南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于微服务的安卓众包在线验证平台的设计与实现

作 者 姓 名 周顺祥

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：**MF1732189**

论文答辩日期：**2019 年 5 月 8 日**

指 导 教 师：　　（签字）

The Design and Implementation of Android Crowdsourced Online Verification Platform Based on Microservice

By

Shunxiang Zhou

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：基于微服务的安卓众包在线验证平台的设计与实现

工程硕士（软件工程领域） 专业 2017 级硕士生姓名： 周顺祥

指导教师（姓名、职称）： 陈振宇 教授

摘 要

为了提升安卓应用质量，诸多平台提供了安卓应用自动化测试服务，并能够生成包含应用缺陷信息的测试报告。受限于当前测试工具和测试脚本质量等问题，测试报告中的缺陷信息可能并不准确。开发者验证此类缺陷往往缺乏足够的人力和设备资源。众包测试能够招募大量众包工人帮助快速完成测试。因此，将缺陷信息转为众包任务并提供在线验证平台，能够提升软件缺陷发现的准确率。

本文设计与实现了一个基于微服务的安卓众包在线验证平台。该平台通过在线真机操控和缺陷众包验证两方面来解决人力与设备资源不足问题。众包工人能够远程在设备中进行测试脚本的录制回放，修改并完善测试脚本。缺陷众包验证则让任务请求者通过众包方式验证缺陷，根据众包工人提交的验证结果和结果统计分布提升效率。本平台分为设备微服务模块和众包在线验证模块。设备微服务模块直接与移动设备进行交互。众包在线验证模块则包含Web界面与服务端，与设备微服务模块交互，并保存用户缺陷验证数据到数据库中。平台在设计上利用Spring Cloud微服务框架对平台进行维护管理，解决因设备数量增多使得设备微服务模块出现多个后难以维护的难题。平台采用MiniCap和MiniTouch工具获取设备实时界面图和执行设备触屏模拟操作，通过WebSocket协议和Netty框架保持Web端与设备之间的数据交换，采用Appium框架对测试脚本进行回放，并最终提升缺陷信息的准确性。

本文选取50台安卓设备和7个有缺陷安卓应用对平台进行运行效果的初步验证。结果表明平台对Android 4.4以上版本设备具有高支持度以及通过众包验证得出的缺陷准确率平均能够达到80%左右，初步验证了平台的可用性。该平台已经在公司上线使用，有效地解决了用户测试设备与人力资源不足问题，加快了缺陷验证速度，帮助用户以更低的成本和更高的效率发现安卓应用中的质量问题。

关键词： 安卓应用，众包测试，微服务，远程操控，缺陷验证

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 Android Crowdsourced Online Verification Platform Based on Microservice

SPECIALIZATION: Software Engineering

POSTGRADUATE: Shunxiang Zhou

MENTOR: Professor Zhenyu Chen

Abstract

In order to improve the quality of Android applications, many platforms provide automated testing services for Android applications and can generate test reports containing application bug informations. Limited by the current test tools and the quality problems of test scripts, the bug informations in test reports may not be accurate. Developers often lack sufficient human and device resources to verify such bugs. Crowdsourced testing can recruit a large number of crowd workers to help complete the test quickly. Therefore, turning bug informations into a crowdsourced task and providing an online verification platform can improve the accuracy of software bug discovery.

This thesis designs and implements an Android crowdsourced online verification platform based on microservice. The platform solves the problem of insufficient human and device resources through online real machine control and bug crowdsourced verification. Crowd workers can remotely record and replay test scripts in the device, and they can modify and perfect test scripts. Bug crowdsourced verification allows task requesters to verify bugs through crowdsource and improve verification efficiency based on the results of verification submitted by crowd workers and the statistical distribution of results. The platform is divided into device microservice module and crowdsourced online verification module. The device microservice module interacts directly with the mobile devices. The crowdsourced online verification module includes a Web interface and a server, it interacts with the device microservice module and saves users' bug verification datas to the database. In the design of the platform, The platform is maintained and managed by the Spring Cloud microservice framework to solve the problem that it is difficult to maintain more and more device microservice modules due to multiple devices. The platform uses MiniCap and MiniTouch tools

to obtain the real-time interface diagram of the device and execute the touch screen simulation operation of the device. The WebSocket protocol and the Netty framework are used to maintain data exchange between the Web terminal and the device. and the platform uses the Appium framework to replay the test scripts and finally improves the accuracy of bug informations.

This thesis selects 50 Android devices and 7 Android applications which include bugs to verify the running effect of the platform. The result shows that the platform has high support for Android 4.4 or higher devices and the accuracy rate of bugs through crowdsourced verification can reach about 80% on average. This result preliminarily verifies the availability of the platform. The platform has been used in the company, it effectively solves the problem of insufficient users' test devices and human resources, accelerates the speed of bug verification, and helps users find quality problems in Android applications with lower cost and higher efficiency.

Keywords: Android Application，Crowdsourced Testing，Microservice，Remote Control，Bug Verification

目　录

表　目　录

图　目　录

第一章　引言

1.1　项目背景

随着移动互联网高速发展，安卓应用用户量在不断增加。安卓应用软件可以让人们的生活和工作更加便利，这使得人们对安卓应用的需求快速增长。安卓应用质量是决定应用成败的关键，为了确保应用质量可靠性，安卓应用产品在发布前需要进行有效测试，而为了提高测试效率，自动化测试和众包测试被广泛使用。

自动化测试把以人为驱动的测试行为转化为机器执行 [1]。安卓应用自动化测试主要是通过预设的待测安卓应用安装包以及测试脚本，由机器将安卓应用安装到进行测试的设备之中，通过自研或开源的测试工具在这些设备上执行测试脚本。在执行过程中不断获取设备信息以及应用缺陷，最后将这些信息以自动化测试报告的形式展示给用户。众包测试则是雇佣一群线上众包工人在有限周期内对测试目标进行测试和真实结果反馈 [2]，它与传统测试相比缩短了测试时间，节约了测试成本。

当前测试行业中已出现许多移动应用测试平台，它们都提供安卓应用的自动化测试功能。但它们目前都存在一个问题：受制于测试工具以及用户提交的测试脚本质量问题，报告中测试出的所有缺陷中有一部分并不准确。这让用户需要花费一段时间从报告中得到准确的缺陷信息，影响应用的缺陷修复。而用户想要在短期内对这些缺陷进行验证时会出现缺少设备与人力资源问题，难以快速获取真正缺陷。本平台所属的移动应用测试平台也存在这样问题，影响公司业务发展。因此公司计划将测试报告中的缺陷作为测试目标提交到众包测试，通过众包和提供在线真机的方式来解决用户设备与人力问题，更好地帮助用户获取准确与充分的缺陷信息。

本平台正是作为缺陷众包测试的验证平台而被设计与实现出来。在众包工人参与缺陷众包测试后，平台为他们提供在线真机操控功能来验证缺陷，真机中包含了自动化测试中出现该缺陷的设备，让众包工人能更好地复现缺陷。平台还为众包工人提供测试脚本的录制回放功能，可让其中的专家录制出质量更佳的脚本作为参考。任务请求者则可通过该平台查看所有验证结果和测试脚本，从结果列表和统计分布中得到真正的缺陷。平台通过自动化测试与众包的结合，使得安卓自动化测试报告中的缺陷更加准确和完善，帮助用户可以更快地修复真正缺陷。

1.2　国内外发展现状

1.2.1　众包验证平台发展现状

2006年，Howe首次提出众包的概念 [3]。众包是最近流行的一种技术，它将一个问题分成小块，然后由互联网上的一群用户解决 [4]。众包技术在软件工程领域得到应用，特别是在软件测试方面，众包测试已成为软件测试的一个新兴趋势，它利用了众包和云平台的优势和效率。不同于传统测试方法，众包测试由大量在线测试工人进行，可以提供对真实应用场景和真实用户表现的良好模拟，测试周期短且测试成本相对较低 [5]。众包测试技术带来的好处使得它在工程领域里越来越受到人们的关注。

如今社会上已涌现出许多关于软件测试的商业平台，如Testin[1]，百度MTC[2]，腾讯WeTest[3]，阿里云移动测试平台[4]等。其中，针对安卓手机，它们分别向用户提供安卓自动化测试和安卓众包测试服务。安卓自动化测试执行用户提交的测试脚本对安卓应用进行测试，完成后提供一份包含缺陷信息的自动化测试报告给用户 [6]。安卓众包测试则是由任务请求者创建任务，添加必要需求说明和目标应用后，由每个众包工人使用自己手机进行测试，根据已定义好的格式填写报告并提交，最后由任务请求者进行确认。但目前这些平台中这两种测试无多大联系，自动化测试结果准确性主要取决于测试工具和测试脚本的准确与否，没有相关途径通过众包测试对结果中的缺陷进行快速验证。而安卓众包在线验证平台将自动化测试与众包测试联系在一起，并提供在线真机环境。在将安卓自动化测试结果中的缺陷转为众包测试后，由众包工人通过该平台在线操控设备，进行缺陷验证，将验证结果交由任务请求者查看。通过自动化及人工方式获取真正缺陷，完善缺陷信息，以此提高测试的缺陷准确性，使得该平台在软件测试平台中拥有着重大意义。

1.2.2　微服务发展现状

随着领域驱动设计DDD、快速集成、持续交付、大型集群系统这些实践的流行，微服务应运而生 [7]。微服务的概念最早于2014年被提出，它是一种新型软件架构风格，是一种将单个应用由一套小服务来进行开发的方式途径 [8]。在微服务出现之前，许多系统都使用单体架构，将所有应用功能都部署在一起。若某个功能发生故障就会导致整个应用无法运行使用，这样的故障在一些企业

[1]https://www.testin.cn/
[2]http://mtc.baidu.com/
[3]https://wetest.qq.com/
[4]http://mqc.aliyun.com/

中很可能会造成巨大的损失，并且若以后应用功能增多，这样的架构会慢慢变得难以维护和扩展。而微服务就可以将整个系统拆分成多个独立组件分别部署，每个服务都是基于业务能力构建，能够保证资源的有效隔离 [9]。它们分别由对应负责组进行维护，保持最低限度的集中式管理。这些服务运行在各自的进程中，相互独立，使用与语言无关的如HTTP API等轻量级机制通信，其中某个微服务的崩溃不会再影响到整个系统的正常运行 [10]。微服务的出现也改变了企业研发团队组织结构，由以前分为前端、后端、数据库以及测试的水平团队组织结构慢慢向基于业务划分的团队架构倾斜。

微服务将应用进行有效地拆分，实现敏捷开发和部署。它的出现带来了许多好处，使得许多大型网站、电商平台以及游戏行业都基本使用微服务架构来解决服务化和服务治理问题 [11]。微服务框架则是将微服务架构进行应用，让用户能够简单地运用微服务开发项目。当前国内影响力比较大的微服务框架有Spring Cloud 和Dubbo框架，在国外比较流行的有Coral Service服务框架。

Spring Cloud是一套完整的微服务解决方案，它最适用于管理由Spring Boot创建的微服务应用。它集成市面上开发不错的模块并进行封装提供给用户使用，从而减少开发者的代码量，降低开发成本。它能够帮助开发人员快速搭建分布式系统中的公共组件，实现服务发现、服务路由、断路器、配置管理、服务网关等功能。

Dubbo是阿里巴巴开源的一个分布式服务框架，致力于提供高性能优秀的RPC远程服务调用方案以及SOA服务治理方案 [12]。它被广泛应用于集团内各成员站点以及其它公司业务中。它提供Multicast、Zookeeper、Redis以及Simple注册中心进行服务注册和发现，动态管理服务地址信息，并提供软负载均衡及容错机制，具有较好的健壮性。

Coral Service服务框架是亚马逊内部使用的分布式服务框架，它的功能特点有轻量级架构、配置化开发以及与亚马逊的其他基础设施集成实现DevOps等。Coral Service被确定为亚马逊全公司统一服务化开发框架，保证快速高效的服务开发，公司所有应用后端完全服务化且服务之间能进行共享和重用。而且公司系统在服务化后高度自动化，支持一键式部署和回滚。

在这些框架中，相比较而言，Spring Cloud提供的功能更全面，并且Spring Cloud能与Spring Boot项目完美融合，这使系统使用统一的技术框架，能够做到持续集成和快速交付。而本平台在Spring Cloud框架的基础上进行开发实现，这就能够使该平台更加健壮，易于管理和维护，功能扩展性也会更高。

1.3　本文主要研究的工作

本文主要工作是设计和实现基于微服务的安卓众包在线验证平台。该平台的主要目标是解决用户设备和人力资源不足问题，帮助用户快速验证缺陷，获取真正充分的缺陷信息。经过分析，本系统可主要通过在线真机操控和缺陷众包验证两个方面来实现目标。

在线真机操控功能提供给众包工人使用，通过平台提供安卓设备的在线环境来解决用户设备资源不足问题，让用户无需寻找自身设备以及配置验证环境。用户可查看所有有效设备信息，选择其中空闲设备使用后可进行设备的完整控制。浏览器界面会实时展示设备当前屏幕并即时响应用户操作。用户还可以在设备操控过程中进行测试脚本的相关操作，对测试脚本进行录制、修改、生成和回放。测试脚本录制过程中还有设备logcat日志查看功能供使用，帮助用户录制质量更佳的测试脚本。

缺陷众包验证则是在任务请求者将待验证缺陷提交到众包测试项目后，由众包工人参与并在平台上进行缺陷验证，最后让任务请求者查看结果列表来得到真正缺陷。它面向任务请求者和众包工人，让众包工人帮助任务请求者验证缺陷，解决用户自身人力不足问题。众包工人在在线真机操控中通过设备和脚本操作验证缺陷，补充缺陷信息，并选择性地将生成的脚本作为测试脚本供任务请求者参考。任务请求者则在验证结果页面中查看所有已提交的验证结果和测试脚本，并能查看缺陷验证结果以及对缺陷的复现程度、BUG分类和严重等级的统计分布。

