型总结（device-bug-summary）

表 3.15列出了缺陷报告前端展示页面的主要组件。在Angular中，每个组件都可以有对应的模板（template）和样式（style），均为可独立复用的部分。如果把所有的需要显示的内容都放在同一个组件中，那么当应用越来越复杂后将变得不可维护。所以系统在缺陷报告前端部分采用了主从组件的设计，整个缺陷报告是由多个组件组合而成，每一个父组件由多个子组件组成，每个组件符合单一职责原则，实现子组件与父组件的解耦。这样既保证了父组件的灵活性，又便于进行组件的复用。例如表中的问题机型聚类（bug-cluster）组件，由于每一个类型的缺陷都会存在该类缺陷的问题机型聚类，父组件测试概况（test-overview）中使用Angular的模板语法“*ngFor”即可多次复用该组件，只需要填充不同的数据对象即可。

Angular项目中使用TypeScript定义组件、HTML定义模板、CSS定义样式，并在组件对应的ts文件中以元数据的方式进行关联。父组件对应模板通过引入HTML 元素的方式引入子组件，通过属性绑定来对子组件的数据进行控制，子组件可以利用创建EventEmitter 的方式自定义事件，通过事件绑定监听事件，当触发事件时便可将内部数据传递给父组件，并可以通过双向绑定在页面显示的数据能够同步根据用户的修改发生相应改变。前端项目与服务端的交互可以利用Angular提供的HTTP对象进行接口调用，本项目中将调用服务端的代码抽象为Service通过依赖注入的方式传入组件中供组件调用。

3.7.2　前端渲染模块实体设计

移动应用云测试缺陷报告前端实体关系图如图 3.14所示。由于本系统的前端基于Angular框架，使用TypeScript语言进行开发，所以在前端项目中也需要定义实体结构，所以有了下文的实体关系图。前端定义的实体结构应该尽量与后端中定义的VO保持一致。后端之所以定义VO 是因为前端不一定需要在后端定义的实体中的所有属性，并且直接将后端实体呈现给前端也不符合安全性。另一个原因就是希望直接在后端构造好前端页面展示所需要的各属性，从而减少前端系统的运算逻辑。例如图中的Bug-Cluster对象，前端在接受到所有缺陷后也可以自行进行各属性的运算，但是这些运算逻辑放在后端不仅更加方便高效，也使前端的逻辑更加明确，主要负责显示和用户的页面操作逻辑即可。因此，服务端可以通过包装类（Wrapper）构造前端需要的对象，在Controller中使用@RestController注解，直接将该对象以JSON格式返回给前端。

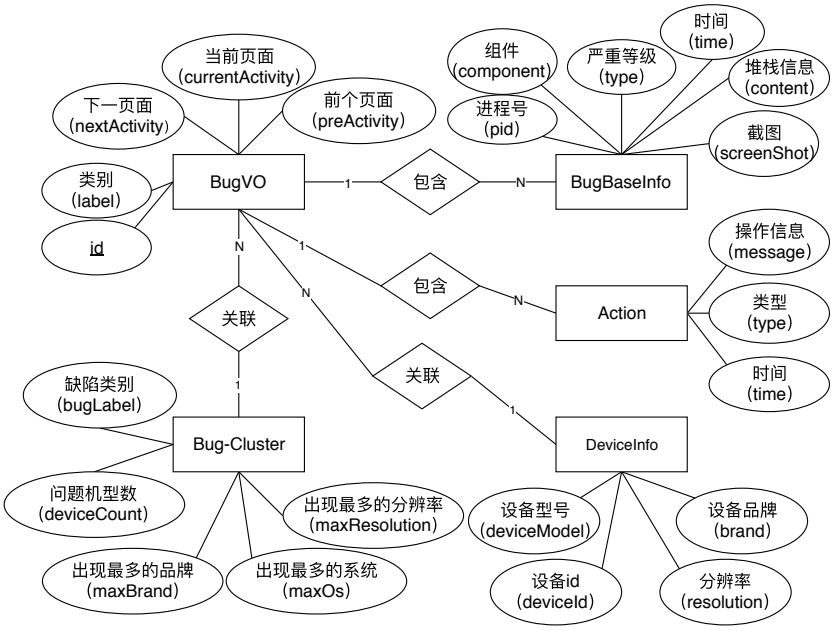

图 3.14: 缺陷报告前端实体关系图

3.8　本章小结

本章对移动应用缺陷报告自动生成系统的功能性需求、非功能性需求、系统用例以及总体设计进行阐述。系统依据业务流程和功能需求分为测试任务创建与分发模块、缺陷分类器建模模块、日志解析与缺陷构建模块和缺陷报告前端渲染模块四个模块，并且通过各模块的架构设计与实体设计详细阐述了系统的各模块设计以及各模块之间的关系。

第四章　移动应用缺陷报告自动生成系统详细设计与实现

4.1　测试任务创建与分发模块

移动应用在线自动化测试任务创建与分发模块主要负责了用户创建移动应用自动化测试任务的流程，包含了普通用户创建任务、平台管理员审核任务、平台管理员执行任务、任务状态匹配、任务转发给自动化测试工具等功能。本小结阐述了移动应用云测试任务创建与分发模块的详细设计与实现。

4.1.1　流程设计

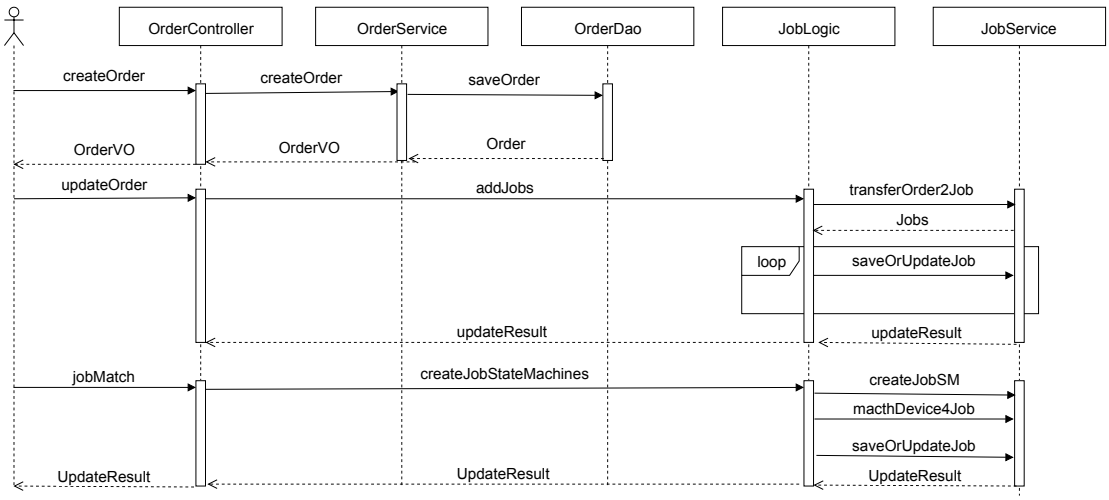

图 4.1: 测试任务创建与分发模块创建任务时序图

移动应用测试任务创建与分发模块的创建任务部分如图 4.1 所示。普通用户通过前端页面提交测试任务后，前端框架会调用云测试平台服务端的OrderController中的createOrder接口将测试任务的订单对象上传至服务端，后续分别通过OrderService和OrderDao层对创建的任务进行数据库的存储工作。

平台管理员审核测试任务并审核通过后，前端框架会调用云测试平台服务端的OrderController中的updateOrder接口更新测试任务的状态，后续调用JobLogic中的addJobs 方法将测试任务拆分成多个Job，拆分的依据是测试任务中包含的用例数和测试设备数，一个Job对应一个设备和一个用例。将拆分的

所有Job状态置为待执行（WAIT_TO_EXECUTE）并存入数据库后更新测试任务的状态为审核通过。

平台管理员点击执行任务后，前端框架会调用云测试平台后端的OrderController中的jobMatch接口。该接口的作用是将测试任务所拆分的Job与移动应用测试机柜中的测试设备进行匹配。系统对Job和测试设备的状态管理采用了状态机的设计模式（具体匹配方式与实现细节详见4.1.3状态机设计模式与实现），jobMatch接口被调用后，会调用JobLogic的createJobStateMachines的方法为当前状态为待执行的所有job分别构造一个状态机，并将状态机的状态初始化为待匹配（WAIT_TO_MATCH）。JobService的matchDevice4Job方法遍历在线设备列表，如果Job所需的设备状态为空闲，则由Job的状态机发出匹配成功事件，并由对应监听器将Job 状态更新为设备匹配成功（MATCHED）。测试设备的状态机发出被占有事件，并将设备状态更新为已锁定（LOCK），此时该设备无法执行其他Job对应的测试任务。

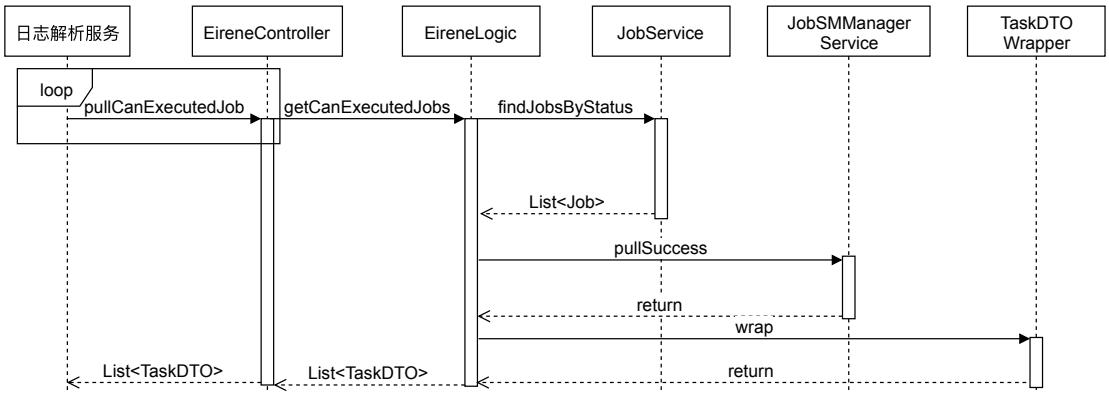

图 4.2: 测试任务分发时序图

图 4.2描述了缺陷报告生成服务从云平台拉取云测试任务的过程。缺陷报告生成服务轮询地调用云测试平台的EireneController中pullCanExecutedJob接口，从而获取所有状态为匹配成功（MATCHED）的Job，并将Job转发给自动化测试工具执行测试。当云测试平台收到该调用请求后，交由EireneLogic调用JobService的findJobsByStatus方法，查询所有状态为MATCHED的Job，并将每个Job由TaskDTOWrapper封装为TaskDTO返回给缺陷报告生成服务。任务获取成功后云测试平台服务端的EireneLogic会调用Job的状态机管理服务JobSMManagerService发出任务获取成功的事件，监听器在接收到该事件后会将对应Job的状态更新为测试中（RUNNING）并更新至数据库中。

4.1.2　数据与类设计

后端采用分层的设计，其中控制器层（Controller）主要负责接收web请求，经过路由分配到达控制器层。在本系统中主要是用来接收前端和缺陷报告生成服务的调用请求。逻辑层（Logic）由Controller直接调用，可以调用多个基础service完成复杂的业务逻辑。服务层（service）主要负责对应实体相关的服务，内聚程度较高，如JobService主要负责对Job实体的处理，为Logic层提供服务。包装类（Wrapper）在分层上也属于服务层，可以提供数据对象转换的服务。数据访问层（Dao）主要负责对应实体的数据库操作。

表 4.1: 测试任务创建与分发主要类

分层	主要包含的类
控制器层	OrderController、EireneController
逻辑层	OrderLogic 、EireneLogic、JobLogic
服务层	OrderService、 JobService、 JobSMManagerService、 DeviceSMManagerService、TaskDTOWrapper
数据访问层	OrderDao、JobDao
数据类	Order、Job、TaskDTO

移动应用自动化测试任务创建与分发模块主要涉及的类如表 4.1所示。本模块中，控住器层的OrderController负责接收前端系统关于创建、审核、执行测试任务相关的HTTP请求，EireneController负责接收来自缺陷报告生成服务的请求。本模块中Order数据类的详细描述如表 4.2所示，表示用户创建的测试任务。用户创建任务过程中的待测应用、测试设备等信息均存储在该对象内部，为了避免过多无谓前后端交互，故前端先将Order 对象存储在浏览器的LocalStorage中，在用户提交任务后再提交给平台服务端。

表 4.2: Order类详情列表

字段	含义	类型	备注
id	订单编号	Long	主键，不为空
name	订单名称	String	测试任务名称，由用户在任务配置页面填写
status	订单状态	Integer	包含任务审核中、审核通过、不通过、成功等状态。
type	测试类型	Byte	云测试平台目前支持多种测试类型，包含移动应用自动化测试、众包测试、web安全漏洞扫描等测试类型。
cases	测试配置	List<Case>	该属性为嵌套对象，内部包含了config对象用来描述配置信息，如测试脚本、测试机型以及待测应用。

当平台管理员点击执行任务后会将Order依据用户选择的测试设备拆分为多个Job，每个Job对应一个测试设备，该类的详情描述如表 4.3所示。对Order进

行拆分是为了便于匹配测试设备状态，更加精确的测试任务的进度进行管理。云测试平台与缺陷报告生成服务通过TaskDTO作为数据传输对象进行交互，每个TaskDTO对应一个Job，并通过Wrapper进行封装。

表 4.3: Job类详情列表

字段	含义	类型	备注
id	主站任务编号	Long	与TaskDTO编号相同。
deviceUdid	设备编号	String	该任务所分配的测试设备的编号。
scriptUrl	测试脚本地址	String	在功能回放测试中选择用例关联的测试脚本的存储地址。
orderId	订单编号	Long	该任务所属的订单编号。
status	任务状态	Integer	任务的状态，主要包含了待执行、带匹配、匹配成功、正在执行、执行停止、执行成功、执行失败等状态。