结合上述用户需求，安卓众包在线验证平台使用微服务架构进行开发，分为设备微服务模块和众包在线验证模块，采用Spring Cloud框架对它们进行统一管理，将它们向Spring Cloud的Eureka注册中心进行服务注册，使服务调用变得更加简便。设备微服务采用Spring Boot框架，通过实现ADB 的监听器对设备进行监听，并在设备中启动MiniCap和MiniTouch工具与设备进行交互。该模块使用HTTP协议和Netty框架与众包在线验证模块进行数据交换，并通过Appium进行测试脚本回放。众包在线验证模块则采用Angular2前端框架和Spring Boot框架，提供丰富的Web端界面与用户交互。模块Web端与服务端也通过Netty和HTTP进行双向通信，并在Netty启动配置中使用WebSocket协议。

目前平台已作为公司移动应用测试平台的子平台上线使用，支撑测试平台的自动化测试和众包测试的业务发展。通过该平台，用户在获取自动化测试报告中的缺陷后能快速对它们进行验证，获取真正缺陷，提高缺陷的准确率，最终帮助用户可以更早地修复应用缺陷来提高应用质量和市场竞争力。

1.4　本文的组织结构

本文共分为六个章节，组织结构如下：

第一章 引言。主要介绍项目背景以及众包验证平台与微服务在国内外的发展现状，并阐述本文的主要研究工作。

第二章 技术综述。将项目中主要涉及的技术和框架进行了简要介绍，包括Spring Cloud，WebSocket，Netty框架，设备交互工具MiniCap、MiniTouch，Appium及Angular2框架。

第三章 安卓众包在线验证平台的需求分析与概要设计。对项目进行整体描述，对系统的功能性需求和非功能性需求结合图表的形式进行说明，并对平台的总体设计、平台中设备微服务模块和众包在线验证模块的分析设计以及相关数据库设计进行阐述。

第四章 安卓众包在线验证平台的详细设计与实现。在需求分析与概要设计的基础上，重点对设备微服务模块和众包在线验证模块中的子模块进行具体描述和实现细节阐述，以类图展示类关系，并展示关键代码。最后展示平台实际运行界面。

第五章 安卓众包在线验证平台的测试与实验设计。对平台进行单元测试和功能测试并展示测试结果。之后对平台进行实验设计，对平台运行效果进行分析，并最后给出实验结果数据，以此验证平台可用性。

第六章 总结与展望。对论文期间所做工作进行总结，分析下一步工作，并且就安卓众包在线验证平台的未来扩展做了进一步展望。

第二章　技术综述

2.1　Spring Cloud

本平台使用Spring Cloud微服务框架，与其它微服务框架相比它只需在Spring Boot项目中添加必要依赖和注解就能完美融合使用。其中平台主要使用框架中的Eureka组件，并在一些重要功能实现上使用了Ribbon和Hystrix组件，这些组件能让平台的服务管理更加方便，服务之间调用更加稳定。

Spring Cloud是一系列框架的有序集合，这些框架致力于分布式服务治理[13]，它封装了Netflix的多个开源组件，同时又实现了和Spring Boot开发框架以及云端平台的集成。Spring Cloud是一套完整的分布式系统解决方案，从服务注册与发现Eureka，到负载均衡Ribbon，再到熔断机制Hystrix，它的子项目涵盖所有实现分布式系统所需要的基础软件设施 [14]。Spring Cloud使得开发部署变得更加简单，微服务管理也变得更加轻松，它的架构图如图 2.1所示。

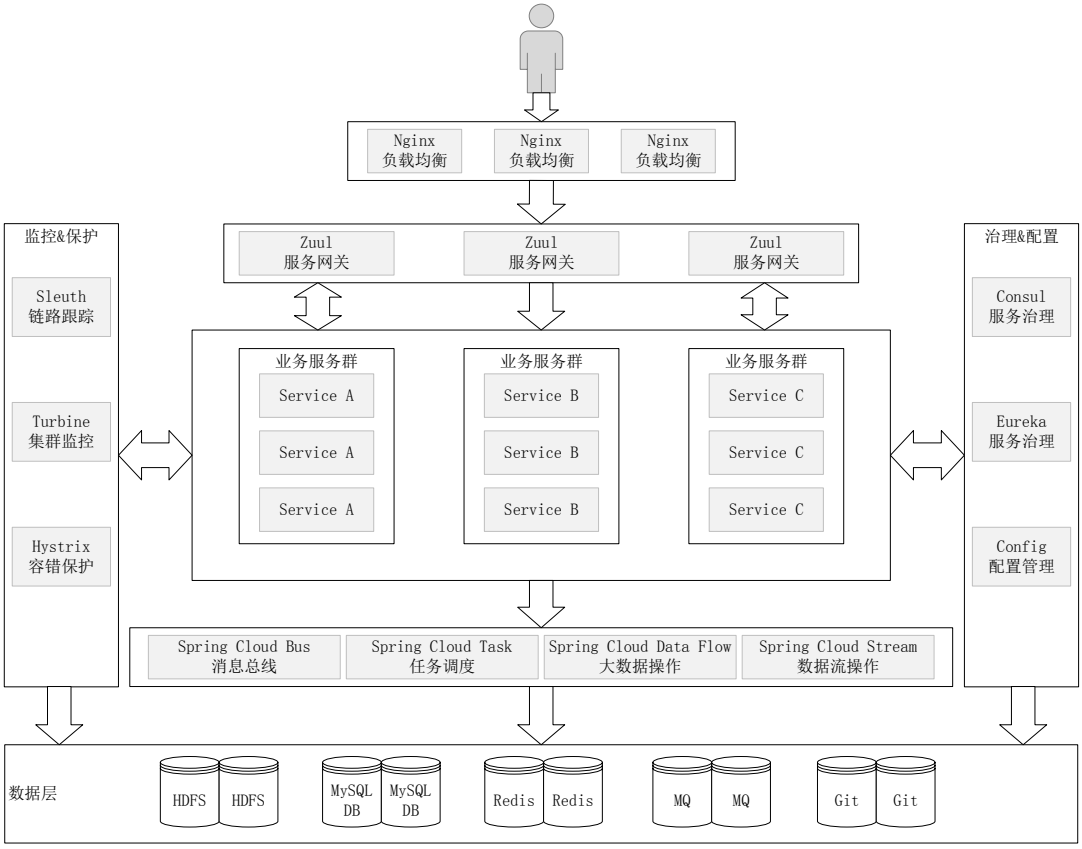

图 2.1: Spring Cloud架构图

在Spring Cloud中可使用Eureka实现服务的注册与发现。Eureka既是客户端组件也是服务端组件，在具体使用时通过注解来区分。Eureka不仅仅适合Java实现的微服务，也支持非Java语言开发的微服务 [15]，这是因为Eureka的服务端组件向开发人员提供较为完善的REST风格的API，这一特性给开发人员提供多种技术栈选择。Eureka主要有服务注册中心、服务提供者和服务消费者三种角色 [16]，其中Eureka服务注册中心负责维护注册服务列表，如果注册中心在一定时间内没有接收到某个微服务实例心跳，则会将该实例注销，当注册中心的服务数过低时，Eureka会进行自我保护。Eureka服务提供者则向注册中心做服务注册、续约、下线等操作。Eureka服务消费者则会发送一个REST请求给服务注册中心来获取注册的服务清单，并通过服务名获取具体提供服务实例名和该实例的元数据信息，最后根据自己需要决定调用哪个实例。

Ribbon是一款云中间层服务的开源项目，主要提供客户端负载均衡算法 [17]。Ribbon组件提供一系列完善的配置项，用户可根据需要自行选择。用户可通过开启@LoadBalanced注解，Ribbon会自动根据某种规则（如简单轮询等）去访问服务提供者，此外用户还可以使用Ribbon实现自定义的负载均衡算法。

在微服务架构中，各个服务之间通过注册与订阅方式相互依赖，服务之间由于网络原因或者服务自身问题导致调用故障或延迟，随着服务调用问题的累积，服务可能会发生崩溃。断路器Hystrix等服务保护机制的出现解决了这些问题。当较低层服务出现故障时，此时若对其调用失败的频度达到一个阈值，断路器就会打开。为避免故障连锁反应，可以使用fallback()方法对其直接返回一个固定值。

2.2　WebSocket

本平台中保持设备界面与浏览器界面同步过程中使用了WebSocket协议，与使用HTTP协议相比，其优势在于能够由服务器主动通过TCP连接快速推送图片信息给前端处理展示，保持浏览器界面流畅性。

WebSocket协议是一种新型网络通信协议，它作为HTML5的核心技术，被从桌面到平板电脑和智能手机的所有主流浏览器采用 [18]。它允许服务端主动向客户端推送数据，使得客户端与服务端之间数据交换变得更加简单。在WebSocket API中，浏览器跟服务器在第一次握手完成之后，它们之间就会创建一个持久性连接，然后通过连接进行双向数据传输。

在计算机网络协议中，HTTP协议是一种无连接、无状态、单向的网络传输协议，该协议采用请求-响应模型 [19]。这就有个弊端，通信只能由客户端发起，它做不到服务器主动传输消息给客户端。这种单向请求特点，使得服务

器若发生连续状态变化时，客户端获知起来非常麻烦。现在，很多网站为了实现推送技术，使用Ajax轮询，轮询是由浏览器根据定义的时间定期向服务器发出HTTP请求，然后由服务器响应并返回当前最新数据给浏览器 [20]。这种模式有一个很明显的缺点：客户端浏览器需要不断发送HTTP请求给服务器，而服务器从请求中获取的真正有效数据可能只有其中一小部分，显然这样就浪费很多带宽等资源。另外还有一种实现轮询效果的技术是Comet，这种技术需要反复发出请求来实现双向通信，会消耗服务器资源 [21]。相反，WebSocket协议定义了客户端和服务器间长时间存活的专用TCP连接，它的出现能够节省很多不必要流量，减少网络资源占用，减少信息延缓的同时节约了服务器资源。

在建立WebSocket连接之前，浏览器首先会发送一个HTTP请求给服务器，与通常的HTTP请求不同的是，该请求会包含一些附件头信息，比如“Upgrade:WebSocket”表明要升级到WebSocket协议 [22]。服务器获取请求后解析里面的附加头信息，然后返回一个应答消息给客户端浏览器，这样客户端与服务器两者之间就建立了WebSocket连接，它们可通过这个连接互相传递信息，并且该连接是一个长连接，它会持续到客户端与服务器中的某一方关闭该连接。WebSocket提供WebSocket API，定义Web应用和服务器进行双向通信的公共接口 [23]。开发人员通过WebSocket API来使用WebSocket协议，可以实现全双工通信的基本功能，如建立和断开连接、收发数据、监听服务端触发事件等，以此实现系统和应用开发。

2.3 Netty

Netty是一个高性能事件驱动的和异步的Java NIO框架 [24]。作为当前最流行网络应用程序框架之一，Netty受到业界广泛关注及应用，在很多互联网公司、游戏企业、通信企业中有很多项目系统是基于Netty来搭建。

Netty内部I/O使用React设计模式，主要由两类线程组成：boss线程和worker线程 [25]。当启动一个Netty的server实例后，唯一一个boss线程就会启动，主要负责监听端口，处理远程网络连接的建立、维护管理、断开等一系列相关工作。当收到新的连接请求时，boss线程会产生一个task交给某一个worker线程进行处理。每一个worker线程都会维护一个或多个队列，并循环对它们做select操作和相关处理。

Netty的核心组件主要有：Channel、回调、Future、ChannelHandler、EventLoop、Bootstrap和ServerBootstrap。

Channel是Netty网络通信的主体，主要是同用户端进行绑定，之后同用户端进行网络连接以及通信。Channel是双向的，既可以读又可以写，它可以一次

读取或写入一个Buffer的数据 [26]。在每个Channel的整个生命周期内，所有操作都将由相同的Thread执行。

回调其实是一个提供给另外一个方法的方法引用。Netty在内部使用回调机制来处理事件，回调方法里可自定义数据处理器，也可以使用Netty内置的ChannelHandler。当回调被触发时，相关事件就可以通过实现ChannelHandler接口来处理。

Netty中所有I/O操作都是异步的，异步表示该操作在执行后不会立即返回结果，这时我们若想要在之后得到结果就需要一种方法来判断该操作已完成并能访问结果。Future和回调相互补充，在操作执行后并在未来某个时刻完成时，它负责通知应用程序并提供对操作结果的访问 [27]。ChannelFuture是由Netty提供来实现Future机制，它在异步操作执行时被使用。

ChannelHandler负责对数据的输入与输出进行处理。Netty提供强大的事件处理机制，当客户端与服务器进行连接时，Netty会新建一个ChannelPipeline来处理事件，每个ChannelPipeline可包含若干个ChannelHandler，它允许用户自定义ChannelHandler的实现来处理数据。

EventLoop定义了Netty的核心抽象，在开始到结束网络连接的这段生命周期里发生的事件就由EventLoop来处理。在Netty内部，由EventLoopGroup为每个Channel分配一个EventLoop。EventLoop处理一个Channel的所有I/O事件，并且它在从启动到连接断开的整个过程中都不会发生变化。这种设计使得开发人员不再考虑需要在一些ChannelHandler实现中进行同步的问题。

Bootstrap和ServerBootstrap负责引导Netty客户端与服务端的启动 [28]。Bootstrap是客户端的引导类，Bootstrap类将会在bind()方法被调用后创建一个新的Channel，在这之后将会调用connect()方法以建立连接，实现网络数据传输。ServerBootstrap是服务端的引导类，在bind()方法被调用时会创建一个ServerChannel来接受来自客户端的连接，当连接被接受时，ServerChannel将会创建一个新的子Channel用于同客户端进行通信，并对这些子Channel进行管理。

Netty作为业界最流行的NIO框架之一，提供简单实用的API接口，功能强大，扩展性强，性能高，支持多种网络协议(HTTP、TCP、UDP、FTP以及WebSocket)，适用于软件的快速开发。通过Netty框架，本平台不需要受到原始socket的每次传输数据大小限制，无需过多的处理数据就能保证设备操控中图片信息和操作命令的传输正常，并且Netty的稳定性也保证设备操控期间不用太过担心连接的异常中断问题。

2.4　设备交互工具

2.4.1　MiniCap

本平台通过MiniCap的高效率截图功能保证了设备界面同步展示不卡顿。MiniCap是开源项目STF（Smartphone Test Farm）中的一个工具，负责屏幕显示 [29]。原生Android截图工具ScreenCap截图十分缓慢，因此STF开发了一个工具MiniCap用来替代它。

MiniCap通过实时抓取手机显示缓存的数据生成二进制的图片文件信息，它在低版本Android系统上采用截屏方式获取画面，在4.2版本以上的Android系统上采用创建VirtualDisplay的方式来获取画面，性能大大提高。MiniCap截图效率极高，并且它会动态改变图片传输数量，当设备页面基本不动时它每秒传输的图片很少。MiniCap工具属于Android底层开发，该工具分为动态连接库.so文件和可执行文件两部分。但它们不是通用的，其中MiniCap可执行文件根据CPU的4种ABI架构：arm64-v8a、armeabi-v7a、x86和x86_64分为不同版本文件，而minicap.so文件在这个基础上还要根据安卓设备的SDK版本不同而分为不同文件。

将MiniCap推送到手机上后启动，MiniCap就会通过socket不断传输信息。这些信息分为三部分：

第一部分：Banner模块。Banner模块信息在连接后只发送一次，是一些汇总信息，一般为24个十六进制字符，每一个字符都表示不同的信息。Banner模块信息中第0位字符表示MiniCap版本信息，第1位表示Banner信息的长度，方便循环使用。信息中第2、3、4、5位字符相加为运行MiniCap的进程id号，第6、7、8、9 位累加得到设备真实宽度，第10、11、12、13位累加得到设备真实高度，第14、15、16、17位累加得到设备的虚拟宽度以及第18、19、20、21位累加可得到设备虚拟高度。最后，信息中第22位字符表示设备方向，第23位则表示设备信息获取策略。

第二部分：图片大小与图片二进制信息模块。在第一次发送Banner部分信息后，MiniCap后续只会发送与图片内容相关的信息。第一个过来的图片信息前4个字符不是图片的二进制信息，而是携带着图片大小信息，我们需要进行累加得到图片大小。这一部分信息除去前4个字符后，剩余的则是图片实际二进制信息，比如我们接受到的信息长度为n，那么4～n部分是图片信息，需要保存下来进行处理及传输。

第三部分：图片二进制信息模块。MiniCap发送的每一个变化的界面信息都会携带图片大小信息和图片二进制信息，由于每张图片的二进制信息可能一

次性无法传输完，当得到图片大小后，后面发送过来的数据可能都是要组装成图片二进制信息，直到当前屏幕数据发送完成。当又遇到第二部分或设定大小的数据已经装满就表明该图片信息已组装完成。

MiniCap只需要通过简单的启动命令就可以持续地获取设备界面图片信息，之后根据用户需求传输到指定模块中进行处理，保证界面上实时展示设备界面的可靠性。

2.4.2 MiniTouch

MiniTouch也是项目STF中开发的一个工具，主要负责设备触摸操作功能。与MiniCap一样，MiniTouch也是用NDK开发的，使用方法与MiniCap类似，不过它只包含可执行文件，文件版本取决于CPU的ABI架构。MiniTouch能够支持api 10以上的设备且不需要通过root [30]，它通过向socket发送信息进行操作，响应Android设备上的多点触摸事件以及手势。MiniTouch将每一个触摸操作为一条socket命令，该命令格式有四种：第一种为“d <contact> <x> <y> <pressure>”，其中d为按下操作，contact为触摸点索引，它从0开始并可以有多个触摸点，x与y为具体点击坐标，pressure为压力值；第二种为“m <contact> <x> <y> <pressure>”，其中m为滑动操作，contact为触摸点索引，它从0 开始并可以有多个触摸点，x与y为要移动到的位置坐标，pressure为压力值；第三种为“u <contact>”，其中u为松开操作，contact为触摸点索引，它从0开始，可以有多个触摸点；第四种为“c”，其中c为commit操作，用于提交当前触摸设置并在屏幕上执行。

其中每个命令结尾端都要加上“\n”，并对于相同触摸点的每条d、m、u操作后面都需要跟一个c操作，这样命令才能正确在屏幕上执行。通过MiniTouch，就可以用简单的命令来实现设备的完整控制。

2.5 Appium

Appium是一款开源自动化测试框架，它是针对原生、Web和混合应用的一款高效测试工具 [31]。它的server是一个用node.js编写的HTTP服务器，为iOS和Android等不同平台创建和处理多个WebDriver会话 [32]。

Appium遵循着四种设计理念来满足应用自动化需求：无需为了自动化测试一个app而重新编译或者修改它；不必刻意学习特定的测试语言或者框架来编写和运行测试脚本；测试的API不变，一个移动自动化框架不需要重复造轮子；无论是精神上，还是名义上，都必须开源。

为了满足第一点，Appium使用iOS和Android自身提供的第三方自动化测试框架 [33]。Appium把这些第三方框架封装成一套API，API.WebDriver指定了客户端到服务端的协议 [34]，这满足第二条理念。使用这种C/S架构，编程人员就可以编写任何语言版本的客户端，并发送合适的HTTP请求给服务端。目前Appium已经实现大多数语言版本的客户端，这意味着编程人员基本可以使用任何测试套件或者测试框架 [35]。事实上，WebDriver已经成为Web浏览器自动化标准，也是W3C工作草案 [36]，因此Appium扩充了WebDriver的协议，在原有基础上添加与移动自动化相关的API方法，这样就满足第三点设计理念。

Appium Server在运行时会不断监听Client端发送的操作请求，通过翻译器将不同编程语言版本的测试脚本请求发送给模拟器或移动终端，通过这种机制实现手机模拟器或真机的自动化操作 [37]。

目前，除了Appium自动化测试框架之外，安卓自动化测试相关的测试框架还有MonkeyRunner，Espresso，Selendroid等。目前业界也有很多科研工作者正在研究，他们对比了Appium和这些测试工具，证明Appium在手机界面测试中具有一定优势，通过Appium可实现多语言版本脚本的执行回放。同时由于本平台所属的移动应用测试平台也使用Appium进行测试，为了保证技术统一性，选择Appium进行测试脚本的回放，这也使得部分逻辑代码可以直接复用，加快项目开发进度。

2.6　Angular2

AngularJS是一款著名的开源Javascript前端页面开发框架，由著名互联网公司谷歌的团队人员进行创建和维护 [38]。AngularJS为快速开发Web前端提供一个很好的解决方案。AngularJS的出现弥补传统HTML，CSS和JavaScript技术栈在构建应用上的不足，使用AngularJS可以使应用更具表现力 [39]。

Angular2是AngularJS的升级版本，性能上得到显著提高，能很好地支持Web开发组件 [40]。框架整体上基于TypeScript开发，与Java相似，在平台中使用时易于理解和开发，帮助缩短开发周期。相比于AngularJS的MVC架构，Angular2使用一个典型的基于组件架构，把应用设计分解为可重用的功能，能够有效增加代码可重用性。Angular2非常有效地分离了渲染与交互逻辑，这使得它可以很好地跟一些渲染引擎搭配使用 [41]。Angular2还分离了数据供应和变更检测逻辑，从而让它可以自由使用包括RxJS在内的第三方数据库，同时它还把后端开发中用来解决复杂问题的依赖注入DI（Dependency Injection）技术引入前端，降低服务端开发难度。最后，Angular2提供项目的全生命周期支持，提高开发效率，保证项目可维护性。

Angular2框架的使用将平台前后端分离开来，在平台开发中可使开发人员专门保持精力放在前端或后端上，并且让前端可以做很大部分的数据处理工作，减小服务器压力。同时市场上有许多基于Angular2框架的开源UI组件库，它们通过简单配置就能在前端使用，帮助绘制出更丰富的交互界面。

2.7　本章小结

本章主要介绍了在项目中用到的关键技术框架和工具，并通过阐述它们的优点表明这些技术工具的选用理由。Spring Cloud作为整个项目的微服务框架，用于微服务管理和维护；WebSocket协议和Netty框架则用于Web端与设备服务之间的双向通讯；MiniCap和MinTouch作为设备交互工具进行设备服务与手机之间的数据传输；Appium是移动应用自动化测试框架，用于运行测试脚本实现安卓手机自动化测试；Angular2框架则用于前端开发。

第三章　安卓众包在线验证平台的需求分析与概要设计

3.1　项目整体概述

当今社会人们对安卓应用的需求在不断扩大，安卓应用质量也变得尤为重要。因此安卓应用在上线之前都需要进行测试，来保证应用功能正常以及完整。目前流行的自动化测试可以对安卓应用进行充分测试，之后将缺陷信息等数据以报告形式交由用户参考处理。

现今已有许多企业向用户提供测试平台，在这些平台上用户可对安卓应用进行自动化测试。但是，这些平台都会由于一些问题使得最终生成的自动化测试报告并不完全准确，比如在报告中出现的某些缺陷并不一定是由安卓应用造成的，可能是因为用户提交的测试脚本中某条命令有错误或测试工具某一流程没有正常运行。这时用户需要对这些缺陷进行验证，但在验证中出现设备资源和人力不足问题，使得缺陷验证时间持续太长，影响之后应用缺陷修复。

本平台的目标是能够让用户对这些缺陷进行快速验证，完善缺陷信息。平台通过众包测试项目解决用户人力问题，通过平台本身提供的在线安卓设备解决用户设备资源不足问题。众包任务请求者在将自动化测试报告中的缺陷提交给众包测试项目后，众包工人就可直接通过本平台进行缺陷验证。本平台向众包工人提供线上真机环境，无需他们使用自己手机，下载测试应用以及自己配置手机环境，提高缺陷验证效率。众包工人在平台上可切换不同安卓手机并对手机进行实时操控，进行各种触屏模拟操作来复现缺陷。并且平台提供脚本录制功能，在录制期间用户每次操作都会转为脚本步骤显示，众包工人可对这些步骤进行修改和调整，最终可以生成脚本文件以及对脚本文件进行调试回放，然后可提交这测试脚本供任务请求者参考。录制中众包工人还可以根据需要查看手机logcat日志，使缺陷验证更准确。之后他们在一系列操作中得出结论，在平台上对缺陷进行验证，完善缺陷信息，最终由任务请求者查看所有众包工人的验证情况，根据统计结果获取真正的缺陷。

3.2　总体需求分析

3.2.1　功能性需求

整个平台总共包含设备微服务模块和众包在线验证模块两大部分，其中设备微服务模块可以有多个，该模块直接与手机设备进行交互，负责对设备连接

进行监听，与设备进行双向数据传输并在设备上回放测试脚本。该模块主要的功能性需求如表 3.1所示。

表 3.1: 设备微服务模块功能性需求列表

需求编号	需求名称	需求描述	优先级
R1	监听设备连接	系统对手机设备与服务器的USB连接进行监听，设备连接时存储设备信息，连接断开时移除设备信息	高
R2	上传并启动MiniCap和MiniTouch工具	用户选择设备开始操控时，系统将MiniCap和MiniTouch工具上传到对应设备中并启动	高
R3	处理设备实时屏幕图片信息	系统不断接收MiniCap传来的实时屏幕图片信息并处理，最后通过TCP连接传给众包在线验证模块	高
R4	处理设备操作信息	系统通过TCP连接不断接收来自众包在线验证模块传来的设备操作信息，根据信息的操作类型执行ADB命令或者传给MiniTouch处理执行	高
R5	下载并安装待测APK应用	用户选择手机开始进行缺陷验证后，系统自动下载待测APK应用到本地后将其安装到手机上	中
R6	启动APK应用	在APK应用安装完成后，系统会启动APK，用户也可以选择重启APK应用	中
R7	获取logcat日志	用户在录制期间选择获取logcat日志时系统会执行ADB命令，获取每条logcat信息并通过TCP连接传给众包在线验证模块	中
R8	停止获取logcat日志	用户在录制期间停止获取logcat日志时系统会中断命令，停止logcat信息的获取	低
R9	定期发送当前连接的所有设备信息	系统会根据设置的时间将当前连接到服务器上的所有设备信息定期发送到众包在线验证模块进行管理	高
R10	获取并解析当前设备界面UI结构文件	用户在录制期间每次进行设备操作后系统会获取当前设备界面UI结构文件并解析成node列表传给众包在线验证模块	中
R11	运行脚本	用户选择回放脚本后，系统会收到测试脚本下载地址并下载到本地，然后启动Appium运行该脚本	中

众包在线验证模块则是包括主平台Web端和服务端，并通过Netty以及Spring Cloud与设备微服务进行交互，主要是进行所有前端界面展示，对所有能使用设备进行管理，保证浏览器上设备操控实时性，对测试脚本进行操作以及最后对缺陷进行验证。该模块的功能性需求如表 3.2所示。