移动应用在线自动化测试任务创建与分发模块主要涉及的类图如图 4.3所示。图中DeviceSMManagerService与JobSMManagerService 分别负责对测试设备与主站任务（Job）的状态机进行管理，具体实现方式详见状态机设计模式与实现部分的阐述。

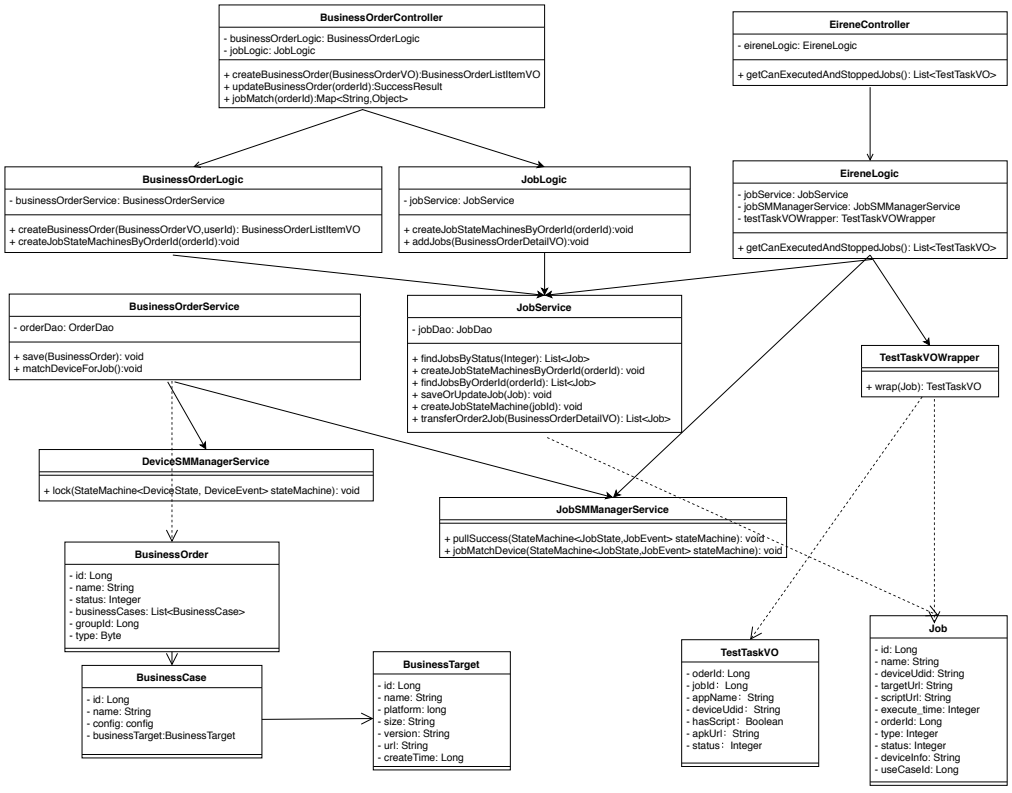

图 4.3: 移动应用自动化测试任务创建与分发模块类图

4.1.3　状态机设计模式与实现

在本模块的开发中，由于Job具有多种状态，并且需要根据状态判断来进行不同的操作。如果使用if…else语句或者switch语句进行判断，就需要根据对象等当前状态在每个分支中写各自的逻辑，那么业务代码中就会充斥着各种条件判断和状态处理逻辑。如果根据需求需要增加中间状态，状体语句的增多或者修改可能会引起很大的改动，极大的减弱了程序的可扩展性与可读性，违背了软件设计中的“开放关闭原则”，代码会变得难以维护。

为了将Job和测试设备（Device）的状态转移处理的代码从业务逻辑中分离出来，本系统采用状态机设计模式。状态机模式基于这样的思路，状态转移可以概括为：$f(S,E) \rightarrow (A,S')$，即如果当前状态为$S$，接收到事件后$E$，则执行动作$A$，同时状态更新为$S'$。

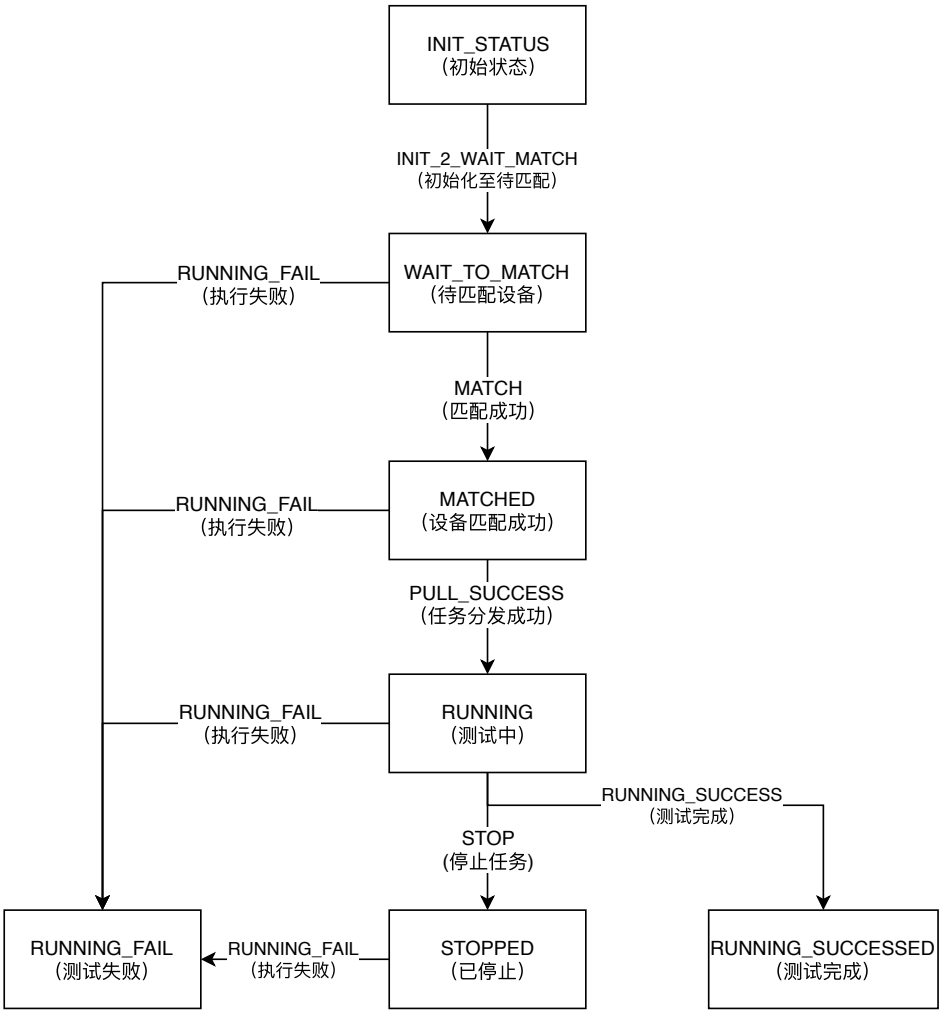

图 4.4: Job状态转移图

在移动应用自动化测试任务创建与分发模块中，Device 与Job的状态都由状态机来维护，实现方式类似，所以此处仅以Job状态机的实现来进行阐述。Job的状态转移如图 4.4所示。使用状态机模式，每一个Job对应一个状态机，由状态机工厂类创建，并由对应的状态机管理类管理所有Job状态机。经过这样的设计，Job的状态管理代码从系统业务逻辑代码中抽离出来，便于扩展也便于后续维护，逻辑也更加清晰，符合软件设计“高内聚，低耦合”的原则。

在本模块中状态机的实现需要引入spring statemachine依赖（即在系统的pom.xml文件中添加spring-statemachine-core的依赖），spring statemachine采用层次化状态机结构，简化了状态配置，在代码上实现了解耦，使逻辑更加清晰，便于开发人员进行维护。实现过程中，首先需要定义Job状态和Job事件的枚举类，如图 4.5所示。

```
// Job 状态枚举类
public enum JobState {
    INIT_STATUS(-1), WAIT_TO_EXECUTE(0), WAIT_TO_MATCH(1), MATCHED(2),
    RUNNING(3), STOPPED(4), RUNNING_SUCCESS(5), RUNNING_FAIL(6);
    @Getter
    private Integer status;
    JobState(Integer status){this.status = status;}
}
// Job 事件枚举类
public enum JobEvent {
    INIT_2_WAIT_MATCH, MATCH, PULL_SUCCESS,
    STOP, RUNNING_SUCCESS, RUNNING_FAIL
}
```

图 4.5: Job状态机状态及事件枚举类

JobAction为Job的执行动作类，可以定义在监听到Job 发出的事件后执行的操作，实现代码如图4.6所示，由于篇幅限制仅截取了Device 与Job匹配成功事件和Job执行成功事件相应的操作。在该类上使用@Service注解使JobAction类可以直接通过Spring的依赖注入机制将实例注入下文中将提到的状态转移的配置类中，从而使配置类可以调用该类中的实例方法。以matchJobAction方法为例，返回值Action是一个函数式接口，该方法的返回值为使用Lambda表达式实现的Action的一个实例。在接收到MATCH事件后，会触发调用matchJobAction返回该实例，并执行其定义的操作，如此处定定义的操作是在接收到MATCH事件时，将对应的Job 状态更新为MATCHED并存入数据库中。

```
@Service
public class JobAction extends BaseAction{
    @Autowired
    JobDao jobDao;
    //Job 与设备匹配成功（接收到 MATCH 事件）的操作
    public Action<JobState,JobEvent> matchJobAction() {
        return stateContext -> {
            // 匹配成功后将数据库中该 Job 的状态更新为已匹配
            Long jobId = Long.valueOf(stateContext.getStateMachine().getId());
            Job job = jobDao.findOne(jobId);
            job.setStatus(JobState.MATCHED.getStatus());
            jobDao.save(job);
        };
    }
    //Jobb 成功执行(接收到 RUNNING_SUCCESS 事件)的操作
    public Action<JobState,JobEvent> jobRunSuccessAction() {
        return stateContext -> {
            Long jobId = Long.valueOf(stateContext.getStateMachine().getId());
            Job job = jobDao.findOne(jobId);
            job.setStatus(JobState.RUNNING_SUCCESS.getStatus());
            jobDao.save(job);
        };
    }
}
```

图 4.6: Job执行动作类

状态机的配置类需要继承EnumStateMachineConfigurerAdapter 类，并重写其configure方法配置状态机的事件、状态转移、执行动作的联系。图4.7 展示MATCH与PULL_SUCCESS事件状态转移配置。其中source指定初始状态，target指定目标状态，event指定触发事件，action 指定监听到触发事件时的执行动作。

```
@Override
public void configure(StateMachineTransitionConfigurer<JobState, JobEvent> transitions)
{
        transitions
                .withExternal()
                .source(JobState.WAIT_TO_MATCH).target(JobState.MATCHED)
                .event(JobEvent.MATCH).action(jobAction.matchJobAction())
                .and()
                .withExternal()
                .source(JobState.MATCHED).target(JobState.RUNNING)
                .event(JobEvent.PULL_SUCCESS).action(jobAction.runJobAction())
}
```

图 4.7: Job状态机配置类的状态转移配置方法

4.2　缺陷分类器建模模块

缺陷分类器建模模块主要负责根据平台管理员的需求，利用已经标注完成的缺陷训练集构建分类器模型。目前实现方案中提供了基于C4.5决策树算法和朴素贝叶斯算法的分类器。

4.2.1　流程设计

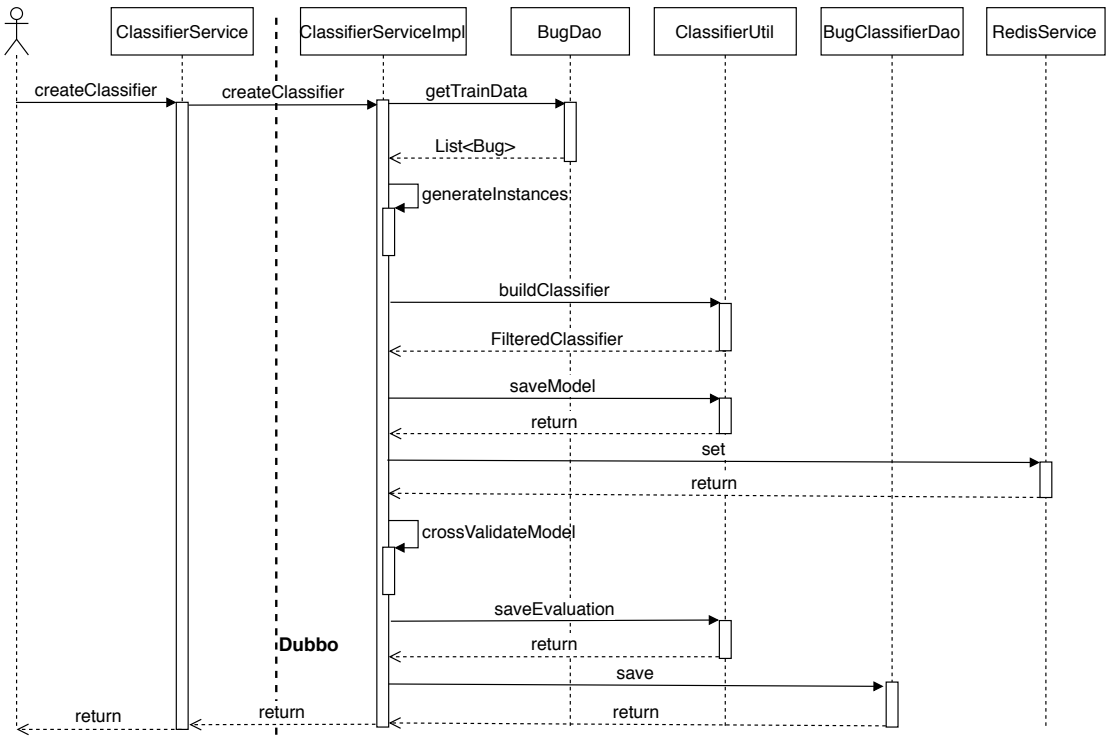

图 4.8: 缺陷分类器建模模块时序图

缺陷分类器建模模块的时序图如图 4.8所示。此模块中云测试平台与缺陷报告生成服务通过Dubbo接口进行交互，图中ClassifierService 即为统一接口，均作为Maven依赖引入到两个项目的pom.xml文件中。云测试平台作为服务消费方，调用ClassifierService的创建分类器模型的方法（createClassifierModel），缺陷报告生成服务作为服务的消费方，实现了ClassifierService中的方法提供给云测试平台进行调用。