表 3.2: 众包在线验证模块功能性需求列表

需求编号	需求名称	需求描述	优先级
R12	接收并存储所有设备信息列表	系统不断接收每一个来自设备微服务的设备信息列表并覆盖上一次信息存储起来，在浏览器上进行展示	高
R13	展示所有连接设备	系统会将所有连接设备在浏览器上展示出来，用户可在界面上查看	高
R14	更新设备状态	用户在开始使用以及使用完设备后都对设备状态进行更新，保证每台设备当前只能被一个用户使用	高
R15	定期清除无效设备信息	系统会定期与所有设备微服务进行心跳测试，对无响应设备微服务中的所有设备信息进行清除	高
R16	展示被操控设备的实时屏幕	系统通过TCP连接不断接收来自设备微服务的实时屏幕图片信息，在Web端将信息解析成图片并实时展示	高
R17	操控设备	用户可对设备进行各种触屏模拟操作，此时Web端会将操作转为ADB命令或MiniTouch支持的命令并通过TCP连接发送给对应设备微服务执行	高
R18	展示logcat日志	系统通过TCP连接获取到logcat日志并逐条在界面上展示	中
R19	录制脚本步骤	用户可录制测试脚本，选择开始录制后，系统会把用户接下来的每一次操作转为脚本步骤展示在浏览器界面上	中
R20	操作脚本步骤	用户可对显示的所有脚本步骤进行删除、修改、移动操作	中
R21	生成脚本	当浏览器上至少有一条脚本步骤显示时，用户可选择生成脚本，此时系统会将每条脚本步骤转化为Appium可执行的Java脚本命令并按顺序填入到脚本模板中，然后生成Java文件到本地并在浏览器上显示文件内容	中
R22	回放脚本	当有测试脚本生成后，用户可选择脚本回放，此时系统会将脚本下载地址发给对应设备微服务进行下载及脚本回放	中
R23	获取缺陷信息	用户可在界面上获取并查看要验证的缺陷信息如截图、缺陷内容、出错设备等	高
R24	验证缺陷	用户可对当前缺陷进行验证，判断当前缺陷是否是缺陷或者不确定，若是缺陷就完善缺陷信息如BUG分类、严重等级和复现程度，并可选择是否将生成的脚本作为参考	高
R25	统计并展示验证结果	系统对每个缺陷的验证结果进行统计，展示统计结果图以及验证列表给用户	高

3.2.2　非功能性需求

对于安卓众包在线验证平台来讲，可扩展性、性能、安全性以及可管理性是该平台重要的非功能性需求，这些需求的具体描述如下。

可扩展性：平台能够在不影响当前功能的前提下扩展新功能，能够增加iOS手机操控，以及能够增加Appium可运行的脚本类型等。

性能：平台在运行过程中能够支持至少100人同时在线操控设备，并且能够保持他们操控过程流畅不卡顿。

安全性：平台中需要用户登录，且只有参与缺陷众包验证项目的众包工人才能在线操控设备进行缺陷验证，只有众包任务请求者才能看到所有众包工人的提交结果。

可管理性：保证平台的部署、监控可追踪和可视化，随时可以了解平台的运行情况。

3.2.3　系统用例

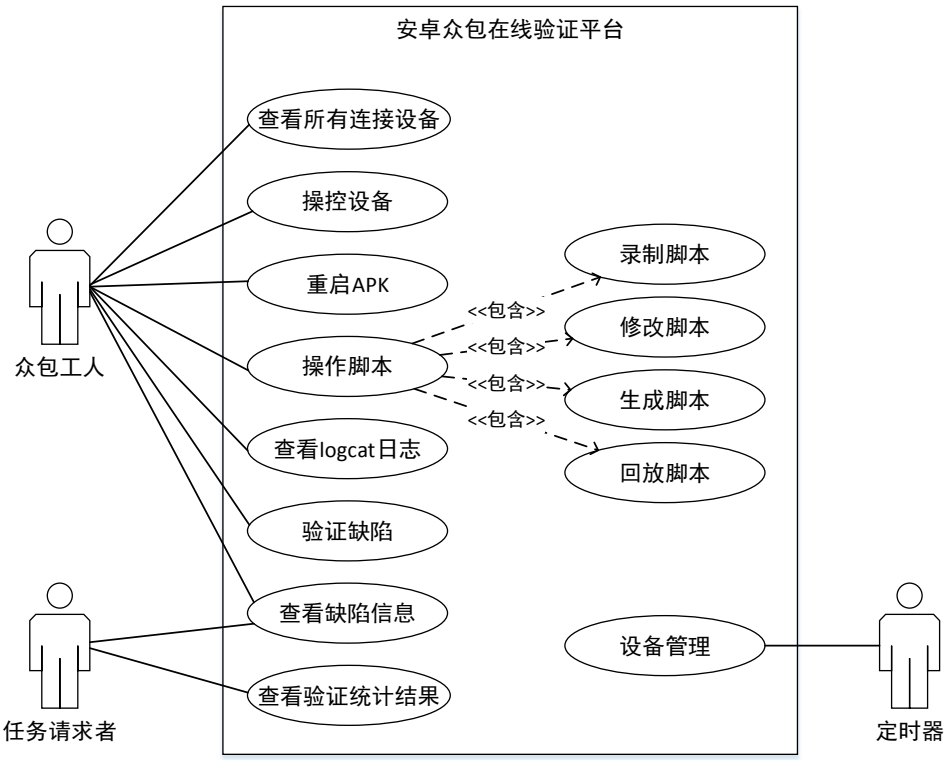

图 3.1: 系统用例图

根据相关功能性需求，可以分析得到如图 3.1所示的系统用例图。本平台主要角色为众包工人、任务请求者和定时器。众包工人在平台上主要可以查看所

有连接设备，可使用空闲设备进行操控，在操控中重启设备上APK应用。众包工人还可以对测试脚本进行录制、修改、生成、回放操作，并在脚本录制过程中查看手机logcat日志，最终根据查看的缺陷信息和界面上设备和脚本操作结果对缺陷进行验证。任务请求者则是可以查看缺陷详细信息以及查看所有参与该众包测试项目的众包工人的验证结果统计图和结果列表。系统定时器则是在系统启动后运行，接收来自各个设备微服务监听到的设备信息，对所有设备进行存储和管理，定期更新设备信息列表。

表 3.3: 系统用例表

用例编号	用例名称	功能需求编号
UC1	查看所有连接设备	R13
UC2	操控设备	R2、R3、R4、R5、R14、R16、R17
UC3	查看logcat日志	R7、R8、R18
UC4	操作脚本	R10、R11、R19、R20、R21、R22
UC5	查看缺陷信息	R23
UC6	验证缺陷	R24
UC7	查看验证统计结果	R25
UC8	重启APK	R6
UC9	设备管理	R1、R9、R12、R15

如表 3.3所示，平台共有9个主要系统用例，分别是查看所有连接设备、操控设备、查看logcat日志、操作脚本、查看缺陷信息、验证缺陷、查看验证统计结果、重启APK和设备管理。各用例附上了用例编号，并与涉及到的功能性需求相对应。

以下是对每个用例的详细描述，描述用例的ID、名称、参与者、优先级、触发条件、前置条件、后置条件、正常流程和异常流程。

表 3.4: 查看所有连接设备用例描述表

ID	UC1
名称	查看所有连接设备
参与者	主要参与者：众包工人 目的：查看所有可以操控使用的设备
优先级	高
触发条件	众包工人点击缺陷验证按钮
前置条件	众包工人参与了该众包测试项目
后置条件	众包工人可以选择可用手机进行操控，进入到缺陷验证界面
正常流程	1. 众包工人点击缺陷操作栏中的缺陷验证按钮 2. 界面展示所有正在连接的设备信息
异常流程	无

用例UC1：查看所有连接设备的主要参与者是众包工人，他们进行验证缺陷之前需要知道当前连接设备信息，好从中选择某台设备进行操控。该用例是进行设备操控的基础，所以优先级为高。参与项目的众包工人在点击缺陷操作栏中的缺陷验证按钮后才能看到界面展示的所有连接设备，然后选择设备使用后可进入到缺陷验证界面进行设备操控。该用例具体描述如表 3.4所示。

用例UC2：操控设备是系统核心用例，优先级为高，用例的主要参与者是众包工人。众包工人在设备列表中可选择未被占用的手机并点击使用，进入到缺陷验证界面，这时若手机与服务器之间处于连接状态，系统会先判断设备上是否有MiniCap和MiniTouch工具，有则直接启动，无则上传工具到设备上后再启动。只有在两工具正常启动后，浏览器界面才会保持与设备的双向通信，保证Web端与设备界面同步。期间系统会自动下载并安装待测APK应用，并在设备上启动，之后众包工人就可对该APK应用进行远程操作。若与设备的连接中断，界面会弹出提示框告知众包工人连接已中断并提供退出按钮。该用例描述如表 3.5所示。

表 3.5: 操控设备用例描述表

ID	UC2
名称	操控设备
参与者	主要参与者：众包工人 目的：对选择的设备进行在线操控
优先级	高
触发条件	众包工人选择未被占用手机并点击使用，进入到缺陷验证界面
前置条件	手机与服务器之间处于连接状态，MiniCap和MiniTouch在设备上正常启动
后置条件	众包工人可以进行脚本操作和缺陷验证
正常流程	1. 众包工人选择设备列表中未占用手机，点击该手机信息下方的使用按钮 2. 众包工人进入到缺陷验证界面 3. 若设备上没有MiniCap和MiniTouch工具，就将它们上传到设备上并启动；若有则直接启动 4. 系统下载、安装并启动待测APK 5. 系统实时展示设备界面，同时系统响应众包工人每次手机操作
异常流程	5. 手机连接中断，弹出错误提示框告诉众包工人连接发生异常并提供退出按钮

用例UC3：查看logcat日志是在测试脚本录制期间用户可选择的用例，用例的优先级为中。该用例的主要参与者是众包工人，在他们进行测试脚本录制时，众包工人可以看到log展示框，并在点击框内“获取”按钮后，系统会不断获取设备里的实时logcat日志并将日志信息如日志等级、时间、进程号、日志信息等逐条展示在框内。该用例的描述如表 3.6所示。

表 3.6: 查看logcat日志用例描述表

ID	UC3
名称	查看logcat日志
参与者	主要参与者：众包工人 目的：查看设备当前的logcat日志
优先级	中
触发条件	众包工人点击log展示框中的获取按钮
前置条件	众包工人正在进行脚本录制
后置条件	无
正常流程	1. 众包工人点击开始录制按钮 2. 众包工人点击log展示框中的获取按钮 3. 系统不断获取设备中的logcat日志并将日志信息逐条展示在界面log展示框中
异常流程	无

用例UC4：操作脚本的主要参与者是众包工人，目的是将操作录制成测试脚本步骤，并同时可对脚本步骤进行修改、生成和回放。脚本操作是平台的主要功能之一，所以优先级为高。在众包工人点击开始录制按钮后，当设备处于正常操控时，众包工人每次设备操作就会被转为测试脚本步骤，然后由众包工人可选择性的进行脚本生成和回放。该用例的详细描述如表 3.7所示。

表 3.7: 操作脚本用例描述表

ID	UC4
名称	操作脚本
参与者	主要参与者：众包工人 目的：将设备操作录制成脚本步骤，对脚本步骤进行修改、脚本生成及回放操作
优先级	高
触发条件	众包工人点击开始录制按钮
前置条件	设备处于正常操控中
后置条件	无
正常流程	1. 众包工人点击开始录制按钮 2. 众包工人对手机进行操作，系统将其转为脚本步骤展示 3. 众包工人对脚本步骤进行修改、脚本生成、脚本回放
异常流程	无

用例UC5：查看缺陷信息的主要参与者可以是众包工人，也可以是众包任务请求者。众包工人需要在了解待验证缺陷的详细信息下才能对缺陷进行准确验证，所以该用例优先级设为高。众包工人选择设备使用后在缺陷验证界面可以看到缺陷详细信息，任务请求者则在验证结果展示界面可以看到缺陷信息。该用例的描述如表 3.8所示。

表 3.8: 查看缺陷信息用例描述表

ID	UC5
名称	查看缺陷信息
参与者	主要参与者：众包工人和任务请求者 目的：查看当前要验证的缺陷信息
优先级	高
触发条件	众包工人选择手机使用并进入到缺陷验证界面或任务请求者进入验证结果界面
前置条件	众包工人选择的手机可以被使用或任务请求者是众包测试项目的创建者
后置条件	无
正常流程	1a. 众包工人选择可用手机，点击使用 2a. 众包工人进入到缺陷验证界面 3a. 界面显示缺陷信息 1b. 任务请求者进入到缺陷的验证结果展示界面 2b. 界面显示缺陷详细信息
异常流程	2a. 手机连接中断，弹出错误提示框告诉众包工人连接发生异常并提供退出按钮

用例UC6：验证缺陷是该平台的核心用例，优先级为高。验证缺陷是该平台的最终目的，主要参与者是众包工人，由众包工人对众包项目中各个缺陷进行在线验证。众包工人在选择可用设备使用后进入到缺陷验证界面，通过缺陷信息的查看以及对设备和测试脚本的操作对缺陷进行判断，若判断是缺陷，则需要再选择缺陷的BUG分类、严重等级和复现程度。在判断完后，若众包工人在验证过程还生成过脚本并认为该脚本更准确，可选择将其作为参考随验证结果一起提交给任务请求者查看。该用例的用例描述如表 3.9所示。

表 3.9: 验证缺陷用例描述表

ID	UC6
名称	验证缺陷
参与者	主要参与者：众包工人 目的：众包工人对缺陷进行验证
优先级	高
触发条件	众包工人选择手机使用并进入到缺陷验证界面
前置条件	众包工人参与了该众包测试项目，手机可以被使用
后置条件	无
正常流程	1. 众包工人选择可用手机，点击使用 2. 众包工人进入到缺陷验证界面 3. 众包工人进行手机和脚本操作，根据操作结果验证缺陷 4. 若判断是缺陷，选择缺陷的BUG分类、严重等级和复现程度；若判断不是缺陷或不确定则不选择 5. 若生成过脚本，选择是否将脚本作为参考 6. 点击确定按钮
异常流程	2. 手机连接中断，弹出错误提示框告诉众包工人连接发生异常并提供退出按钮

用例UC7：查看验证统计结果的参与者是众包任务请求者，只有任务请求者才能看到所有众包工人提交的缺陷验证结果并最终根据结果统计判断缺陷，优先级为高。只有任务请求者可以在缺陷列表中缺陷的操作栏中看到验证结果按钮，并在点击后进入验证结果界面，看到界面展示的结果统计图和结果列表。该用例的用例描述如表 3.10所示。

表 3.10: 查看验证统计结果用例描述表

ID	UC7
名称	查看验证统计结果
参与者	主要参与者：任务请求者 目的：查看所有众包工人提交的缺陷验证结果
优先级	高
触发条件	任务请求者点击缺陷列表中某一缺陷的验证结果按钮，进入验证结果界面
前置条件	任务请求者是众包测试项目的创建者
后置条件	无
正常流程	1. 任务请求者点击某一缺陷的验证结果按钮 2. 界面展示缺陷详细信息 3. 界面展示验证结果统计图和结果列表
异常流程	无

用例UC8：重启APK是用户在设备操控中可根据自己需要选择使用的一个用例，其主要参与者为众包工人，优先级为中。该用例主要是平台提供一个重启待测APK的快捷功能按钮，并可根据众包工人选择让APK重启中清除设备中的应用数据，相当于重新安装APK，保证测试脚本的录制回放不受影响。该用例的描述如表 3.11所示。

表 3.11: 重启APK用例描述表

ID	UC8
名称	重启APK
参与者	主要参与者：众包工人 目的：重启设备中安装完成的待测APK
优先级	中
触发条件	众包工人点击重启App按钮
前置条件	APK已在手机上安装
后置条件	无
正常流程	1. 众包工人点击重启App按钮 2. 众包工人选择是否清除设备中的应用数据 3. 系统重启手机上的待测APK
异常流程	无

用例UC9：设备管理主要参与者是系统，由系统内部的定时器和监听器负责对设备进行统一管理，优先级为高。系统中每个设备微服务在启动后就会启动设备监听器，对设备的连接进行监听，获取连接设备信息并保存，对下线设备信息进行移除，并在同时启动定时器，将所有设备信息发给众包在线验证模块，众包在线验证模块会接收设备信息并保存服务-设备信息映射。在系统启动时该模块也会启动定时器，对每个设备微服务发送心跳，及时移除未响应服务与设备信息的映射。该用例的用例描述如表 3.12所示。

表 3.12: 设备管理用例描述表

ID	UC9
名称	设备管理
参与者	主要参与者：系统 目的：对所有设备微服务中的设备进行统一管理
优先级	高
触发条件	系统启动定时器
前置条件	定时器正常启动和运行
后置条件	用户可以查看并选择使用已连接的设备
正常流程	1a. 每个设备微服务启动设备监听器，监听设备的连接状态 2a. 每个设备微服务启动定时器，定期将监听到的连接的设备信息列表发送给众包在线验证模块。 3a. 众包在线验证模块中系统对收到的设备信息列表进行覆盖存储到服务-设备信息映射表 1b. 众包在线验证模块启动定时器，定期向每个设备微服务发送心跳，对于无响应设备微服务清除服务-设备信息映射表中属于该微服务的所有设备信息
异常流程	无

3.3 系统总体设计与模块设计

3.3.1 总体结构

如图 3.2所示描述了安卓众包在线验证平台的总体架构设计，对平台中各功能模块之间的架构关系进行说明。平台分为设备微服务模块和众包在线验证模块两部分，由Spring Cloud微服务框架对它们进行统一管理。

平台中的设备微服务模块直接与手机设备进行交互，实现与设备直接交互相关的逻辑功能。硬件层中的手机设备通过USB连接到已部署设备微服务模块的服务器上，并打开USB调试模式。平台通过设备数量以及每个服务器能连接设备数决定需要部署的设备微服务模块数量。

众包在线验证模块则是作为系统的主平台，向用户提供可交互的Web端界面并与所有设备微服务模块进行交互。该模块Web端采用Angular2框架进

行页面开发并能完成相对复杂的前端逻辑。模块Web端使用成熟的UI组件库Bootstrap和PrimeNG，可通过组件库中定义好的样式轻松绘制出更丰富的页面与用户交互。

众包在线验证模块服务端采用Spring Boot框架进行开发，有利于快速集成其它服务组件。服务端通过各子模块的功能实现来支撑Web端的正常流程，保证Web端最终与硬件层中设备的正常交互。服务端还与MySQL数据库进行连接，保证用户提交的缺陷验证结果持久化。设备微服务模块与服务端以及服务端与Web端之间通过Netty传输数据保证设备操控和logcat日志获取正常，其余交互方式则通过调用HTTP接口实现。

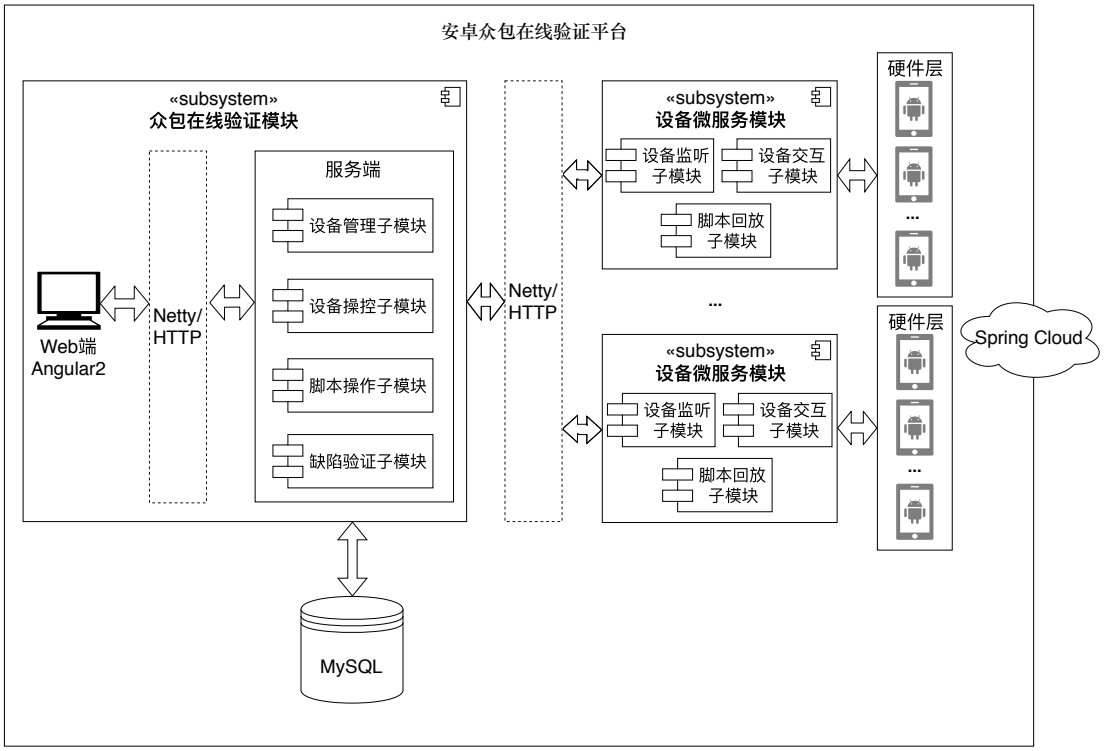

图 3.2: 系统总体架构图

平台系统部署图如图 3.3所示，安卓众包在线验证平台的Web端通过配置的Nginx代理转发用户请求到众包在线验证服务上。Spring Cloud的Eureka注册中心单独部署在一个服务器，对注册服务进行管理。众包在线验证服务连接部署在服务器上的MySQL数据库，并注册到Eureka注册中心上。设备微服务则部署在直接与设备连接的服务器上，每个服务器部署一个设备微服务，并也分别向Eureka注册中心进行服务注册。

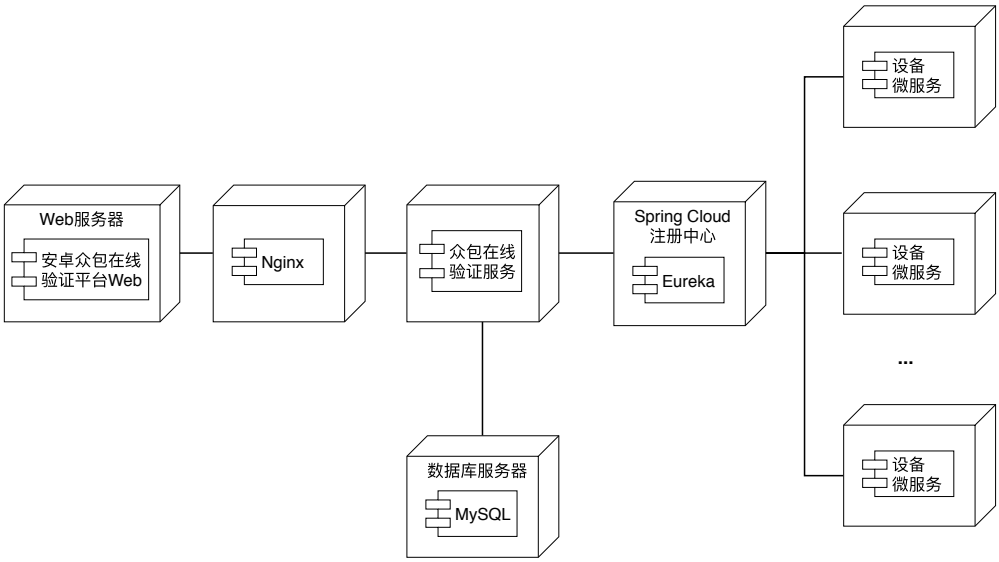

图 3.3: 系统部署图

3.3.2　模块架构

整个平台的模块架构如图 3.4所示，安卓众包在线验证平台分为设备微服务模块和众包在线验证模块。其中设备微服务模块包含设备监听子模块、设备交互子模块以及脚本回放子模块，众包在线验证模块则由设备管理子模块、设备操控子模块、脚本操作子模块和缺陷验证子模块组成。

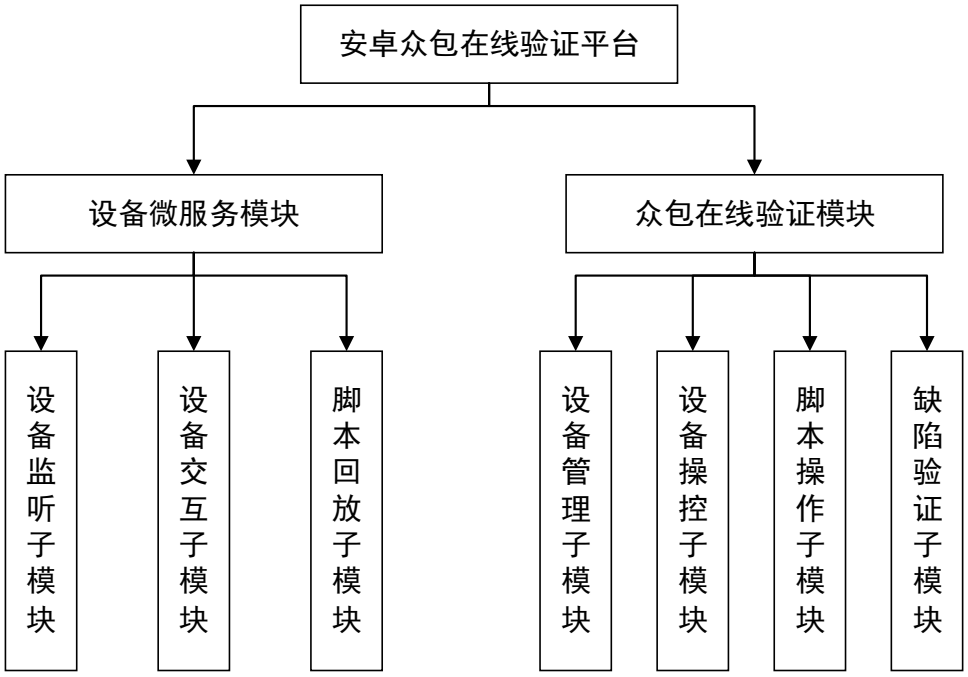

图 3.4: 系统模块图

设备监听子模块：该模块主要负责对通过USB连接到服务器的设备进行监听，根据设备当前连接状态处理设备信息，对连接中的设备信息进行存储，对断开的设备信息在内存中进行移除。同时该模块还会定期将当前设备信息以列表形式发往众包在线验证模块。

设备交互子模块：设备微服务的核心模块，负责与设备进行连接和通信，包括接收、处理以及传输来自设备中MiniCap的图片信息或logcat日志信息，接收来自众包在线验证模块的设备操作命令，对设备执行ADB命令或交由设备中的MiniTouch执行。该模块在设备连接成功后会根据下载地址下载和安装待测APK到设备上并启动。

脚本回放子模块：包含了Appium客户端与服务端的相关逻辑，主要是下载测试脚本到本地，然后启动Appium服务执行脚本，在设备上进行脚本回放。

设备管理子模块：主要负责接收来自各个设备微服务的设备信息列表，覆盖存储每个设备微服务与其包含的设备信息列表的映射关系，更新到服务-设备信息映射表中，并在响应用户请求时返回所有设备信息展示在界面上。该模块还会定期发送心跳到每一个设备微服务中，对于无响应设备微服务清除总映射表中属于它的所有设备信息。

设备操控子模块：保证用户正常操作设备，负责设备界面实时展示以及设备操作实时性。将来自设备微服务的手机图片信息实时展示在界面上，同时将用户设备操作转为约定好的可执行命令实时发往对应设备微服务执行，保证浏览器界面图片与设备屏幕同步。

脚本操作子模块：该模块保证用户在界面上正常进行脚本操作。在用户录制脚本期间，负责在浏览器上用红色矩形框突出鼠标所处于的设备界面控件，将用户每次手机操作转为测试脚本步骤并按顺序排列显示在界面上，根据用户脚本操作对脚本步骤进行修改、删除、移动等操作，并提供logcat日志查看给用户。在用户选择生成脚本时，将脚本步骤按顺序逐个转为Appium能执行的Java测试脚本命令并插入到定义好的脚本模板中，并在模板中填写Appium关于设备和测试应用的配置信息，最终生成Java文件，将内容展示到界面上并提供脚本下载功能。在用户选择脚本回放时将生成的脚本下载地址发送给对应设备微服务进行回放。