在缺陷报告生成服务接收到该方法的调用后，通过BugDao从MongoDB数据库中取出已经标注好类别的缺陷作为分类器的训练集。generateInstances 方法负责将数据集封装为构建分类器需要的格式，并进行设置停用词、构造文

本向量空间模型等数据预处理，调用ClassifierUtil的buildClassifier方法构建缺陷分类器模型。并使用十折交叉验证（crossValidateModel）来评估所用算法的准确性。由于分类器模型和算法评估数据不方便直接存入数据库中，所以先使用分类器工具类将模型和评估数据存入本地文件中，并将文件文件路径作为BugClassifier的属性存入MongoDB中。通过调用RedisService将分类器模型放入Redis缓存中，提高对缺陷分类前载入分类器模型的速度。

4.2.2　数据与类设计

本模块总涉及到的主要类如表 4.4所示。其中ClassifierService作为云测试和缺陷报告生成服务公共依赖接口，定义了缺陷报告生成服务提供了哪些服务给云测试平台进行消费。本模块中只用到了触发构造分类器的方法createClassifierModel。

ClassifierServiceImpl是缺陷报告生成服务中ClassifierService接口的实现类。BugDao与BugClassifierDao分别为Bug和BugClassifier 的数据访问类，内部利用注入的MongoTemplate实例构造与MongoDB进行交互的方法。RedisService负责与Redis缓存进行交互的功能，内部持有JedisPool 实例，为Redis连接池，可以负责Redis缓存对象的存取。

表 4.4: 缺陷分类器建模模块主要类

分层	主要包含的类
服务层	ClassifierService、RedisService
服务实现层	ClassifierServiceImpl
数据访问层	BugDao、BugClassifierDao
工具类	ClassifierUtil
数据类	Bug、BugClassifier、FilteredClassifier、Evaluation

ClassifierUtil为分类器模型的工具类，负责训练和存储模型至文件中。FilteredClassifier即分类器的封装类对象，提供buildClassifier 方法基于训练集构造分类器模型。Evaluation 类是分类器的算法评估的封装类，提供crossValidateModel方法对分类器的分类准确性进行评估。Bug类作为缺陷对象的定义，是较为复杂的嵌套模型，此处主要是从训练集中取出用到，详细定义参见4.3小节中的定义。BugClassifier的定义如表 4.5所示。负责将分类器模型及其评估数据的文件路径等信息进行封装后存于MongoDB数据中，便于分类器加载时获取最新版本的模型，此处使用分类器模型创建的时间戳作为版本号。

表 4.5: BugClassifier类详情列表

字段	含义	类型
id	分类器编号	String
createTime	创建时间	String
version	分类器版本（构建分类器时的事件戳作为版本）	Long
algorithm	算法名（构建分类器时基于的算法名）	String
classifierPath	分类器模型存储路径	String
evalutionPath	分类器评估数据的存储路径	String

缺陷分类器建模模块中涉及的主要类图如图 4.9 所示。其中的核心逻辑类为ClassifierServiceImpl，该类负责整合训练集数据给分类器的工具类（ClassifierUtil），并由该类基于训练集构建分类器模型。Redis 官方建议使用Jedis作为Spring的连接开发工具，在pom文件中添加相应的Maven依赖即可使用，RedisService持有JedisPool成员变量，为Redis的连接池，负责存取Redis 缓存对象。

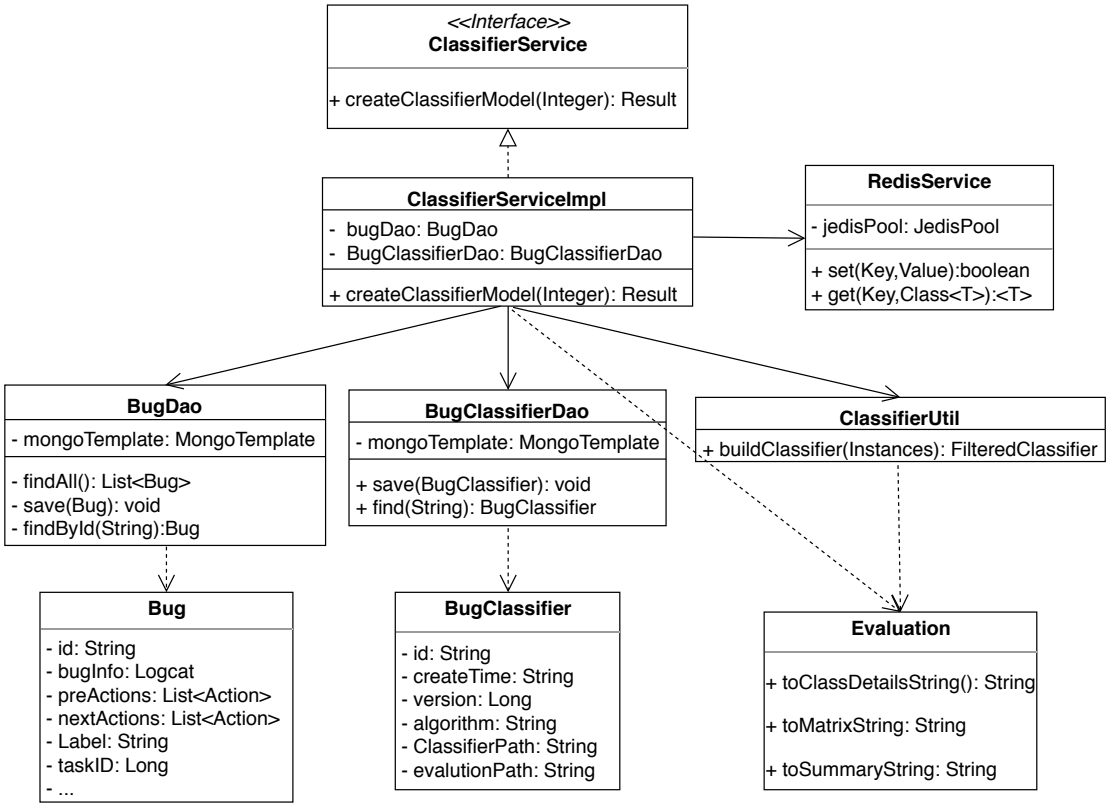

图 4.9: 缺陷分类器建模模块主要类图

4.2.3　分类算法实现

如图 4.10描述了构造分类器的主要流程。为了保证程序的可扩展性，该流程中数据预处理部分和训练模型部分都可以抽象出来为统一的流程，不同算法的实现不同之处在于设置不同的分类算法对象，算法实例只需要实现Classifier接口并重写接口内的方法即可完成构造分类器的流程，本系统实现了C4.5决策树的算法实例和朴素贝叶斯的算法实例。取出训练集后首先需要进行数据预处理，包括构造为Instances对象、分词、设置词干与停用词、构建文本向量空间模型等处理方式。基于缺陷训练集构造分类器模型后，存储于文件中，并默认选择分类效果更优的分类器模型存于Redis 缓存中以提高后续进行缺陷分类时分类器模型的载入速度。

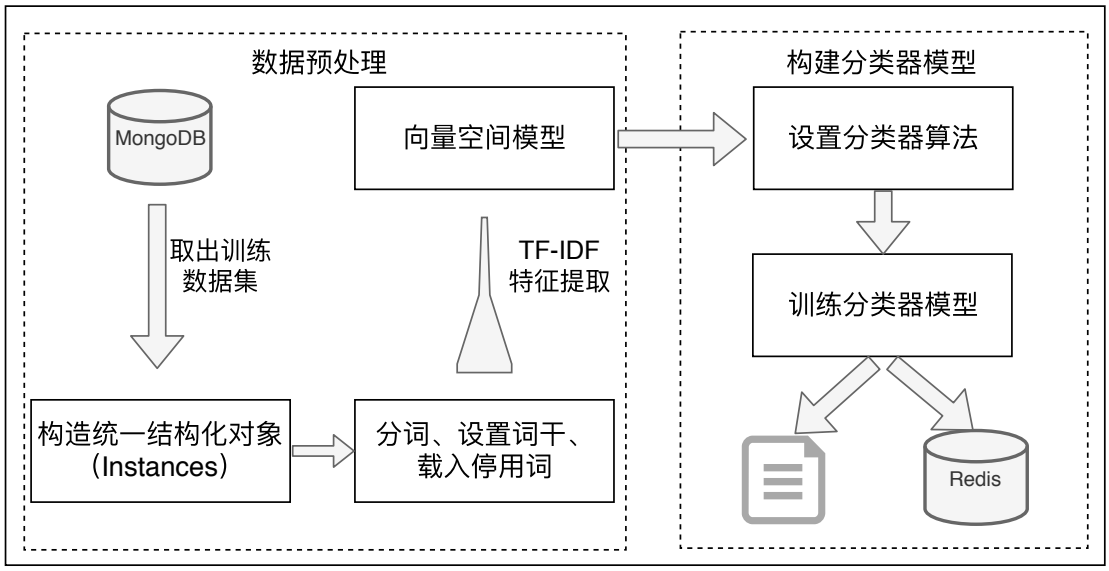

图 4.10: 构造分类器流程

已标注的缺陷训练集存储于MongoDB数据库中，取出训练集后，需要将训练集中的数据构造为统一的格式，即构造为Instances，构造代码如图 4.11所示。Instances的构造函数需要传入attributes属性值，在该方法中添加了两个属性，一个是类别属性，key值为Label，表示缺陷分类的标签，另一个属性就是需要进行分类的属性，key值为content，此处主要使用缺陷的堆栈信息作为分类依据。构造完成后使用instances 的setClassIndex设置第几个属性为类别属性。构造完成后Instances对象就是训练集在内存中的存在形式，基于该训练集，可以开始进行分类器模型的训练。

```
private Instances generateInstances(List<Label2ReasonAndSolution> allLabels) {
List<String> varietyOfLabel = new ArrayList<>(100);
// 获取所有类别名称
    for (Label2ReasonAndSolution label2ReasonAndSolution : allLabels) {
        varietyOfLabel.add(label2ReasonAndSolution.getLabel());
}
// 依据训练集数据构造缺陷详情与类别的对应关系
    List<BugContent2Label> data = createArffData(allLabels);
    ArrayList<Attribute> attributes = new ArrayList<>();
    attributes.add(new Attribute("label", varietyOfLabel));
    attributes.add(new Attribute("content", true));
    // 构建 instances
    Instances instances = new Instances("classifyBug", attributes, 500);
    // 添加数据
    for (BugContent2Label bugContent2Label: data) {
        Instance instance = new DenseInstance(attributes.size());
        instance.setDataset(instances);
       if (varietyOfLabel.contains(bugContent2Label.getLabel())) {
            instance.setValue(0, bugContent2Label.getLabel());
            instance.setValue(1, bugContent2Label.getContent());
        }
        instances.add(instance);
    }
    // 设置分类的索引
    instances.setClassIndex(0);
    return instances;
}
```

图 4.11: 构造Instances方法实现代码

构造Instances完成后首先需要对数据进行预处理，预处理代码如图 4.12所示，目的是将文本格式内容转为向量空间模型内容。预处理的功能主要由StringToWordVector类负责，该类主要依赖Weka[1] 完成具体功能。先进行分词，依据WordTokenizer中定义的英文常用切割符（空格、逗号、分号等）对字符串进行切割。使用词干提取出词根，此处使用的是Lovins 算法，主要策略为依据Lovins 研究后提出的294 种词尾（terminaison），29种构词条件（Condition）和35种转化规则（Transformation rules），在294 个词尾中，寻找符合29个条件的最长匹配词尾，删除找到的词尾，测试截尾后的词是否符合35种转化规则，若符合进行转化 [39]。设置停用词表，读取项目停用词文件，若单词在停用词列表中则不会对该词进行map 映射和在vector 中出现。

[1]https://github.com/Waikato/weka-3.8

```
private Instances stringToVector(Instances instance) {
    StringToWordVector filter = new StringToWordVector();
    Instances dataFiltered;
    try {
        // 设置分词器
        filter.setTokenizer (new WordTokenizer());
        // 设置词干
        filter.setStemmer(new LovinsStemmer());
        // 设置停用词
        filter.setStopwords(new File("resources/stop_words.txt"));
        filter.setUseStoplist(true);
        filter.setOutputWordCounts(true);
        dataFiltered = Filter.useFilter(instance, filter);
        return dataFiltered;
        } catch (Exception e) {
            log.error("预处理异常："+e.getMessage());
        }
    return instance;
}
```

图 4.12: 数据预处理实现代码

构建分类器模型的实现代码如图 4.13所示。设置分类算法，FilteredClassifier类中持有Classifier 成员变量，该类为一个接口，并定义了所有分类器应该具有的基于训练集构造方法（buildClassifier）和对待分类对象进行分类（classifyInstance）的方法。基于C4.5决策树算法的J48类和基于朴素贝叶斯算法的NaiveBayes类均实现了该接口。代码中调用FilteredClassifier的buildClassifier方法实际上就是调用器内部持有的Classifier的buildClassifier方法，后续如果需要新增分类算法，只需要实现该接口即可使用。

构建分类器完成后，使用Weka提供的Evaluation类中的交叉验证方法（crossValidateModel）对分类器的分类效果进行评估 [40]。crossValidateModel方法将训练集内部分为多个子集，每个子集均做一次测试集，其余子集作为训练集，交叉验证多次后取平均结果作为评估数据，该方法的优点在于所有样本都分别作为训练集和测试集参与验证。