缺陷验证子模块：主要负责根据用户请求获取当前要验证的缺陷信息并展示到界面上，在众包工人验证完后将用户提交的验证内容存储到数据库中。对每个缺陷的所有验证结果进行统计，包括计算众包工人对缺陷的判断以及是缺陷时它的BUG分类、复现程度和严重等级比例，以统计图的形式展示给用户，并同时展示原始验证结果列表。

3.4　设备微服务模块设计

3.4.1　设备监听子模块设计

平台进行缺陷验证的基础是需要对设备进行操控，根据操控过程以及结果判断缺陷。设备操控就需要了解设备当前是否与服务器处于连接状态，当前是否可用，因此这就需要时刻对设备进行监听，设备监听子模块就是负责此功能。该模块的主要流程如图 3.5所示。

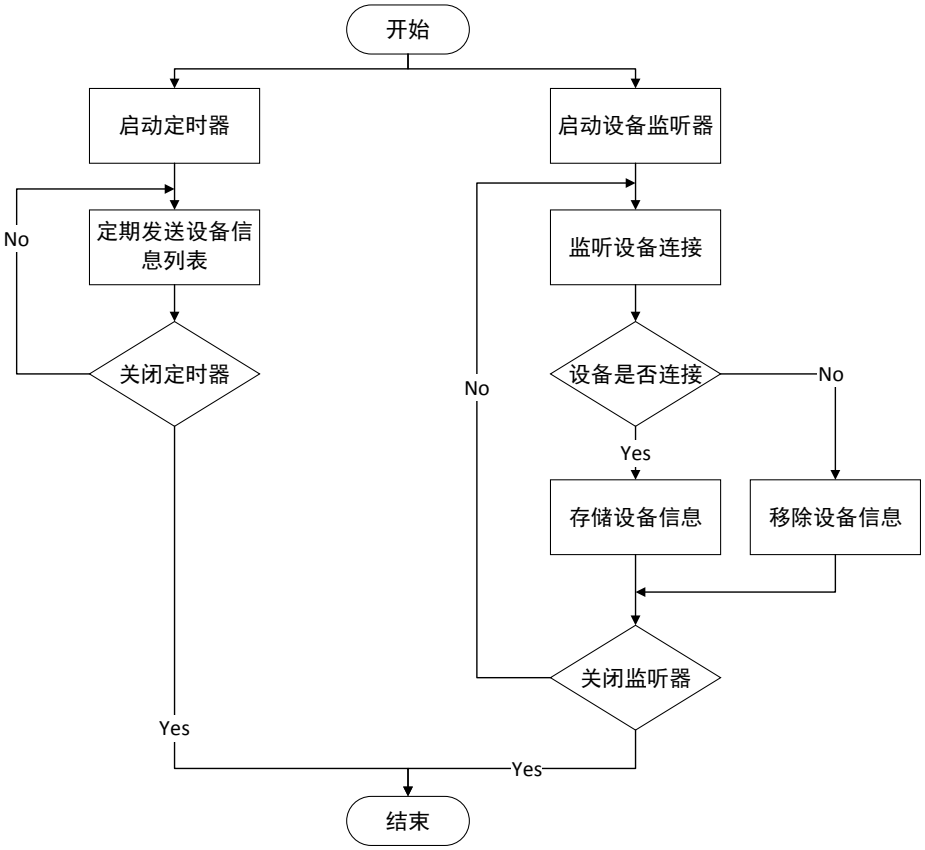

图 3.5: 设备监听流程图

设备监听模块在系统启动后就会开始通过安卓的调试连接桥ADB启动设备监听器，对通过USB接口与服务器连接的手机进行监听，其中部分手机需要开启USB调试才能被监听到。当监听器监听到刚连接上的设备时，获取该设备的主要信息如设备序列号、品牌、型号、分辨率、安卓版本等，之后将这些信息存储在内存中。当监听器监听到设备刚断开连接，就根据设备的唯一标志序列号找到内存中的设备信息并移除。同时，在系统启动后设备监听模块还会启动一个定时器，该定时器在项目启动期间一直执行一个定期任务，任务内容是将当前内存中所有设备信息以列表形式发送给主平台，即众包在线验证模块。

3.4.2　设备交互子模块设计

设备交互子模块是设备微服务的核心部分，平台的主要功能设备操控就是在该模块正常运行基础下实现。它的任务是负责与设备之间进行实时双向通信以及根据需要获取并传输logcat日志。该模块流程图如图 3.6所示。

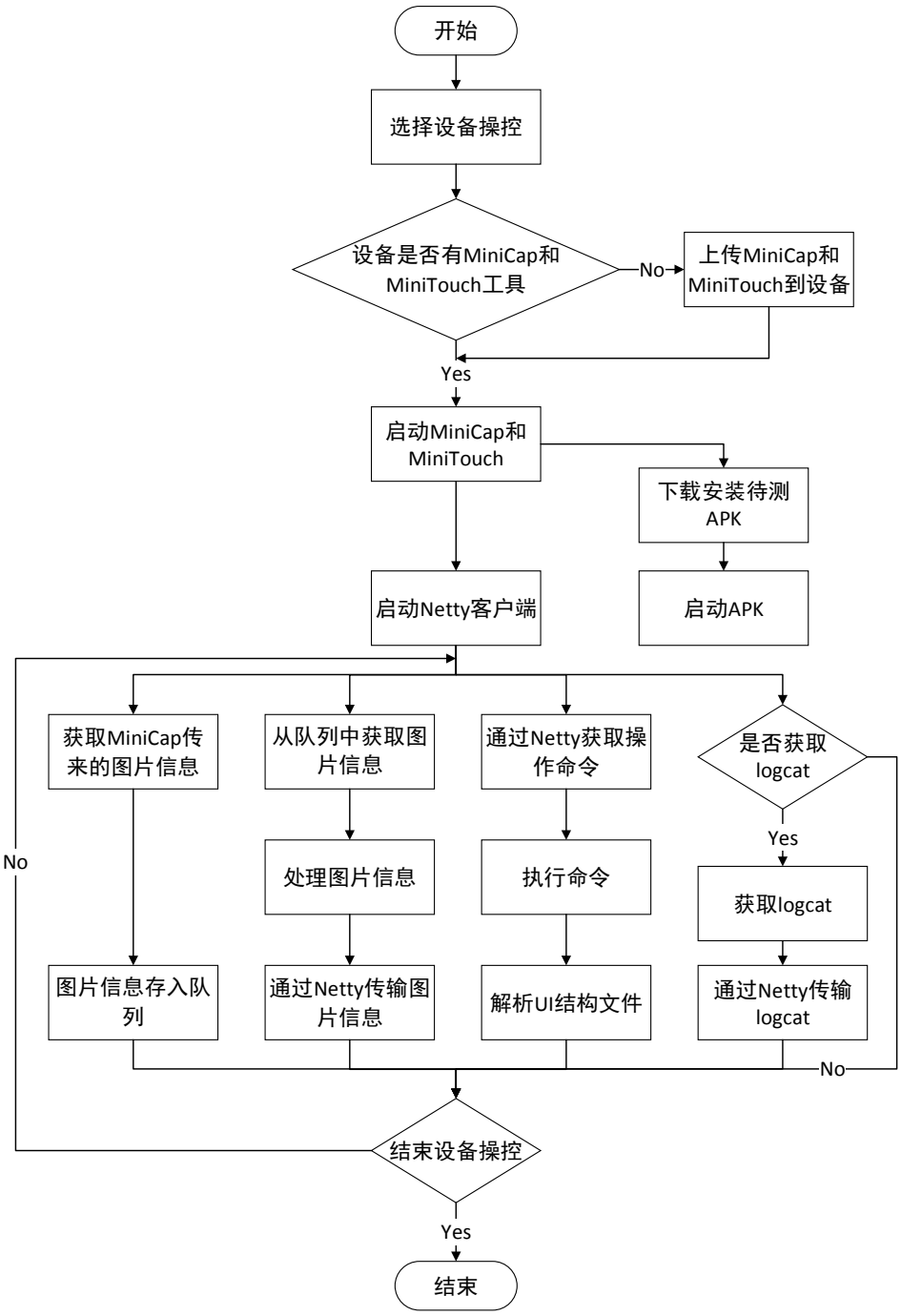

图 3.6: 设备交互流程图

当用户选择使用手机后，设备交互模块就会为每个设备创建一个实例，在该实例中，模块首先会确认设备中是否已有MiniCap和MiniTouch，对于第一次连接使用的手机，将它们的相关文件上传到手机上。之后在手机上启动这两工具，通过端口转发将服务器约定的端口分别与设备中MiniCap和MiniTouch的通讯端口映射起来，这样服务器就可通过这些端口与设备进行交互。

在与设备的通讯准备好以后，模块就会启动Netty客户端与主平台的服务端建立连接，保证数据在主平台与设备微服务之间正常传输，同时模块会根据待测APK的下载地址下载APK到本地存储，并将APK上传到设备中进行安装启动。所有连接工作准备好以后，模块就会开启三个主要线程：第一个线程会不断获取来自MiniCap的实时图片信息字节流，并将信息流存入到队列中；第二个线程则是不断轮询队列，若有数据则逐条拿出进行处理，然后在Netty客户端启动后将处理后的图片信息传输到主平台中；第三个主要线程则是启动Netty客户端，不断获取来自服务端的操作消息，根据消息的命令格式执行相关ADB命令或传输到设备上由MiniTouch执行。当用户在录制脚本步骤时，模块会解析UI结构文件并返回节点信息列表给主平台。用户需要获取logcat日志时，模块会开启新线程启动新的Netty客户端传输从设备获取到的logcat日志信息。

3.4.3　脚本回放子模块设计

脚本回放子模块在设备微服务中主要负责下载和执行Appium脚本。模块流程如图 3.7所示。

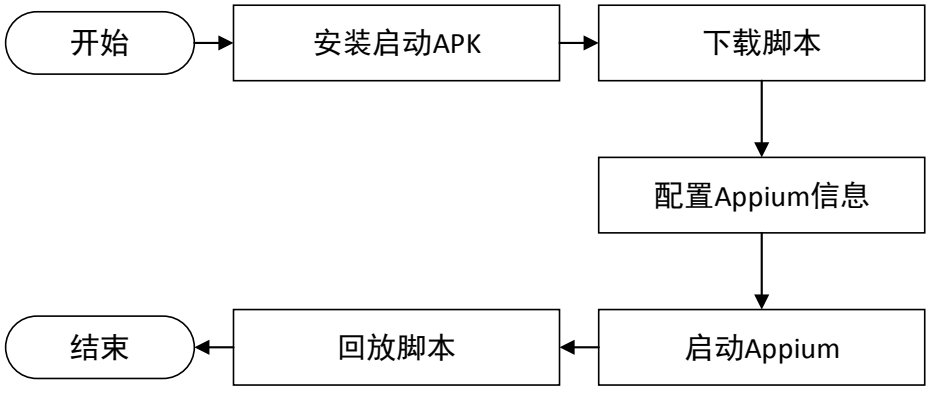

图 3.7: 脚本回放流程图

为了保证执行脚本时应用处于打开状态，脚本交互模块会重新给设备安装待测APK并启动。之后，模块会根据获取的脚本下载地址下载脚本到本地，然后配置Appium执行脚本所需要的一些必要信息，创建Appium的一个实例，启动Appium Server，执行脚本，达到回放脚本的目的。

3.5　众包在线验证模块设计

3.5.1　设备管理子模块设计

作为众包在线验证模块中的设备管理子模块，它负责对所有来自设备微服务的设备信息进行存储管理，并根据用户需求在界面展示这些设备信息。该模块的流程图如图 3.8所示。

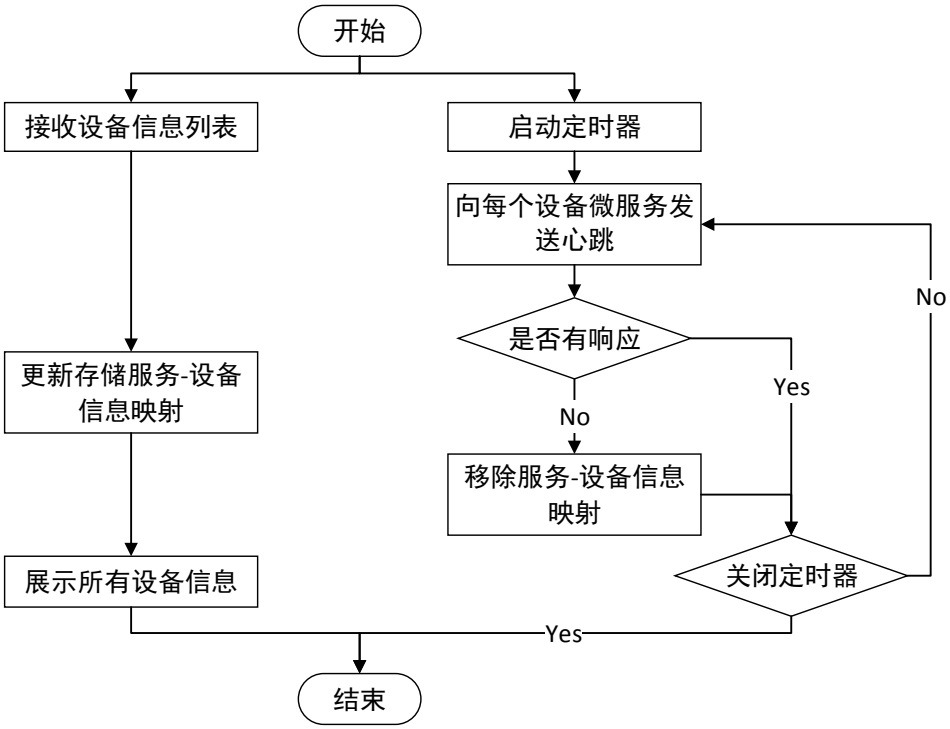

图 3.8: 设备管理流程图

设备管理模块会不断接收来自每个设备微服务发送过来的设备信息列表，并将服务-设备信息映射表中的对应信息更新成当前信息列表，服务-设备信息映射表一直存储在系统内存中。当用户需要在Web界面上查看设备时，模块会将当前映射表中所有设备信息展示给用户。在系统启动后，设备管理模块还会启动一个定时器，定期向每个设备微服务发送一个心跳，对于未响应设备微服务将该服务与它的设备信息映射全部移除，更新映射表。

3.5.2　设备操控子模块设计

设备操控模块是缺陷验证中必不可少的部分，它的主要功能体现在Web端，它保证Web端与设备之间双向通信正常，保持界面图片与设备屏幕同步。模块流程图如图 3.9所示。

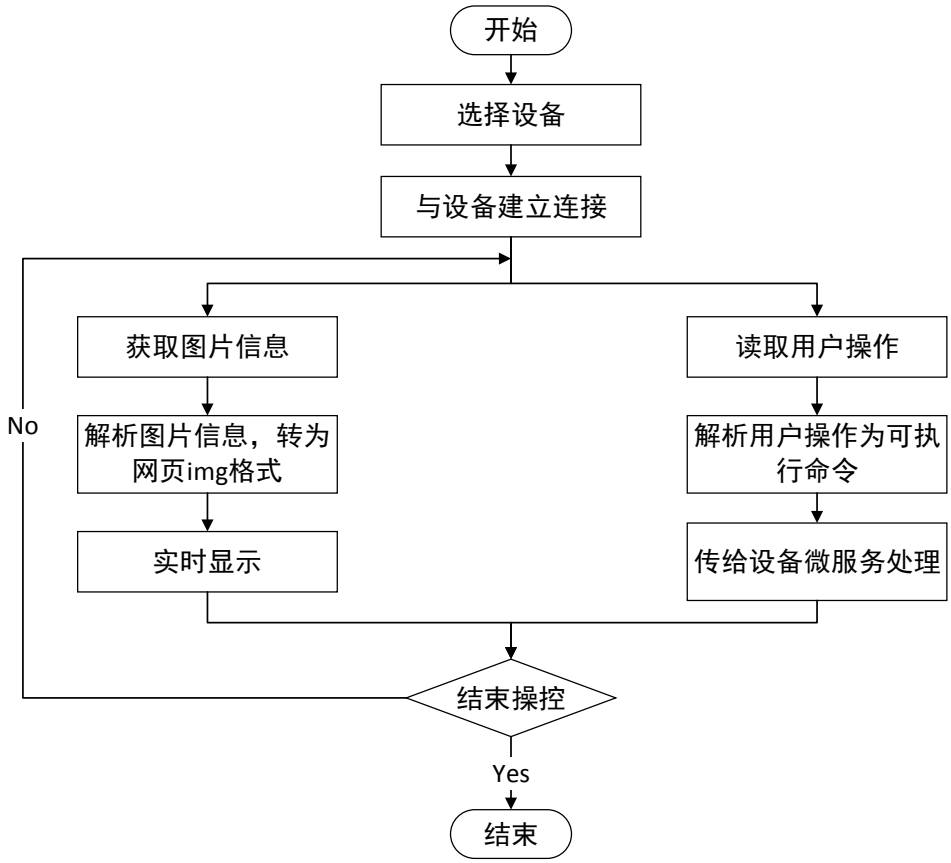

图 3.9: 设备操控流程图

系统在启动后就会在主平台建立Netty服务端，在用户选择完一个设备进行使用后，设备操控模块就会在Web端通过WebSocket协议与Netty服务端进行连接，并通知对应设备微服务建立一个Netty客户端与服务端进行连接。在设备微服务做好所有连接准备后，该模块就可以一直获取来自设备微服务的图片信息，将信息传给Web端，由Web端对图片信息进行读取解析，转为网页支持的img格式显示在canvas界面上。在用户对浏览器上手机界面进行操作时，Web会及时将操作转为可执行命令并传给主平台再由主平台传给设备微服务处理执行。模块就通过重复这些流程保证界面与设备屏幕同步，保证用户操作正常。

3.5.3　脚本操作子模块设计

脚本操作子模块是在设备操控基础上协助缺陷验证的功能模块补充。该模块逻辑主要存在于Web端，它让用户能够在Web端对脚本进行录制、修改、生成及回放操作，模块在录制期间还提供控件属性展示功能及logcat日志查看功能。模块的主要流程如图 3.10所示。

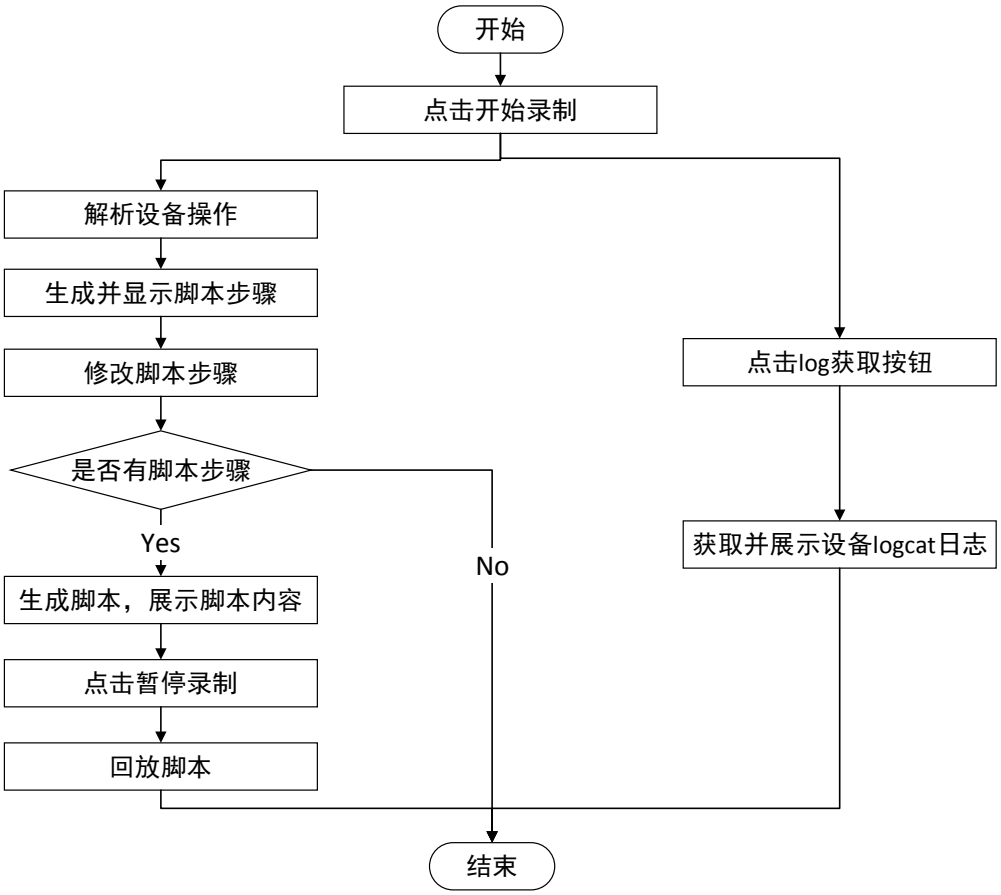

图 3.10: 脚本操作流程图

脚本操作模块给用户提供脚本录制入口，在用户选择进行脚本录制后，模块就会不断监听用户的每次设备操作，在用户每次操作后首先确认操作方式，若是点击操作则获取点击的控件属性，若是长按则在点击基础上增加长按时间，若是滑动操作则获取触摸起始点与释放结束点的坐标。对于用户的键盘事件则视为输入操作，当用户使用键盘输入文字时获取输入内容，按下快捷键则获取Android中对应的KeyCode，相邻操作之间的间隔时间则作为等待时间记录下来作为等待操作，最后将这些数据组合起来形成一条脚本步骤。界面上至少存在一条脚本步骤时，模块就可以将它们生成一个Java文件格式测试脚本，并在录制结束后可对其进行回放。录制期间，模块还提供logcat日志查看功能，可帮助用户更好地验证缺陷。

3.5.4　缺陷验证子模块设计

缺陷验证功能是本平台的最终目的，其它模块都是作为验证手段来辅助用户实现这个目的。该模块的功能就是查看缺陷信息并对缺陷进行验证，将验证

结果存入到数据库中进行统计，最后提供统计结果给任务请求者查看。它的主要流程如图 3.11所示。

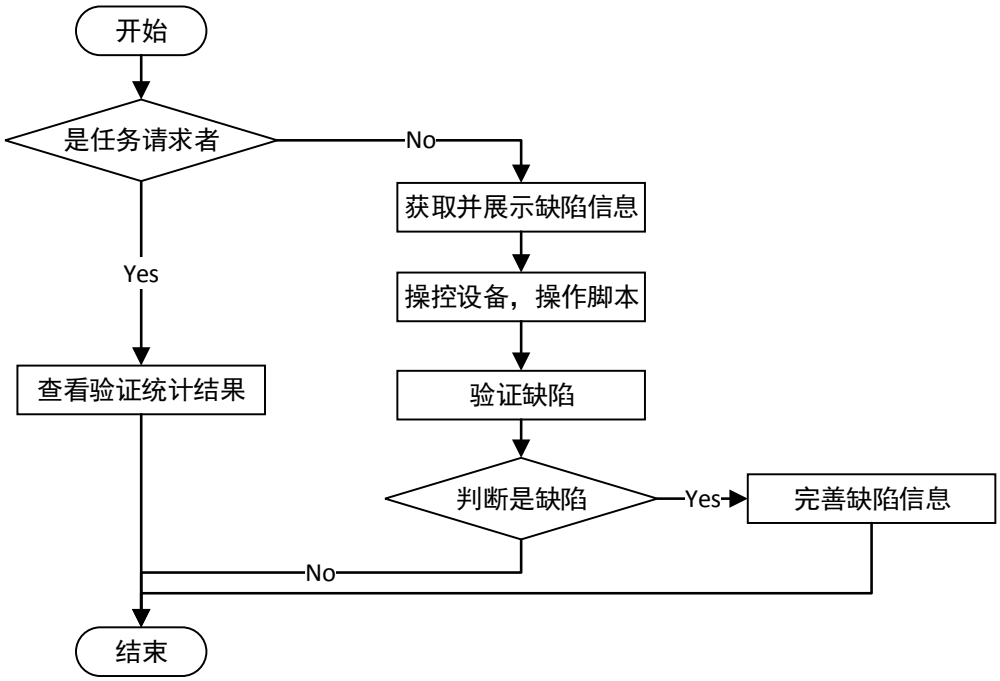

图 3.11: 缺陷验证流程图

在众包工人选择设备使用并进入到缺陷验证界面后，该模块就会获取当前缺陷的基本信息如截图、缺陷内容、当前页面、产生缺陷组件以及出错设备等，然后将信息展示在Web界面上。模块会提供三种缺陷验证结果选择项给用户，分别为是缺陷、不是缺陷和不确定。若用户判断是缺陷时则继续让用户完善缺陷信息，如缺陷复现程度和严重等级等。并且该模块在用户生成脚本后会提供是否将测试脚本作为参考选项，在用户选择该选项后，模块会最终将验证信息连带脚本地址一起存入到数据库中。在任务请求者要查看验证结果时，对所有结果进行统计，展示结果统计图和结果列表。

3.6　数据库设计

本平台使用MySQL作为数据库，由众包在线验证模块与数据库进行连接进行数据操作。本平台中涉及到的关键数据库中的实体有用户user、众包任务task、任务实例case、移动应用app、缺陷验证结果bug_verification_report以及缺陷bug。这些实体的关系如图 3.12所示。

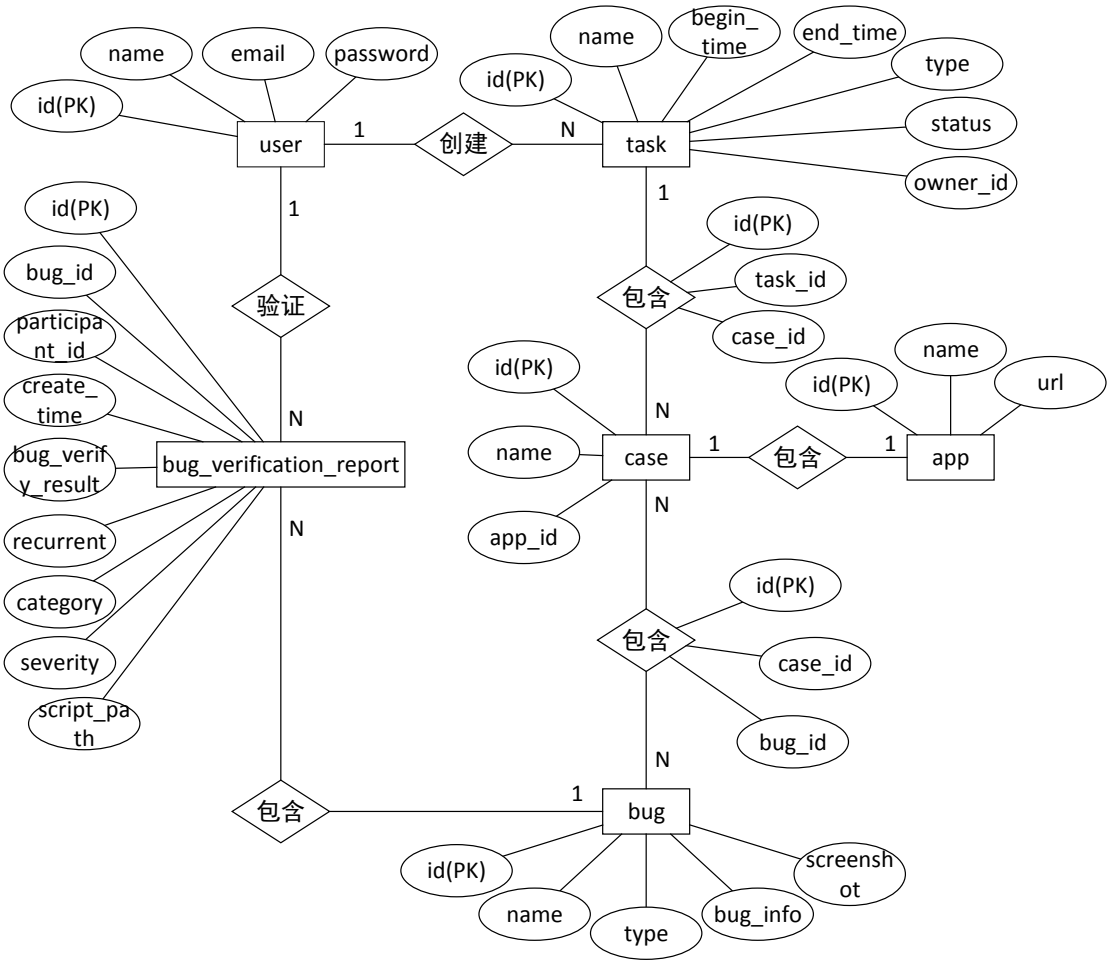

图 3.12: 系统关键数据库实体关系图

在这些实体中，user是保存用户信息，用户可以是众包任务请求者，也可以是众包工人。task表示众包任务，它除了存储它自身的基本信息以外，还保存创建者的id。一个task可以包含多个任务用例case，由单独的表保存它们之间的映射关系，每个任务用例会包含一个app应用，提供用户需要测试的app信息以及下载路径。每个任务用例还包含多个缺陷bug，缺陷也可以由多个任务用例共同拥有给用户进行查看以及验证。每个缺陷可以由多个众包工人对其进行验证，缺陷验证结果存入bug_verification_report表里面。这样保证每个众包工人可对任务中所有缺陷进行验证，让任务请求者从中得到更准确的验证结果。