为了方便后续分类时分类器的载入，在算法评估完成后将分类器与评估数据以对象的方式存入文件中，并将文件存储路径存入数据库中，将构建完成的分类器模型存入Redis缓存中，提高使用分类器时模型载入的效率。在构建分类器模型的过程中，使用构建完成时的时间戳作为该分类器的版本号，当平台管理员选择分类算法后，如果已经存在该类型分类器模型，缺陷报告生成服务会从该算法的分类器模型中选取最新版本的分类器进行载入。

```
public static FilteredClassifier evaluateAndLearn(Instances trainData) {
    FilteredClassifier classifier = new FilteredClassifier();
    // 可选择不同算法 这里选择效率比较高的算法
    classifier.setClassifier(new NaiveBayes());
    // 使用决策树分类传入相应的算法实例类即可
    // classifier.setClassifier(new J48());
    // 基于训练集训练分类器模型
    classifier.buildClassifier(trainData);
    // 分类器模型评估类
    Evaluation eval = new Evaluation(trainData);
    // 交叉验证评估分类器
    eval.crossValidateModel(classifier, trainData, 10, new Random(1));
    trainData.setClassIndex(0);
    return classifier;
}
```

图 4.13: 训练分类器实现代码

4.2.4　分类效果评估

表 4.6: 分类器评估参数列表

名称	描述
FN	假阴性，即预测类别为no而实际为yes
FP	假阳性，即预测类别为yes而实际为no
TN	真阴性，即预测类别为no实际确实为no
TP	真阳性，即预测类别为yes，实际确实为yes
TP Rate	真阳性率：TP/(TP+FN)
FP Rate	假阳性率：FP/(FP+TN)
Precesion	查准率：TP/(TP+FP)，查准率越高，分类器假阳性率越低
Recall	查全率：TP/(TP+FN)，查全率越高，分类器很少将正例误判
F-measure	Precesion与Recall调和均值，值越高可以保证查全率和查准率都较高
ROC Area	接受者操作特征曲线，越接近1模型越准确

表 4.6 描述了分类器算法评估中每项参数的含义，图4.14和图4.15分别为使用C4.5决策树算法与朴素贝叶斯算法基于训练集进行交叉验证的评估数据。从两张评估数据图中可以看出分类效果差距不大，查准率均在0.9以上，并且ROC曲线值均在0.95以上，具有良好的分类效果。由于F-measure为查准率和查全率的调和均值，值越高可以保证查全率和查准率都较高，从实验结果可以看出C4.5决策树算法的F-measure值较高，系统默认选用该值较高的分类器模型作为日志解析过程中缺陷分类的分类器。

C4.5决策树算法

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	java超时异常
0.778	0.000	1.000	0.778	0.875	0.874	0.990	0.921	Socket异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	参数异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	权限获取错误
0.538	0.000	1.000	0.538	0.700	0.714	0.931	0.817	域名解析失败
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	选择无效的索引
0.800	0.009	0.800	0.800	0.800	0.791	0.896	0.648	找静态元数据条目出错
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	JSON解析异常
0.833	0.000	1.000	0.833	0.909	0.904	0.917	0.850	文件未找到
0.556	0.000	1.000	0.556	0.714	0.732	0.907	0.813	SSL错误
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	DeadObjectException异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	中断异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	空指针异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	状态错误
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	Socket超时异常
1.000	0.139	0.563	1.000	0.720	0.696	0.942	0.767	忽略
0.874	0.021	0.925	0.874	0.876	0.875	0.963	0.895	Weighted Avg.

图 4.14: 使用C4.5决策树分类算法分类评估数据

朴素贝叶斯算法

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.800	0.000	1.000	0.800	0.889	0.891	0.933	0.823	java超时异常
1.000	0.255	0.243	1.000	0.391	0.426	0.985	0.851	Socket异常
0.500	0.009	0.750	0.500	0.600	0.596	0.914	0.666	参数异常
0.800	0.009	0.889	0.800	0.842	0.830	0.995	0.943	权限获取错误
0.692	0.000	1.000	0.692	0.818	0.817	0.991	0.940	域名解析失败
0.750	0.000	1.000	0.750	0.857	0.862	1.000	1.000	选择无效的索引
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	找静态元数据条目出错
0.750	0.000	1.000	0.750	0.857	0.862	0.889	0.768	JSON解析异常
0.833	0.000	1.000	0.833	0.909	0.904	0.996	0.971	文件未找到
0.778	0.009	0.875	0.778	0.824	0.812	0.945	0.831	SSL错误
0.667	0.000	1.000	0.667	0.800	0.809	1.000	1.000	DeadObjectException异常
0.750	0.000	1.000	0.750	0.857	0.862	0.961	0.795	中断异常
0.667	0.000	1.000	0.667	0.800	0.813	0.928	0.702	空指针异常
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	状态错误
0.500	0.000	1.000	0.500	0.667	0.701	0.902	0.647	Socket超时异常
0.500	0.000	1.000	0.500	0.667	0.678	0.996	0.978	忽略
0.739	0.021	0.911	0.739	0.782	0.786	0.975	0.900	Weighted Avg.

图 4.15: 使用朴素贝叶斯分类算法分类评估数据

4.3　日志解析与缺陷构建模块

日志解析与缺陷构建模块主要实现功能为：从自动化测试工具产出的多元日志中构建出包含完整上下文信息（包含发生缺陷前后操作链、操作页面、缺陷发生时界面截图、设备性能信息等）的缺陷模型。缺陷报告生成服务利用缺陷分类器模型对缺陷进行分类后，对同类别重复出现的缺陷通过计算文本相似度的方式进行去重，统计后将所有测试任务的缺陷解析结果进行合并，生成缺陷报告展示需要的数据。该模块主体逻辑由缺陷报告生成服务实现，并提供RPC接口给云测试平台调用获取缺陷报告相关数据。

4.3.1 流程设计

缺陷报告生成服务中提取结构化的缺陷对象流程如图 4.16所示。TaskController负责与移动应用自动化测试工具进行交互，自动化测试工具完成测试任务并将所有结果日志上传至缺陷报告生成服务器后会调用updateTaskStatus接口，缺陷报告生成服务可以开始调用BugLogic提供的parseBugList方法进行日志解析并构建缺陷对象。

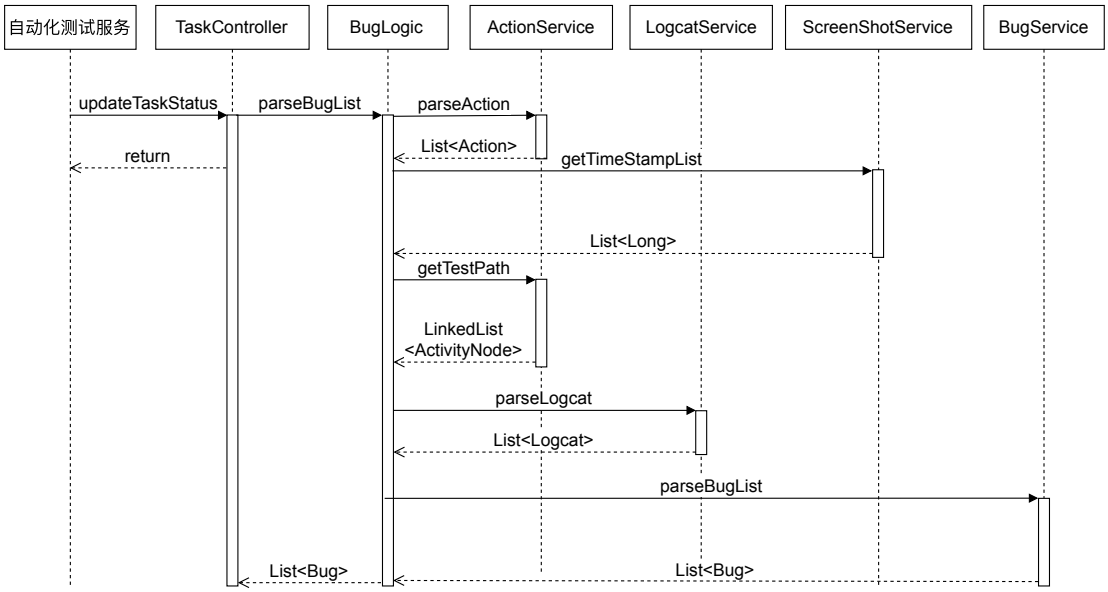

图 4.16: 缺陷报告生成服务提取缺陷对象时序图

ActionService对操作日志进行解析返回操作序列List<Action>，由于页面截图按照时间顺序每隔指定时间进行截图，故ScreenShotService 可获取执行过程中各截图对应的时间列表，再由ActionService依据操作序列内点击页面和进入页面的时间生成测试路径（数据结构为LinkedList<ActivityNode>）。LogcatService负责依据规则寻找可能为缺陷的对象，然后交由BugService补充缺陷的完整上下文信息。

解析出的缺陷进行分类的流程如图4.17所示。首先利用RedisService尝试从Redis缓存中获取最新版本分类器模型，如果缓存中不存在该模型，再由ClassifierDao从MongoDB数据库中取出最新版本分类器模型对应的存储文件将分类器模型加载入内存，同时将分类器模型存入Redis缓存以确保下次可以直接从缓存中载入。

内存中加载完成分类器模型后，对上述解析的缺陷进行分类，添加类别属

性（label）后由BugDao存入数据库。BugService从Label2ReasonAndSolution表获取该标签对应的缺陷可能的产生原因和修复方案提供给用户作为指导信息。由TaskLogic将相应任务的缺陷进行去重后封装为BugClassifyResult表示缺陷的分类结果对象。最后由云测试平台的OrderLogic汇总该order对应的所有Task的分类结果并解析得到前端需要的缺陷类型分布、问题机型聚类等信息。

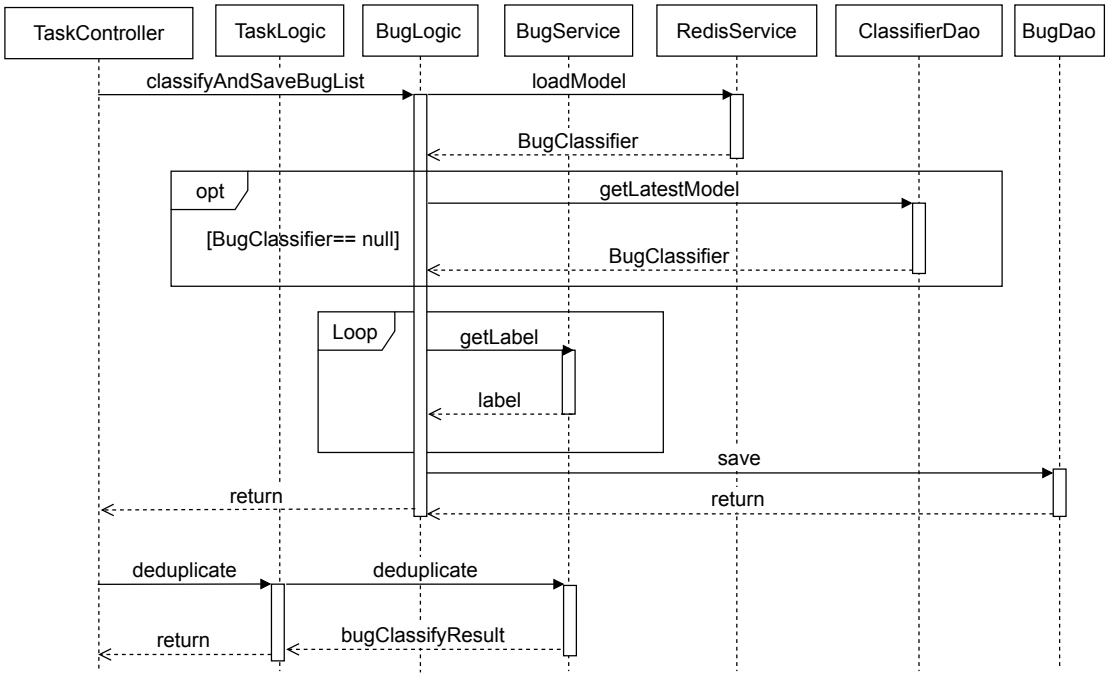

图 4.17: 缺陷报告生成服务缺陷分类时序图

在测试过程中发现存在短时间内出现大量的重复的缺陷的情况，为了避免用户重复审核相同的缺陷，所以在对缺陷分类完成后，缺陷报告生成服务调用BugService的deduplicate方法对同一类别的缺陷进行去重。由于重复缺陷的堆栈信息基本一致，所以采用计算缺陷堆栈信息文本相似度的方式对缺陷去重。

系统采用计算Levenshtein距离作为判断缺陷堆栈信息的相似度的依据，如果相似度高于设定的阈值即判定为重复缺陷，通过配置文件配置阈值以便灵活更改。BugService 统计缺陷重复出现的次数后存入首次出现的缺陷的folds属性中，即代表相同缺陷重复发生的次数，并将重复出现的缺陷编号从BugClassifyResult 中剔除。Levenshtein距离又称为编辑距离，最早由俄国科学家Levenshtein提出 [41]，是指两个字符串之间，由其中一个转换为另一个在许可的编辑操作下所需最少操作次数，许可的编辑操作包括替换一个字符、删除一个字符或插入一个字符。

4.3.2　数据与类设计

表 4.7: 日志解析与缺陷构建模块主要包含的类列表

分层	主要包含的类
控制器层	TaskController
逻辑层	ActionService、LogcatService、ScreenShotService、TaskForDbWrapper、RedisService
数据访问层	BugDao、Label2ReasonAndSolutionDao、ClassifierDao、TaskDao、OrderDao
工具类	ClassifierUtil
数据类	Bug、Action、Logcat、Label2ReasonAndSolution、Task、bugClassifyResult、ActivityNode、TaskDTO、TaskForDb

自动化测试日志解析与缺陷构建模块主要包含的类如表 4.7所示。ActionService、LogcatService、ScreenShotService 分别负责了操作日志、Logact日志、界面截图的解析服务，为BugService的构建缺陷对象提供基础。BugLogic负责缺陷上下文信息的装配逻辑，具体详情参见4.3.3小结的详细实现。ClassifierUtil负责通过读取分类器模型存储文件加载分类器，从而可以对构建的缺陷进行分类。表 4.8为Logcat对象详细定义，该对象从Logcat日志中提取，并根据时间戳关联缺陷发生时的测试设备截图，包含缺陷的进程号、堆栈信息等最基本信息。

表 4.8: Logcat类详情列表

字段	含义	类型
pid	进程号	String
type	严重等级	String
component	产生缺陷的组件	String
content	堆栈信息	String
time	发生时间	String
screenShot	此处关联的是缺陷发生时截图文件路径	String

表 4.9描述了Bug类的详细设计，该类作为实体类存于MongoDB数据库中，表示具有完整上下文信息的缺陷对象，其中preActions和nextActions 分别表示缺陷发生前后的操作序列，组成操作序列的Action类由操作日志解析而来，包含操作的发生时间、操作类型、操作页面（ActivityNode，包含了页面名称与进入该页面的时间）和操作详情等信息，如果操作为点击操作，会在对应的设备截图中对点击事件进行标注，便于复现缺陷。基于操作序列构建测试路径List<ActivityNode>，依据测试路径解析出操作发生时、发生前、发生后的页面，即currentActivity、preActivity、nextActivity属性。

表 4.9: Bug类详情列表

字段	含义	类型
id	缺陷编号	String
preActivity	前一个页面	String
currentActivity	当前页面	String
nextActivity	后一个页面	String
bugInfo	基本信息	String
preActions	缺陷发生页面在缺陷发生前的所有操作	String
nextActions	缺陷发生页面在缺陷发生后的所有操作	String
label	使用分类器对缺陷的分类类型	String
reasons	该类别缺陷发生的可能原因	String
solutions	解决方案	String
folds	短时间内连续发生的此处，去重前进行统计	int
taskID	任务编号	Long

TaskForDb为缺陷报告生成服务从云测试平台拉取的TaskDTO进行封装，详细设计如表 4.10所示。存于数据的测试任务，与TaskDTO一一对应。其中测试状态（testStatus）具有待推送、测试中、解析中、完成成功、完成失败等值，对测试状态的管理也使用4.1.3小节中的状态机模式进行管理。测试任务完成后所有缺陷的解析结果以及分类情况存储于BugClassifyResult中，包含通过缺陷分类器进行类别分类结果以及依据缺陷严重等级分类结果。为了防止对象嵌套层级过深，此处指存储缺陷对应的编号即可。

表 4.10: TaskForDb类详情列表

字段	含义	类型
taskID	任务编号（与TaskDTO编号相同）	Long
orderId	订单编号	Long
testStatus	测试状态	Integer
app	待测应用	String
apkUrl	应用apk文件路径	String
deviceInfo	设备详情	String
bugClassifyResult	缺陷分类结果	BugClassifyResult

图 4.18为测试日志解析与缺陷构建模块的类图，描述了该模块主要类之间的关系。从上到下依次为负责对自动化测试工具提供接口调用的控制器类，控制器类内部持有负责构建缺陷模型和合并各个测试任务结果的逻辑类，逻辑类内部通过调用方法粒度较细的服务类完成相应的操作，最底层为该模块涉及的主要数据实体类。

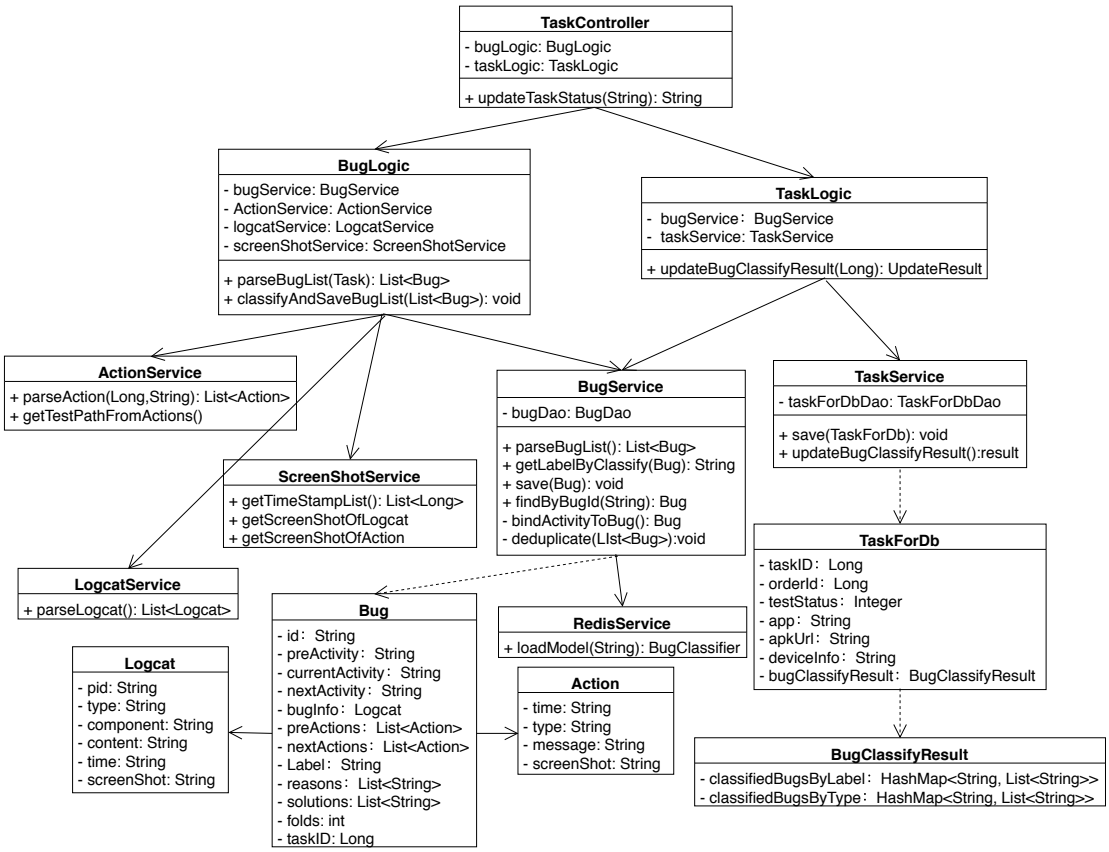

图 4.18: 日志解析与缺陷构建模块类图

4.3.3　日志解析与缺陷构建流程实现

缺陷报告生成服务首先需要获得自动化测试推送的本次任务的所有日志，系统目前主要实现针对安卓应用自动化测试的缺陷报告生成工作，故主要包含了以下日志信息：（1）Logcat日志，Logcat日志是Android中一个命令行工具，可以用于得到程序的日志信息。（2）操作日志，操作日志是指测试脚本中执行的一系列操作所生成的日志，以及经过这些操作后应用页面、组件的变化所生成的日志。（3）截图日志，截图日志主要包含三种，第一种是每个应用页面首次出现时的截图，这种截图对于每个页面有且仅有一个；第二种是截图线程按照时间顺序的截图，这种截图是按照每隔指定时间所截的图，与其他因素无关。第三种是遍历页面的生成树图，描述不同页面出现、跳转的关系图。（4）性能测试相关日志，包括CPU占用、流量消耗、内存占用等性能信息。

自动化测试工具将所有日志信息上传完成后，会调用缺陷报告生成服务的TaskController接口更新任务状态，通知报告生成服务可以开始日志解析工作。

首先对Logcat日志的详细内容进行解析，从原始的字符串信息中提取出结构化的Logcat对象。具体而言，读入每一行均为一个Logcat对象，并用“||”作为分割符分别解析Logcat对象的各个属性。最终返回一个Logcat对象的list。依据“exception”、“error”、“anr”等关键词在Logcat 对象的内容属性中进行检索，找出含有这些关键词的Logcat对象，作为候选的Bug对象。

操作日志记录了自动化测试过程中操作的页面、操作类型、操作坐标等信息，从中可以提取出结构化的Action对象，对Action对象的list进行建模，得到操作序列图模型，并将Bug对象与操作序列图模型依据时间进行关联，得到缺陷发生前后的操作、页面等信息，并利用操作对象中的坐标信息对缺陷截图进行标注，便于后续缺陷的复现。ActionService根据获取的Action对象列表生成操作序列的方法如图 4.19 所示，测试路径的结构为LinkedList<ActivityNode>，其中ActivityNode为页面节点对象，包含了页面的名称和进入页面的时间。

```
@Override
LinkedList<ActivityNode> testPath(List<Action> actionList, List<Long> timelList) {
    if (actionList == null) {
        return testPath;
}
// testPath 即为操作序列对象
    LinkedList<ActivityNode> testPath = new LinkedList<>();
    Collections.sort(timelList);
Long startTime = timelList.get(0);
//  标识是否为第一个页面
    boolean firstActivityFlag = true;
    for (Action action : actionList) {
        String preActivity = action.getPreActivity();
        String nextActivity = action.getNextActivity()
        Long actionTime = action.getTime()
        if (!nextActivity.equals(preActivity)) {
            if (firstActivityFlag) {
                testPath.addLast(new ActivityNode(startTime, preActivity));
                testPath.addLast(new ActivityNode(actionTime, nextActivity));
                firstActivityFlag = false;
            } else if (testPath.getLast().getName().equals(preActivity)) {
                //  如果当前操作内部下个页面和前个页面不同则证明进入新的页面
                testPath.addLast(new ActivityNode(actionTime, nextActivity));
               } else {
                Log.error("页面匹配错误");
            }
        }
    }
    return testPath;
}
```

图 4.19: 获取测试路径方法代码

在BugService中基于Action对象列表、测试路径、作为候选Bug的Logcat对象列表以及从截图日志中解析出的时间序列解析出Bug对象，具体实现代码如图 4.20所示。

```
@Override
public List<Bug> parseBugList(List<Action> actionList, LinkedList<ActivityNode>
testPath, List<Logcat> logcatList) {
    List<Bug> bugList = new ArrayList<>();
    for (Logcat logcat : logcatList) {
        Bug bug = new Bug();
        Long logcatTime = logcat.getTime();
        // 通过时间戳与测试路径匹配获取缺陷发生页面
        ActivityNode currentNode = findCurrentNode(logcatTime, testPath);
        ActivityNode nextNode = null;
        int currenNodeIndex = testPath.indexOf(currentNode);
        // 缺陷当前节点不为最后一个即将路径中下一个节点设为缺陷发生后页面
        if (currenNodeIndex + 1 < testPath.size()) {
            nextNode = testPath.get(currenNodeIndex + 1);
        }
        bug = bindActivityToBug(bug, testPath, currentNode);
        Long startTime = currentNode.getTime();
        Long endTime = nextNode != null ? nextNode.getTime() : -1;
        // 获取缺陷发生前后的操作序列并绑定至缺陷对象
        List<Action> preActions = findPreActions(logcatTime, actionList, startTime);
        List<Action> nextActions = findNextActions(logcatTime, actionList, endTime);
        bug.setPreActions(preActions);
        bug.setNextActions(nextActions);
        bug.setBugInfo(logcat);
        //存入数据库中
        bugDao.save(bug);
        bugList.add(bug);
    }
    return bugList;
}
```

图 4.20: 解析缺陷对象方法代码

Bug对象存入MongoDB中会自动为每个Bug分配String类型的ID，依据Task-ID 取出该任务所对应的所有缺陷，使用分类器模型预测出每个缺陷的类型并存入数据库中，后续可以进行统计构造BugClassifyResult更新到对应的TaskForDb对象并存入数据库中。为了防止对象嵌套层级过深，所以在BugClassifyResult中每个类别都只存储对应属于该类别的Bug的ID。解析完成后，通过状态机发出成功事件，将任务的测试状态更新为成功完成。如果云测试平台调用了报告查看的接口，则会根据订单编号查找所有的拆分出来的任务状态，根据每个任务的测试状态给出解析进度。云测试平台可以展示报告前端页面，并通过Dubbo接口调用获取基于当前所有已完成任务的缺陷报告数据。

4.4　测试缺陷报告前端渲染模块

测试缺陷报告前端渲染模块基于Angular框架进行开发，前端项目打包完成后部署于Nginx服务器中，利用Nginx实现反向代理解决前端跨域请求服务端接口及访问静态资源文件的问题，增加网站安全性的基础上提供负载均衡，提高系统稳定性。通过前端项目与服务端的解耦，前端静态化与服务端数据化，服务端提供统一的RESTful Api不同前端架构可共用，服务端也可水平扩展应对并发负载较大的情况。

用户通过浏览器客户端访问云测试平台前端项目，调用的HTTP请求经过Nginx 反向代理转发给平台服务端，若为查看日志解析结果的请求，服务端收到请求后通过Dubbo RPC接口调用缺陷报告生成服务，并封装成规定对象返回给前端。

4.4.1　交互流程设计

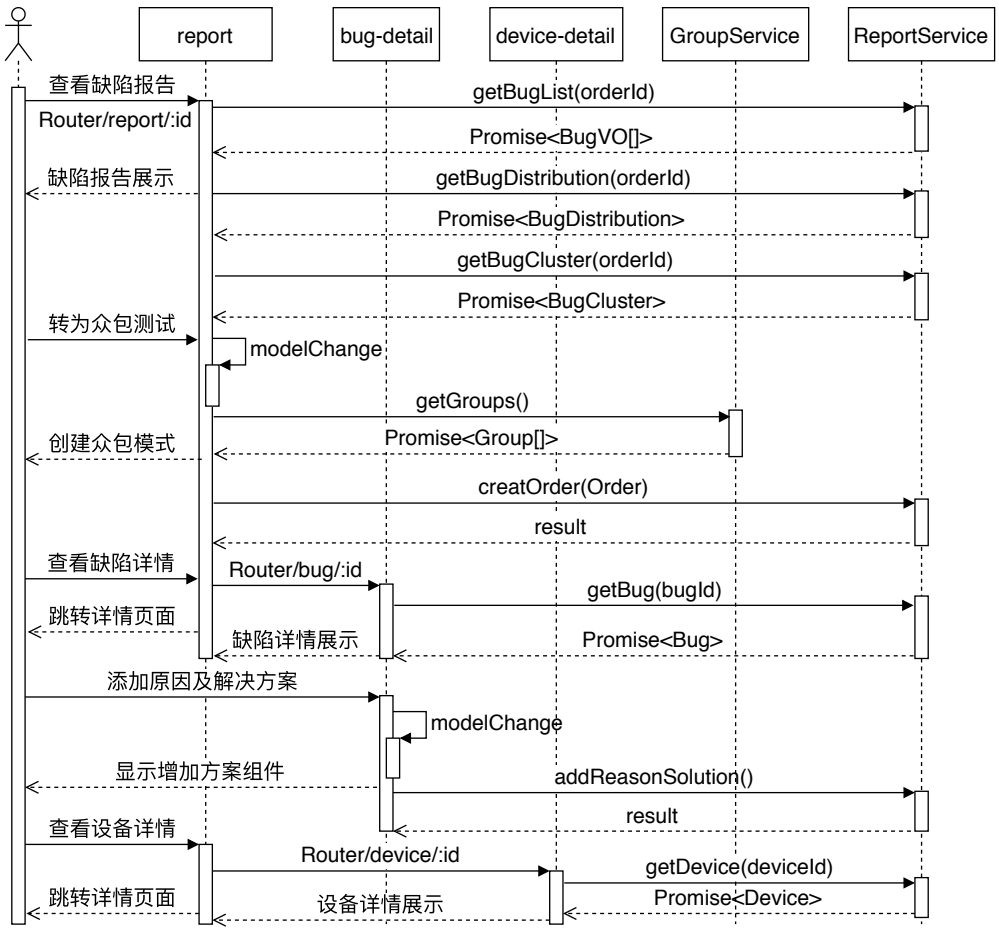

图 4.21: 缺陷报告页面顺序图

缺陷报告前端渲染模块页面顺序图如图 4.21 所示。此模块主要涉及三个前端页面，分别为报告入口页面（report）、缺陷详情页面（bug-detail）和设备详情页面（device-detail）。用户通过Angular提供的Router进行页面跳转。点击查看报告后，跳转报告入口页面，在页面初始化时（Angular 生命周期ngOnInit）通过Promise 异步解决方案获取缺陷列表（BugVO[]）、缺陷分布（BugDistribution）及问题机型聚类（BugCluster），将组件内部成员变量赋值，依据Angular双向绑定机制，在数据更新后页面随之更新。在报告入口页面提供了转为众包测试组件（create-crowdsourcing），当用户点击转为众测任务后，通过Angular指令*ngIf控制显示该组件，并对每个缺陷提供了选择操作，用户可以选择缺陷直接创建众包测进行缺陷的在线验证。

4.4.2　页面设计与实现

Angular框架开发模式如图 4.22 所示。Angular的视图由组件（component）和模板（template）组成，组件类上的装饰器通过添加元数据，关联到相关的模板；模板可以通过属性绑定获取组件中的数据源从而改变视图，也可以通过事件绑定将数据传入组件的数据源或触发组件中的方法逻辑，还可以通过双向既显示数据属性，同时再用户做出更改时更新属性。

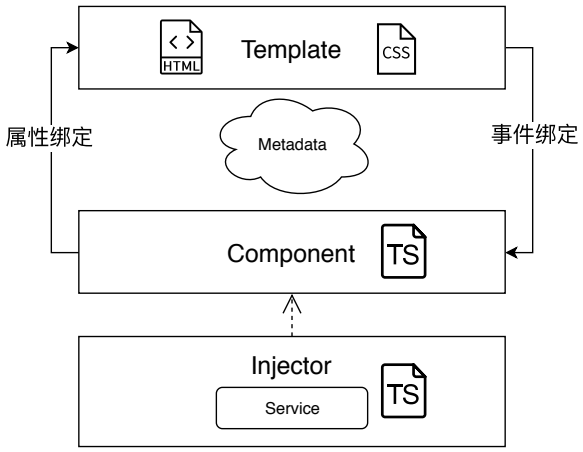

图 4.22: Angular开发模式

Angular可以通过创建服务类解耦出与特定数据无关并且希望可以跨组件共享的逻辑或者数据，如本系统中与云测试平台服务端交互获取日志解析数据的逻辑可以封装在ReportService服务类中，并可以通过依赖注入器注入到组件类中，ReportService类实现代码如图 4.23 所示，由于篇幅限制此处仅展示了获取缺陷列表和提交缺陷修复建议的服务。

```
@Injectable()
export class ReportService {
  // 构造函数传入 Angular 的 HTTP 服务，用于向服务端发送 http 请求
  constructor(private http: Http) {
  }
  …
  // 获取缺陷列表
  getBugListByOrderId(orderId: number): Promise<BugVO[]> {
    let param = new URLSearchParams;
    // 构造查询参数
    param.set('orderId', String(orderId));
    return this.http          // 依据 URL 通过 get 方法调用服务端接口
      .get('api/report/bugs', {search: param, withCredentials: true})
      .toPromise()
      .then(this.extract)
      .catch(this.handleError);
  }
  // 提交缺陷修复建议
  submitBugSuggest(bugSuggest: BugSuggest): Promise<any> {
    return this.http          // 通过 post 方法调用服务端接口
      .post('api/report/bugSuggest', bugSuggest,{ withCredentials: true})
      .toPromise()
      .then(this.extract)   // 从 http 的响应（response）中提取需要的信息
      .catch(this.handleError);
  }
  …
}
```

图 4.23: ReportService实现代码

本模块中所涉及的页面及内部组件如表 4.11 所示。主要包含了三个页面，使用Angular框架的路由功能，通过不同的URL 可以路由到不同的视图。使用路由需要先在RouterModule 中对路由信息进行配置，本模块的三个页面视图分别绑定不同的URL，其中缺陷和测试设备的详情页面的路由需要带参数的路由定义，如缺陷详情页的路由定义为path: ‘/bugDetail/:id’,component: ReportBugDetailComponent，通过路由参数将Bug的id传递到缺陷的组件中，从而可以基于id利用与后端交互的服务获取缺陷的详细信息。

所有页面都由若干子组件组成，将页面元素抽离为子组件的好处是可以进行复用，如报告页面中的bug-cluster组件，每个类型的缺陷会有一个对应的该组件，因此不需要再报告页面中重复定义，只需要利用Angular的*ngFor模板语句循环子组件，每次通过属性绑定传入不同的值即可。

表 4.11: 缺陷报告前端渲染页面及组件列表

页面	子组件
报告入口页面（report）	基本信息（app-baseinfo）
	缺陷类别分布（bug-distribution）
	问题机型聚类（bug-cluster）
	缺陷列表（bug-list）
	测试设备列表（device-list）
	转为众包测试（create-crowdsourcing）
缺陷详情页面（bug-detail）	缺陷上下文信息展示（bug-context）
	添加缺陷原因及解决方案（bug-solution）
设备列表（device-list）	设备硬件信息（device-info）
	设备测试结果详情（device-bugresult）
	性能详情（device-performance）

4.5　系统测试与实验评估

4.5.1　测试目标与测试环境

本系统的软件测试主要针对系统进行有效性测试，基于真实生产环境进行测试设计与测试执行，保障软件质量，减少软件缺陷。测试的主要目标为系统提供的功能，包括创建、审核、执行、停止移动应用在线自动化测试任务、用户构建缺陷分类器模型、查看移动应用自动化测试报告、测试报告缺陷转为众包测试进行验证以及缺陷修复方案的提交与审核等功能。通过执行测试设计的测试用例确保系统线上正常稳定运行。

表 4.12: 系统测试硬件与软件环境

服务	硬件与软件环境
云测试平台服务端	阿里云服务器，型号ecs.sn1ne.xlarge，Ubuntu 16.04，1台
缺陷报告生成服务	Ubuntu 16.04，4G内存10M宽带，1台
Web服务	JDK 1.8.0_131、Tomcat 8.0.53、Maven 3.3.9
数据库服务	MySQL 5.7.18、MongoDB 4.0.6
反向代理服务	Nginx 1.10.3
注册中心服务	Zookeeper 3.4.8
缓存服务	Redis 5.0.3
持续集成服务	Jenkins 2.156、Docker 17.09.0
测试设备	华为、小米等主流安卓手机20台

系统测试环境如表 4.12所示，详细描述了系统部署的软硬件版本。云测试平台前端与服务端均部署于阿里云服务器，通过Nginx 提供反向代理实现前后端交互，通过Zookeeper作为注册中心负责微服务的注册与发现。缺陷报告生成

服务部署于移动应用自动化测试机柜服务器中，通过Dubbo框架为云测试平台提供服务接口。自动化测试工具部署于测试机柜服务器中，测试机柜服务器与测设设备通过USB 连接，所有设备均打开USB 调试模式。云测试平台服务端使用MySQL数据库，缺陷报告生成服务使用MongoDB数据库，并使用Redis作为缓存服务用于存储用户session等一些需要在服务器之间共享的临时数据和需要快速加载的数据如缺陷分类器模型等。云测试平台服务端与缺陷报告生成服务均使用Maven负责依赖管理和项目打包，使用Tomcat 作为底层运行容器。项目通过Docker生成镜像进行打包部署并使用Jenkins进行持续集成。

4.5.2　单元测试

单元测试是针对软件设计中最小单元的正确性进行检验的测试工作，单元测试不仅可以验证当前代码是否存在问题，还可以降低项目开发后期因业务变更、修复缺陷、系统重构等代码变更带来的风险。移动应用缺陷报告自动生成系统使用Junit 4框架进行单元测试，并利用Jenkins构建可执行文件时生成单元测试报告。

系统采用微服务架构，云测试平台与缺陷报告生成服务分别作为独立的微服务进行部署并分别进行单元测试。由于系统采用分层架构，本系统单元测试依照各层不同的业务特性进行针对性的测试设计，如Dao层负责与数据库等数据源进行交互，故重点对该层代码的数据连通性以及语法正确性进行测试。Service层主要负责粒度较细的底层业务，故重点对判定覆盖和代码覆盖进行测试，Logic负责参数校验与服务调用，故重点验证参数校验逻辑，Controller负责与前端和其他服务进行交互，故重点需要对参数合法性校验进行测试。

云测试平台的自动化测试任务分发与报告渲染模块的单元测试报告如图4.24所示，服务采用分层架构，分别编写单元测试用例对各层的最小可测单元进行测试。报告显示共运行131个单元测试且全部通过。

图 4.24: 云测试平台任务分发及报告渲染模块单元测试

缺陷报告生成服务单元测试如图 4.25所示，该服务采用同样的分层架构，单元测试报告显示共113个单元测试且全部通过，证明本服务可靠性较高。

图 4.25: 缺陷报告生成服务单元测试

4.5.3　功能测试

系统功能测试用例依据第三章中阐述的系统用例进行设计，高度覆盖各项功能需求以确保项目符合产品预期，能够稳定支撑云测试平台的移动应用在线自动化测试业务。

创建移动应用自动化测试任务的测试用例描述如表 4.13所示，针对系统用例UC1涉及的功能需求进行设计，使用真实移动应用与测试设备执行测试用例确保功能可靠性，平台管理员与普通用户均有权限创建自动化测试任务，创建任务完成后需要平台管理员审核并通过后才会执行。

表 4.13: 创建移动应用自动化测试任务测试用例

测试编号	TC1
测试名称	创建移动应用自动化测试任务
测试功能	普通用户创建自动化测试任务
测试步骤	1. 普通人员账号登录； 2. 点击自动化测试； 3. 上传待测应用安装包文件； 4. 选择深度遍历测试服务； 5. 填写项目名称“移动应用自动化测试”，选择在线空闲20台测试设备； 6. 点击提交任务。
预期结果	1. 登录成功； 2. 进入创建自动化测试任务流程； 3. 提示上传成功，可以点击下一步； 4. 选择后即可点击下一步； 5. 点击选择设备出现当前在线设备列表，可以批量选择； 6. 提示提交成功，在当前用户“任务列表”中存在该任务，任务状态为“待审核”。

审核移动应用自动化测试任务的测试用例描述如表 4.14所示，主要针对系统用例UC2涉及的功能进行设计，执行本测试前需要提前创建两个状态为“待审核”的自动化测试任务分别用于测试审核通过和审核拒绝。

表 4.14: 审核移动应用自动化测试任务测试用例

测试编号	TC2
测试名称	审核移动应用自动化测试任务
测试功能	平台管理员审核移动应用自动化测试任务
测试步骤	1. 平台管理员账号登录，点击侧边栏“任务审核”； 2. 点击测试进度下拉框，筛选测试进度为“待审核”任务； 3. 选择其中一个的任务，查看任务详情； 4. 点击已选机型查看测试设备列表，点击通过； 5. 选择另一个状态为“待审核”的任务，点击拒绝。
预期结果	1. 登录成功，显示任务列表，并按时间倒序排序； 2. 任务列表按创建时间倒叙显示所有测试进度为“待审核”的测试任务； 3. 进入任务详情页面； 4. 显示任务选择的所有测试设备列表，任务状态更新为“审核通过”； 5. 任务状态更新为“审核不通过”。

执行与停止移动应用自动化测试任务的测试用例描述如表 4.15所示，主要针对系统用例UC3和用例UC4 涉及的功能进行设计，执行本测试前需要在系统中已存在测试进度为“审核通过”的测试任务，本测试用例涉及的执行与停止测试任务仅平台管理员账号拥有该权限。

表 4.15: 执行与停止移动应用自动化测试任务测试用例

测试编号	TC3
测试名称	执行与停止移动应用自动化测试任务
测试功能	平台管理员执行、停止移动应用自动化测试任务
测试步骤	1. 登录平台管理员账号； 2. 选择测试进度为“审核通过”的自动化测试任务； 3. 点击“查看任务详情” 4. 点击“执行”按钮； 5. 点击“停止”按钮。
预期结果	1. 登录成功； 2. 任务列表仅显示测试进度为“审核通过”的测试任务； 3. 进入任务详情页面，并显示任务状态为“审核通过”； 4. 任务状态更新为“测试中”，“执行”按钮更新为“停止”按钮； 5. 任务状态更新为“任务已停止”

查看移动应用自动化测试缺陷报告的测试用例描述如表 4.16所示，针对系统用例UC5进行涉及的功能进行设计，执行本测试前需要在系统中已存在测试

进度“测试完成”的任务，仅平台管理员和当前任务的创建者拥有权限查看自动化测试缺陷报告。

表 4.16: 查看移动应用自动化测试缺陷报告测试用例

测试编号	TC4
测试名称	查看移动应用自动化测试缺陷报告
测试功能	任务创建者查看移动应用自动化测试缺陷报告
测试步骤	1. 登录测试任务创建者账号，点击“我的任务”； 2. 点击任务测试进度栏下拉框，选择“测试完成”； 3. 选择任务，点击查看详情； 4. 点击查看报告； 5. 分别点击“测试概况”、“缺陷列表”、“设备列表”； 6. 点击“缺陷列表”tab页并选择一个缺陷点击“查看详情”。
预期结果	1. 按照创建时间倒序显示任务列表； 2. 任务列表仅显示测试进度为“测试完成”的任务； 3. 进入任务详情页面，并显示测试进度为“测试完成”； 4. 进入测试缺陷报告入口页面，显示测试报告的测试app、测试时间、测试设备数等基本信息，并显示“测试概况”、“缺陷列表”、“设备列表”三个tab ； 5. “测试概况”显示缺陷分布状况与问题机型聚类页面组件，“缺陷列表”显示产生所有缺陷的列表并提供筛选功能，“设备列表”显示所有进行测试的移动设备信息； 6. 进入缺陷详情页面，详细描述该缺陷上下文信息、产生原因分析和修复方案推荐。

选择测试报告中的缺陷转为众包测试测试用例详细描述如表 4.17所示，针对系统用例UC5 涉及功能进行设计，执行测试前需要在系统中已存在测试进度为“测试完成”的任务，并已创建接收缺陷验证的众包项目组。

表 4.17: 选择缺陷转为众包测试测试用例

测试编号	TC5
测试名称	选择缺陷转为众包测试
测试功能	自动化测试任务创建者选择报告中缺陷转为众包测试
测试步骤	1. 点击查看报告； 2. 点击缺陷列表tab页； 3. 点击“转为众测”； 4. 勾选全部缺陷； 5. 填写众测任务名称、选择测试项目组。
预期结果	1. 进入测试报告页面； 2. 显示测试缺陷列表； 3. 缺陷列表显示复选框，并可以进行勾选； 4. 缺陷前选择状态为已选择； 5. 在该账号创建的众包测试任务列表中存在上述任务。

选择并构建缺陷分类器模型测试用例详细描述如表 4.18所示，针对系统用例UC7涉及功能进行设计，执行前需要在缺陷报告生成服务的MongoDB 数据库中存在已标注的缺陷训练集，仅平台管理员账号拥有该权限。

表 4.18: 选择并构建缺陷分类器模型测试用例

测试编号	TC6
测试名称	选择并构建缺陷分类器模型测试
测试功能	平台管理员选择并构建缺陷分类器模型测试
测试步骤	1. 登录平台管理员账号； 2. 点击查看缺陷分类模型； 3. 点击构造缺陷分类器模型； 4. 选择分类器模型。
预期结果	1. 登录成功； 2. 进入分类器构造页面； 3. 默认构造基于C4.5算法和朴素贝叶斯算法的分类器模型，并展示评估数据，包括查准率、查全率、真阳性率等。 4. 选择成功，若不选择默认使用分类效果更优的分类器模型。

提交缺陷修复建议并审核测试用例详细描述如表 4.19所示，针对系统用例UC8和UC9涉及的功能进行设计，执行本测试前需要在系统中已存在状态为“测试完成”的自动化测试任务，且该任务的待测应用经过自动化测试捕获到缺陷，缺陷报告中存在相应的缺陷。

表 4.19: 提交缺陷修复建议并审核测试用例

测试编号	TC7
测试名称	提交缺陷修复建议并审核测试
测试功能	普通用户提交缺陷修复建议并由平台管理员审核测试
测试步骤	1. 登录测试任务创建者账号查看缺陷报告； 2. 点击缺陷列表； 3. 选择缺陷点击查看详情； 4. 点击添加修复意见； 5. 填写缺陷产生原因及解决方案，点击提交； 6. 切换管理员账号登录，查看待审核缺陷，点击通过。
预期结果	1. 登录成功，进入缺陷报告页面； 2. 显示缺陷列表； 3. 进去缺陷详情页面,显示包含缺陷详细上下文信息以及系统提供的缺陷发生原因及解决方案推荐； 4. 显示填写缺陷产生原因和解决方案的输入框 5. 提交成功； 6. 通过审核，缺陷分类器重新建模，更新评估数据。

测试人员在生产环境严格按照上述测试用例步骤执行测试，表 4.20记录了各测试用例的执行结果，所有测试用例均符合预期结果，证明系统完整完成了系统需求分析中提出的所有功能需求与设计并完整实现系统，可以稳定支撑云测试平台的自动化测试功能。

表 4.20: 测试用例执行结果

测试编号	用例编号	测试结果
TC1	UC1	通过
TC2	UC2	通过
TC3	UC3、UC4	通过
TC4	UC5	通过
TC6	UC7	通过
TC7	UC8、UC9	通过

4.5.4　实验评估与运行展示

移动应用缺陷报告自动生成系统在公司服务器部署成功后，为了验证系统在真实场景下的可用性与可靠性，本节设计并执行移动应用缺陷报告自动生成实验进行评估。首先通过开源安卓应用发布平台F-Droid[2]选取8款不同类型应用作为待测应用以全面评估系统的功能，考虑到应用可能存在与特定系统版本或型号的设备相关的兼容性缺陷，本次实验在创建测试任务时选用20台不同型号的主流安卓设备作为自动化测试机群，自动化测试工具执行完成后生成相应的缺陷报告。实验中为了验证缺陷的真实性和分类准确性，将缺陷报告内所有缺陷转为众包测试任务，借助公司云测试平台的众包测试功能，由众包工人验证报告中的缺陷是否是真正的缺陷，并审核系统对缺陷分类的准确性。

表 4.21: 移动应用缺陷报告生成结果

名称	版本	测试设备数	捕获缺陷数	真实缺陷率	分类准确率	报告生成耗时
咕咚翻译	1.8.0	20	13	92.3%	84.6%	15min
记乐部	1.0.1	20	23	95.7%	91.3%	17min
Bilibili	2.3.0	20	9	100.0%	88.9%	21min
Ve2X	1.2.5	20	6	100.0%	100.0%	18min
IThouse	1.0.0	20	17	88.2%	82.3%	13min
GnuCash	2.3.0	20	5	100.0%	100.0%	20min
2048	2.2.0	20	7	85.7%	85.7%	16min
Amaze	3.3.2	20	11	90.9%	81.8%	19min

[2]https://f-droid.org

实验结果如表 4.21所示，缺陷报告自动生成系统从8款移动应用检测出91个缺陷，其中缺陷报告的缺陷真实率平均为93.4%，分类准确率平均为87.9%。每个应用对应的缺陷报告自动生成平均耗时17分钟，此处统计的报告生成时间为用户提交待测应用到能够查看完整缺陷报告的总体时间，其中包含自动化测试工具执行遍历测试的时间，因此应用复杂程度越高报告生成时间也越长，如表中Bilibili视频应用页面和组件较多，因此报告生成耗时最长。通过上述实验结果表明本系统可以在较短时间内自动生成高准确率的缺陷报告，能够稳定支撑云测试平台的移动应用自动化测试功能，极大提高移动应用缺陷定位效率。

系统通过前端页面将缺陷报告中主要的数据进行可视化，以直观的图表形式提高缺陷报告的可读性。自动化测试任务完成后用户在相应任务详情页面点击查看报告即可跳转至缺陷报告入口页面。

图 4.26: 缺陷报告入口页面

缺陷报告入口页面如图 4.26所示，此处以记乐部应用的缺陷报告为例进行阐述，该应用为一款记账工具类安卓应用。页面展示测试任务基本信息，包括测试应用名称、任务测试时间、测试设备数目等，在测试概况中展示了各类缺陷在多平台测试中的分布状况，包括缺陷类别分布和严重等级分布。从图中可以直观地看到该应用在20台测试设备上进行自动化测试过程中共产生6种类型缺陷，其中占比最大的为JSON解析异常，点击分布图中该类别扇形部分即可查看该类缺陷的详细情况，包括发生该类别的机型、报错的堆栈信息等，下文中描述的缺陷详情中会提供该类缺陷的修复方案。

图 4.27: 问题机型聚类页面

问题机型聚类展开图如图 4.27所示，分别从测试设备品牌、系统版本、设备分辨率三个维度展示该类缺陷的分布状况，图中此应用在7台移动设备上均发生文件未找到类型的缺陷，其中系统版本为Android 6.0的设备占比最高，因此移动应用开发人员可以通过此类信息对特定移动设备进行针对性测试以提高应用的兼容性，尽早发现潜在的应用缺陷。

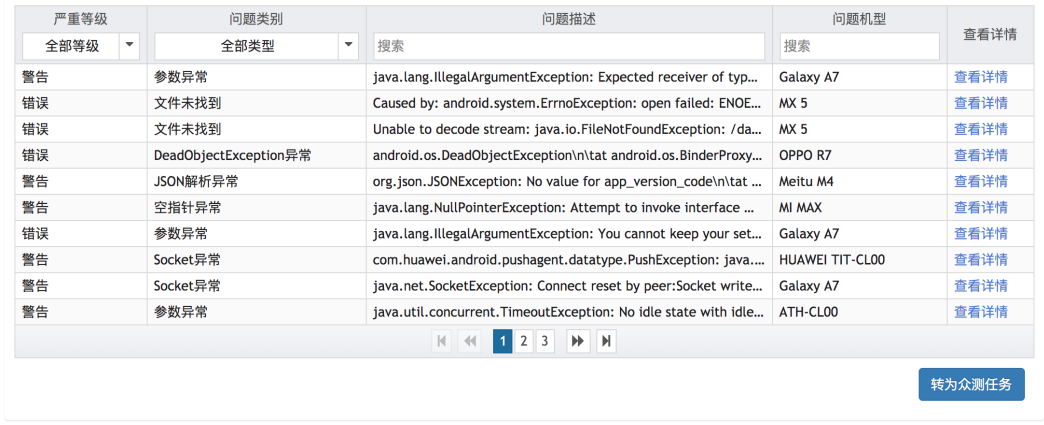

图 4.28: 缺陷列表页面

缺陷列表页面如图 4.28所示，展示该应用在所有测试设备上发生的所有缺陷，并提供缺陷严重等级、问题类别、问题描述以及产生机型等筛选条件方便快速定位。此处提供将应用缺陷转为众包测试进行验证的功能，点击“转为众测任务”按钮后，页面如图 4.29所示，所有缺陷前增加复选框可以进行选择，补充创建众包测试的配置项，如接受缺陷验证的项目组、项目开始和结束时

间、项目介绍等，即可发布关于已选缺陷验证的众包测试任务，由测试组内的众包工人审核缺陷报告中缺陷是否是真正的缺陷。

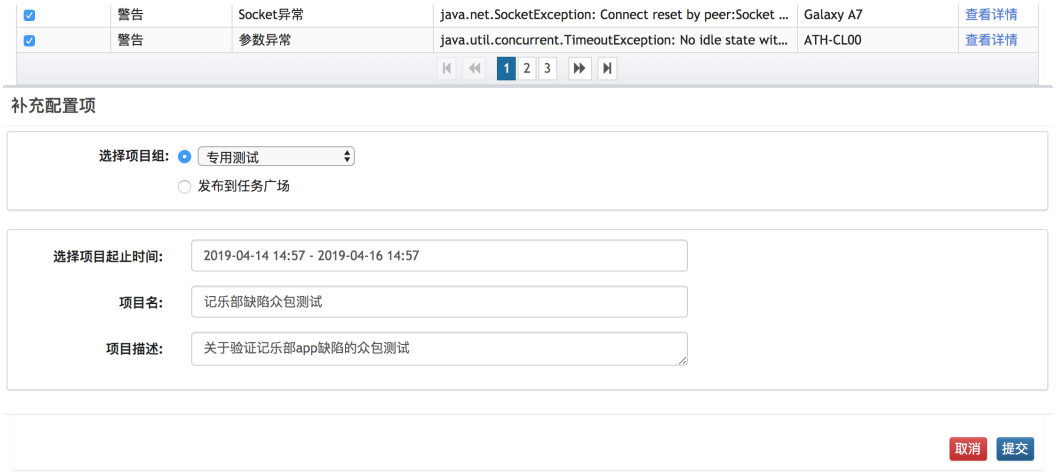

图 4.29: 缺陷转为众包测试页面

点击缺陷列表中查看详情按钮，跳转至对应缺陷的详情页面，如图 4.30所示。在缺陷上下文的操作序列中分别记录了缺陷发生前、后的操作链，并依据操作坐标在相应截图中使用红色线框进行标注，准确记录操作区域和操作类型，便于用户验证该缺陷时复现缺陷。在缺陷详细信息中记录缺陷堆栈信息、发生时间、产生缺陷的组件等基本信息并关联缺陷发生时页面截图，从而构成该缺陷的详细上下文信息。

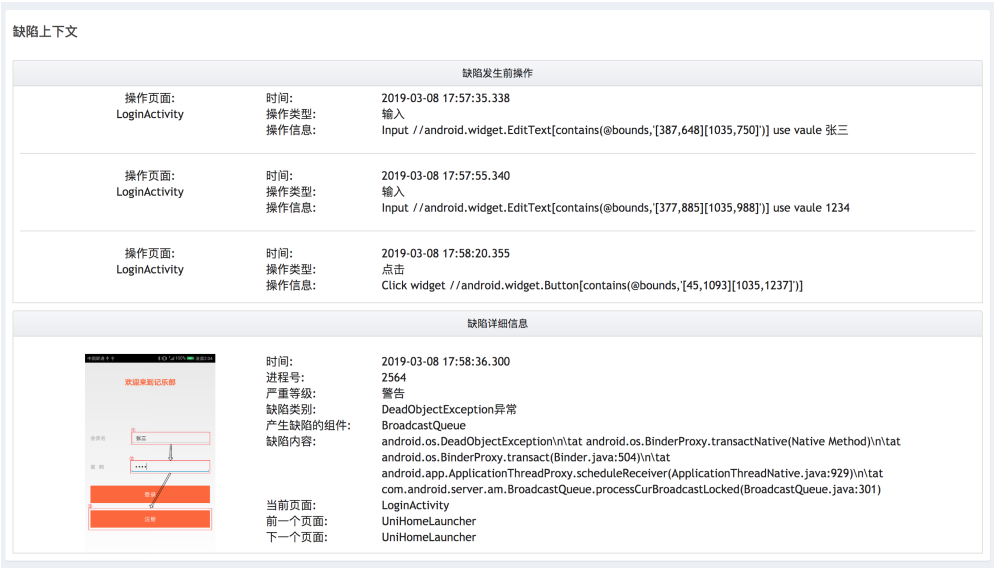

图 4.30: 缺陷上下文展开图页面

在缺陷详情页面针对该缺陷系统会提供缺陷发生原因以及修复方案供用户参考，如图 4.31所示。为了丰富系统缺陷修复方案库，在本页面还提供用户提交个人修复方案的入口，用户提交修复方案后，交由平台管理员审核通过后加入方案库，如果再次出现该类缺陷会推荐更新后的修复方案，从而实现修复方案知识共享。如果用户点击同意方案，管理员审核过便可以加入系统缺陷训练集，丰富训练集提高缺陷分类器模型的分类能力。

图 4.31: 缺陷发生原因及解决方案推荐页面

4.6　本章小结

本章介绍了系统的任务创建与分发模块、缺陷分类器建模模块、日志解析与缺陷构建模块、缺陷报告前端渲染模块的详细设计与实现，通过时序图阐述各模块的流程设计与实现、通过类表与类图阐述各模块数据与类设计，服务端详细介绍系统实现涉及的设计模式与算法实现，前端介绍页面与组件的设计与实现,并展示部分关键代码。设计并执行单元测试和功能测试证明系统可靠性，通过移动应用缺陷报告生成实验评估系统可用性，并展示系统部分运行图。

第五章　总结与展望

5.1　总结

随着移动应用开发周期越来越短，自动化测试以测试效率高的优势逐渐取代传统人工测试成为移动应用主流测试方式。为了解决自动化测试中日志可读性低，人工填写缺陷报告效率低下的问题，同时为了将自动化测试产出的多元日志进行深度挖掘并自动生成高可读性缺陷报告，本文设计并实现了移动应用缺陷报告自动生成系统。

本文首先阐述了系统的项目背景以及国内外相关方向的研究现状，针对国内主流云测试平台测试报告可读性不高的问题提出了本系统的改进方向，然后介绍了实现系统过程中使用的主要技术，包括用于构成微服务架构的Dubbo框架和Zookeeper注册中心、服务端开发框架Spring Boot、前端开发框架Angular、用于数据持久化的MongoDB数据库、用于数据缓存以提高数据访问速度的Redis缓存、容器化工具Docker、持续集成工具Jenkins以及用于缺陷分类的C4.5决策树算法和朴素贝叶斯算法。

本文在第三章对系统的功能性和非功能性需求进行分析后，通过软件架构设计中的场景视图、逻辑视图、开发视图、物理视图和过程视图（4+1视图）阐述了系统的总体设计，并依据功能需求对系统分进行模块划分，包含测试任务的创建与分发、缺陷分类器建模、日志解析与缺陷构建和缺陷报告前端渲染四个模块，并在第四章通过时序图、类图、类表、展示关键代码等方式对每个模块的详细设计和实现进行了阐述，基于真实生产环境设计并执行单元测试和功能测试验证系统可靠性，并展示了系统在公司上线后的实际运行效果图。

目前系统已经在公司上线并稳定支撑云测试平台的移动应用缺陷报告自动生成功能，完善了平台在线自动化测试服务，移动应用开发者只需要上传待测试应用的安装包文件并配置需要进行测试的机型便可以在云测试平台进行自动化测试并获取该应用的缺陷报告。系统通过解析自动化测试日志构建出包含完整上下文信息的缺陷模型，包括缺陷发生前后的操作序列、缺陷发生时日志截图、缺陷堆栈详细信息等，并对缺陷进行分类和去重，以防止开发人员重复审核相同缺陷。缺陷报告汇总各类缺陷在各个测试机型的分布状况，通过前端页面进行数据可视化，生成可读性高、具有指导价值的缺陷报告，极大提高开发人员定位、复现和修复缺陷的效率。本系统的实现提高了公司云测试平台的价值，为公司业务发展做出积极贡献。

5.2 展望

目前移动应用缺陷报告自动生成系统已经完成开发并在公司服务器完成部署，运行稳定，但依然存在一些不足之处与后续可以改进的方面。

第一，目前公司用于在线自动化测试的移动应用测试机群已经涵盖市场主流机型，但是随着移动设备快速地更新换代，仍需要不断补充新的机型作为测试设备，从而提高移动应用缺陷报告的时效性。

第二，当前云测试平台主要提供针对安卓系统的自动化测试服务，缺陷报告生成系统也主要提供的是安卓应用的缺陷报告，但是系统将缺陷捕获和构建的相关逻辑封装为通用的接口，使得平台具备较好的可扩展性，后续集成IOS平台应用测试的缺陷报告也较为方便。

第三，系统目前对于各类缺陷推荐的修复方案还不够全面和完善，系统提供了用户反馈应用缺陷修复方案的功能，因此随着系统上线后，平台用户的增多以及长时间的方案反馈积累，可以不断更新各类缺陷全面的修复方案供用户进行参考，提高缺陷报告对用户的指导价值。

参考文献

[1] H. Muccini, A. D. Francesco, P. Esposito, Software Testing of Mobile Applications: Challenges and Future Research Directions, in: Proceedings of the 7th International Workshop on Automation of Software Test, 2012, pp. 29–35.

[2] W. Jun, F. Meng, Software Testing Based on Cloud Computing, in: Proceedings of 2011 International Conference on Internet Computing and Information Services, 2011, pp. 176–178.

[3] J. Gao, X. Bai, W. Tsai, T. Uehara, SaaS Testing on Clouds - Issues, Challenges and Needs, in: Proceedings of the 7th International Symposium on Service-Oriented System Engineering, 2013, pp. 409–415.

[4] O. Starov, S. Vilkomir, V. Kharchenko, Cloud Testing for Mobile Software Systems, in: Proceedings of the 8th International Joint Conference on Software Technologies, 2013, pp. 124–131.

[5] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, G. Candea, Cloud9: A Software Testing Service, Operating Systems Review 43 (4) (2009) 5–10.

[6] J. Kaasila, D. Ferreira, V. Kostakos, T. Ojala, Testdroid: Automated Remote UI Testing on Android, in: Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia, 2012, p. 28.

[7] K. Moran, M. L. Vásquez, C. Bernal-Cárdenas, D. Poshyvanyk, Auto-Completing Bug Reports for Android Applications, in: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 673–686.

[8] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss, What Makes a Good Bug Report?, IEEE Transactions on Software Engineering 36 (5) (2010) 618–643.

[9] K. Moran, R. Bonett, C. Bernal-Cárdenas, B. Otten, D. Park, D. Poshyvanyk, On-Device Bug Reporting for Android Applications, in: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, 2017, pp. 215–216.

[10] K. Moran, M. L. Vásquez, C. Bernal-Cárdenas, C. Vendome, D. Poshyvanyk, Automatically Discovering, Reporting and Reproducing Android Application Crashes, in: Proceedings of 2016 International Conference on Software Testing, 2016, pp. 33–44.

[11] S. Li, X. Huang, Dubbo's Serialization Protocol Extension and Optimization of Its RPC Protocol Thrift, in: Proceedings of the 8th International Conference on Software Engineering and Service Science, 2017, pp. 721–726.

[12] L. Zheng, B. Wei, Application of Microservice Architecture in Cloud Environment Project Development, in: Proceedings of 2018 MATEC Web of Conferences, Vol. 189, 2018, p. 03023.

[13] C. Artho, Q. Gros, G. Rousset, K. Banzai, L. Ma, T. Kitamura, M. Hagiya, Y. Tanabe, M. Yamamoto, Model-Based API Testing of Apache Zookeeper, in: Proceedings of 2017 International Conference on Software Testing, 2017, pp. 288–298.

[14] A. Balalaie, A. Heydarnoori, P. Jamshidi, Migrating to Cloud-Native Architectures Using Microservices: An Experience Report, in: Proceedings of 2015 Advances in Service-Oriented and Cloud Computing, 2015, pp. 201–215.

[15] E. Wolff, Microservices: Flexible Software Architecture, Addison-Wesley Professional, 2016.

[16] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil, Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud, in: Proceedings of the 10th Computing Colombian Conference, 2015, pp. 583–590.

[17] H. Suryotrisongko, D. P. Jayanto, A. Tjahyanto, Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot, Procedia Computer Science 124 (2017) 736–743.

[18] M. Macero, Learn Microservices with Spring Boot, Springer, 2017.

[19] F. Gutierrez, Pro Spring Boot, Springer, 2016.

[20] J. L. Carlson, Redis in Action, Manning Publications Co., 2013.

[21] V. Das, Learning Redis, Packt Publishing Ltd, 2015.

[22] K. Banker, MongoDB in Action, Manning Publications Co., 2011.

[23] Z. Parker, S. Poe, S. V. Vrbsky, Comparing NoSQL MongoDB to An SQL DB, in: Proceedings of the 51st ACM Southeast Conference, 2013, pp. 5:1–5:6.

[24] N. Murray, F. Coury, A. Lerner, C. Taborda, Ng-book: The Complete Guide to Angular, CreateSpace Independent Publishing Platform, 2018.

[25] R. Garofalo, Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern, Microsoft Press, 2011.

[26] C. Kan, DoCloud: An Elastic Cloud Platform for Web Applications Based on Docker, in: Proceedings of the 18th International Conference on Advanced Communication Technology, 2016, pp. 478–483.

[27] H. Kang, M. Le, S. Tao, Container and Microservice Driven Design for Cloud Infrastructure DevOps, in: Proceedings of 2016 International Conference on Cloud Engineering, 2016, pp. 202–211.

[28] M. Fowler, M. Foemmel, Continuous Integration, Thought-Works 122 (2006) 14.

[29] 刘小虎, 李生, 决策树的优化算法, 软件学报9 (10) (1998) 797–800.

[30] J. R. Quinlan, C4.5:Programs for Machine Learning, Morgan Kaufmann, 1993.

[31] J. R. Quinlan, Simplifying Decision Trees, International Journal of Man-Machine Studies 27 (3) (1987) 221–234.

[32] 徐鹏, 林森, 基于C4.5决策树的流量分类方法, 软件学报20 (10) (2009) 2692–2704.

[33] H. Elaidi, Z. Benabbou, H. Abbar, A Comparative Study of Algorithms Constructing Decision Trees: ID3 and C4.5, in: Proceedings of 2018 International Conference on Learning and Optimization Algorithms: Theory and Applications, 2018, pp. 26:1–26:5.

[34] M. E. Maron, J. L. Kuhns, On Relevance, Probabilistic Indexing and Information Retrieval, J. ACM 7 (3) (1960) 216–244.

[35] D. D. Lewis, Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval, in: Proceedings of the 10th European Conference on Machine Learning, 1998, pp. 4–15.

[36] H. Zhang, The Optimality of Naive Bayes, in: Proceedings of the 7th International Florida Artificial Intelligence Research Society Conference, 2004, pp. 562–567.

[37] T. R. Patil, S. Sherekar, Performance Analysis of Naive Bayes and J48 Classification Algorithm for Data Classification, International Journal of Computer Science and Applications 6 (2) (2013) 256–261.

[38] C. Hu, I. Neamtiu, Automating GUI Testing for Android Applications, in: Proceedings of the 6th International Workshop on Automation of Software Test, 2011, pp. 77–83.

[39] J. B. Lovins, Development of A Stemming Algorithm, Mech. Translat. & Comp. Linguistics 11 (1-2) (1968) 22–31.

[40] Z. Markov, I. Russell, An Introduction to the WEKA Data Mining System, ACM SIGCSE Bulletin 38 (3) (2006) 367–368.

[41] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, Soviet physics doklady 10 (8) (1966) 707–710.

简历与科研成果

基本情况 李沛源，男，汉族，1995 年 10 月出生，江苏省南京市人。

教育背景

2017.9～2019.6 南京大学软件学院 硕士

2013.9～2017.6 华中科技大学软件学院 本科

参与项目

1. 国家重点研发计划课题：基于协同编程现场的智能实时质量提升方法与技术（2018YFB1003901），2018-2021
2. 国家新技术实验室创新项目-面上项目：基于群体智能的移动应用自动化测试研究（ZZKT2017B09），2017-2019

致　　谢

云测试平台系统的开发工作作为我两年的研究生学习期间重要的一部分，能够从最初的设计到现在的实现，离不开老师、同学、亲友的支持和帮助，在此希望对他们致以最诚挚的感谢。

首先需要感谢我的研究生导师陈振宇教授和房春荣老师，从项目最初的愿景到实现的解决方案，从工程的开发到论文的写作，他们给予了我太多的帮助。通过每次例会的探讨，在他们的指导下不断完善整体的实现思路与实现方案，在遇到困难时及时帮助我拓展其他的思路，以渊博的知识和耐心的指导帮助我完成了这项对我来说具有一定挑战的项目。感谢陈老师能够让我加入具有创新精神和极强研发氛围的ISE实验室，十分荣幸能够在提升自己的同时为实验室也贡献出自己的一份力量。

其次还要感谢实验室的黄勇老师，在他的指导下完成适合项目的具体技术选型。也十分感谢感谢实验室共同协作开发的同学，没有他们的全力支持和密切配合，我一个人也无法完成本项目的开发工作。

最后希望感谢父母和家人在我的研究生学习期间的支持和鼓励，他们给了我前进的动力，感谢过去两年所有关心和帮助我的老师、同学和好友。

感谢百忙之中抽出时间审阅论文和参加答辩的评审专家们，向你们的辛勤劳动致敬。特此感谢!

版权及论文原创性说明

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：　年　月　日