本平台目的是对缺陷进行验证，这需要保持缺陷信息以及缺陷验证结果的持久性，这就涉及到bug表和bug_verification_report的设计应用。如表 3.13所示为bug数据库表说明，bug表主要存储缺陷的名称、类型、内容以及截图。

表 3.13: bug表

字段	字段名称	类型	可空	描述
id	缺陷id	varchar	否	自增主键
name	缺陷名称	varchar	否	缺陷的名称
type	缺陷类型	varchar	否	缺陷的类型
bug_info	缺陷内容	varchar	否	缺陷的主要内容
screenshot	缺陷截图	varchar	是	缺陷的截图所在路径

bug_verificaiton_report数据库表模型如表 3.14所示，它保存用户对缺陷的验证结果。包括用户验证的缺陷id、验证人在user表中的id、验证结果完成时间、验证结果、参考测试脚本路径以及确认是缺陷时要填的复现程度、BUG分类和严重等级。

表 3.14: bug_verification_report表

字段	字段名称	类型	可空	描述
id	验证数据id	bigint	否	自增主键
bug_id	缺陷id	varchar	否	验证的缺陷id
participant_id	验证人id	bigint	否	参与验证的用户id
create_time	创建时间	datetime	否	验证结果的完成时间
bug_verify_result	验证结果	smallint	否	验证是否是缺陷，0为是，1 为不是，2为不确定
recurrent	复现程度	smallint	是	验证是缺陷时缺陷的复现程度
category	BUG分类	varchar	是	验证是缺陷时缺陷的BUG分类
severity	严重等级	smallint	是	验证是缺陷时缺陷的严重等级
script_path	参考脚本路径	varchar	是	给任务请求者的参考测试脚本路径

3.7　本章小结

本章对安卓众包在线验证平台进行需求分析和概要设计。首先对平台进行整体概述，讲述平台背景和要实现的主要功能。接着对平台进行需求分析，包括功能性需求和非功能性需求，并通过用例图和用例表对用例进行描述。然后对平台进行总体结构设计，并对设备微服务模块和众包在线验证模块中的子模块分别进行分析设计，包括每个模块的主要作用以及主要流程图。最后对平台涉及的关键数据库进行设计，展示实体关系图以及重要的数据库表结构信息。

第四章　安卓众包在线验证平台的详细设计与实现

4.1　设备微服务模块

4.1.1　设备微服务模块介绍

设备微服务模块向Spring Cloud的Eureka注册中心注册服务，它负责接收来自众包在线验证模块的请求并与设备进行通信，对连接的设备进行监听，更新设备信息列表，并根据收到的请求与设备进行连接，通过连接通道进行双向通讯，之后根据要求运行Appium回放脚本。

4.1.2　设备监听子模块详细设计与实现

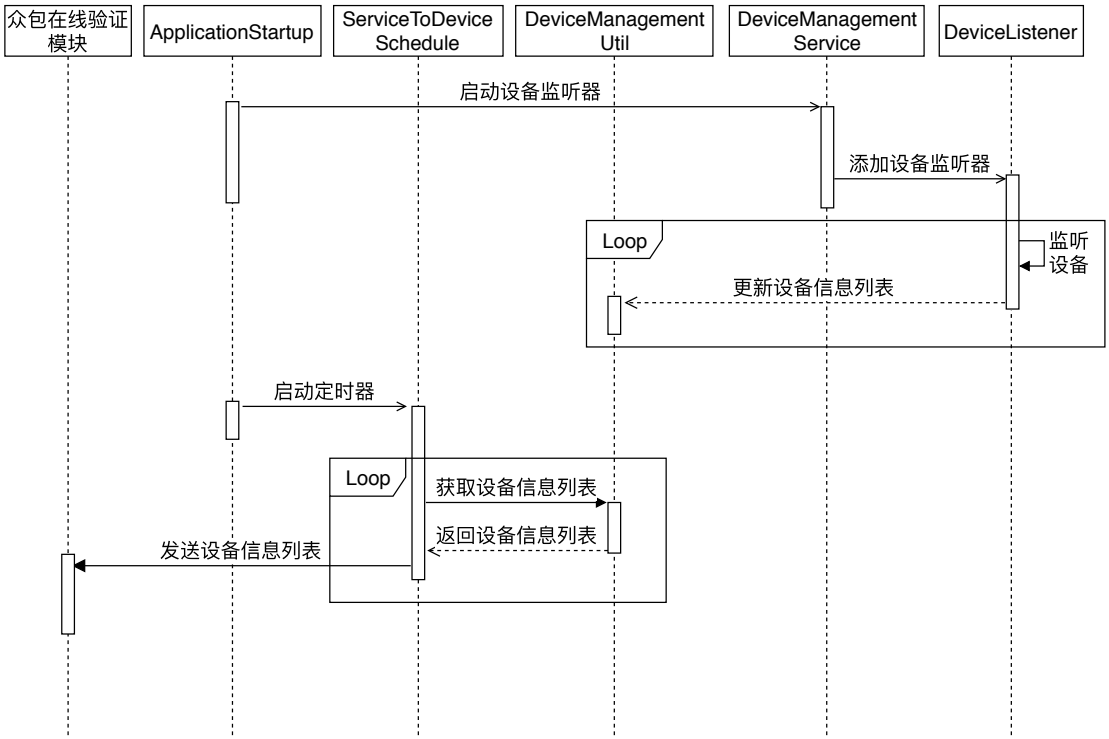

图 4.1: 设备监听子模块时序图

设备监听子模块负责在项目启动后启动设备监听器，对通过USB与服务器进行连接的设备进行监听，并根据监听情况修改设备信息列表。如图 4.1所

示，ApplicationStartup是系统启动监听器，当微服务启动完成后，它会启动定时器ServiceToDeviceSchedule和设备监听器服务DeviceManagementService。定时器在启动后会根据设置的轮询时间定期从DeviceManagementUtil中保存设备信息的List<Device>静态变量中获取当前设备信息列表并将列表发送给众包在线验证模块，设备监听服务则添加设备监听器DevcieListener对设备进行监听，更新设备信息到DeviceManagementUtil。

设备监听子模块中涉及到的主要类如表 4.1所示，对类的名称、分类以及作用进行描述。主要有实现监听器功能的ApplicationStartup类与DeviceListener类，实现定时器功能的ServiceToDeviceSchedule类，实现具体逻辑的服务类DeviceManagementService，封装AndroidDebugBridge并提供操作监听器功能的AndroidDebugBridgeWrapper类，对设备信息列表进行管理的管理工具类DeviceManagementUtil类以及存储设备信息的设备实体类Device。

表 4.1: 设备监听主要类

类名称	分类	作用
ApplicationStartup	监听器	对系统启动进行监听，在系统启动完成后执行相关逻辑
ServiceToDeviceSchedule	定时器	定期发送设备信息列表
DeviceListener	监听器	对设备连接情况进行监听
DeviceManagementService	服务类	单例模式，启动设备监听器服务
AndroidDebugBridgeWrapper	包装类	对android库的AndroidDebugBridge进行封装，提供对监听器的操作方法
DeviceManagementUtil	工具类	管理设备信息列表
Device	数据类	设备实体，保存设备信息

模块中这些主要类的关系如图 4.2所示，ApplicationStartup实现系统自带监听器ApplicationListener的onApplicationEvent方法，在系统启动完成后该方法会被执行，方法内部启动设备监听器和定时器逻辑就会被系统处理。定时器定期将设备微服务在Eureka注册中心注册的服务名和当前设备信息列表List<Device>一起通过HTTP传给众包在线验证模块。设备监听器则是创建DeviceManagementService单例，配置并启动android库中的安卓调试桥AndroidDebugBridge，添加实现AndroidDebugBridge的IDeviceChangeListener接口的监听器DeviceListener，当设备连接或断开时在相应函数中调用DeviceManagementUtil提供的方法获取、保存或移除设备信息，实现设备信息列表的更新。Device是设备实体类，保存单一设备信息，DeviceManagementUtil则保存当前设备信息列表，并以静态方法形式提供设备信息列表的增删改查操作。

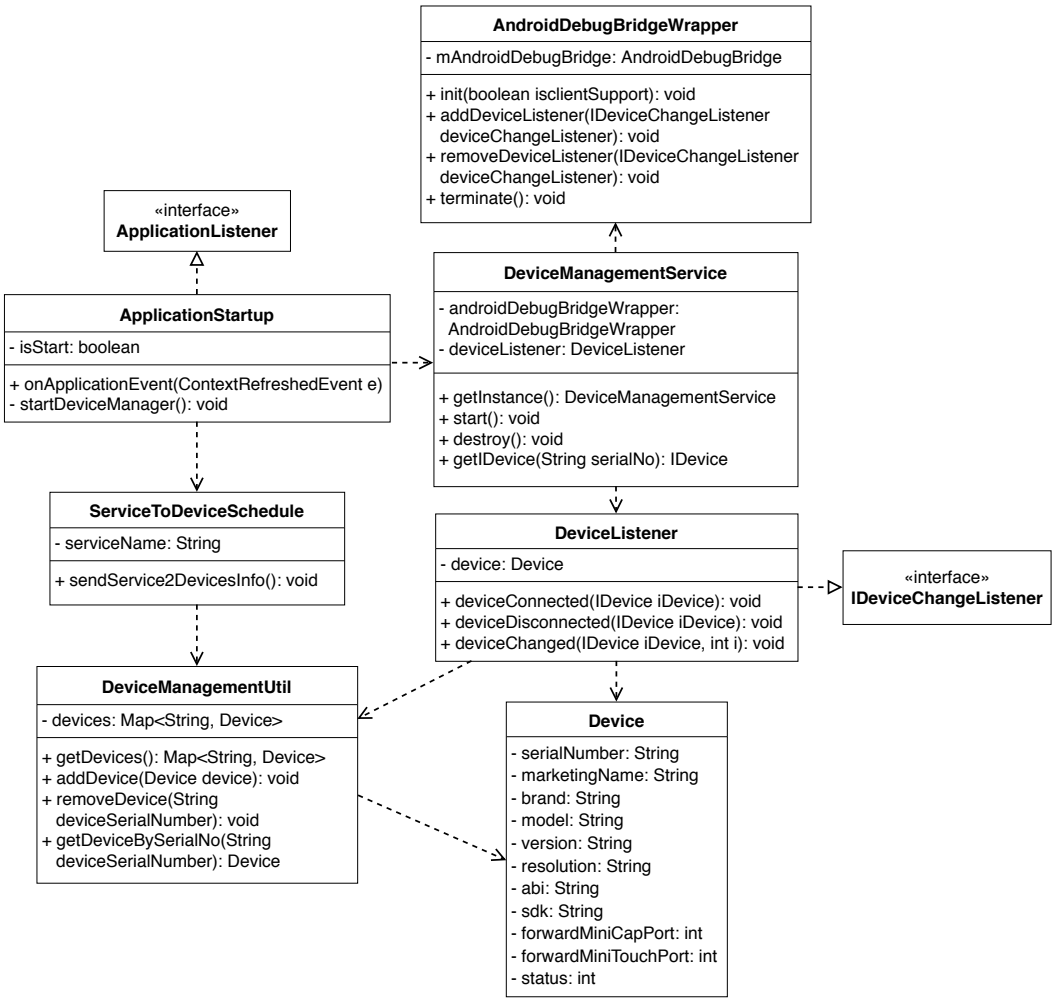

图 4.2: 设备监听子模块类图

所有设备信息都是在设备监听器中处理得到，因此设备监听器的正常启动与运行是之后能够搜索到设备并与其进行交互的保障。设备监听器相应配置及启动相关代码如图 4.3所示。

```
//配置设备监听器并启动
public void start() {
    androidDebugBridgeWrapper = new AndroidDebugBridgeWrapper();
    deviceListener = new DeviceListener();
    androidDebugBridgeWrapper.addDeviceListener(deviceListener);
    androidDebugBridgeWrapper.init(false);
}
```

图 4.3: 设备监听器配置和启动相关代码

4.1.3　设备交互子模块详细设计与实现

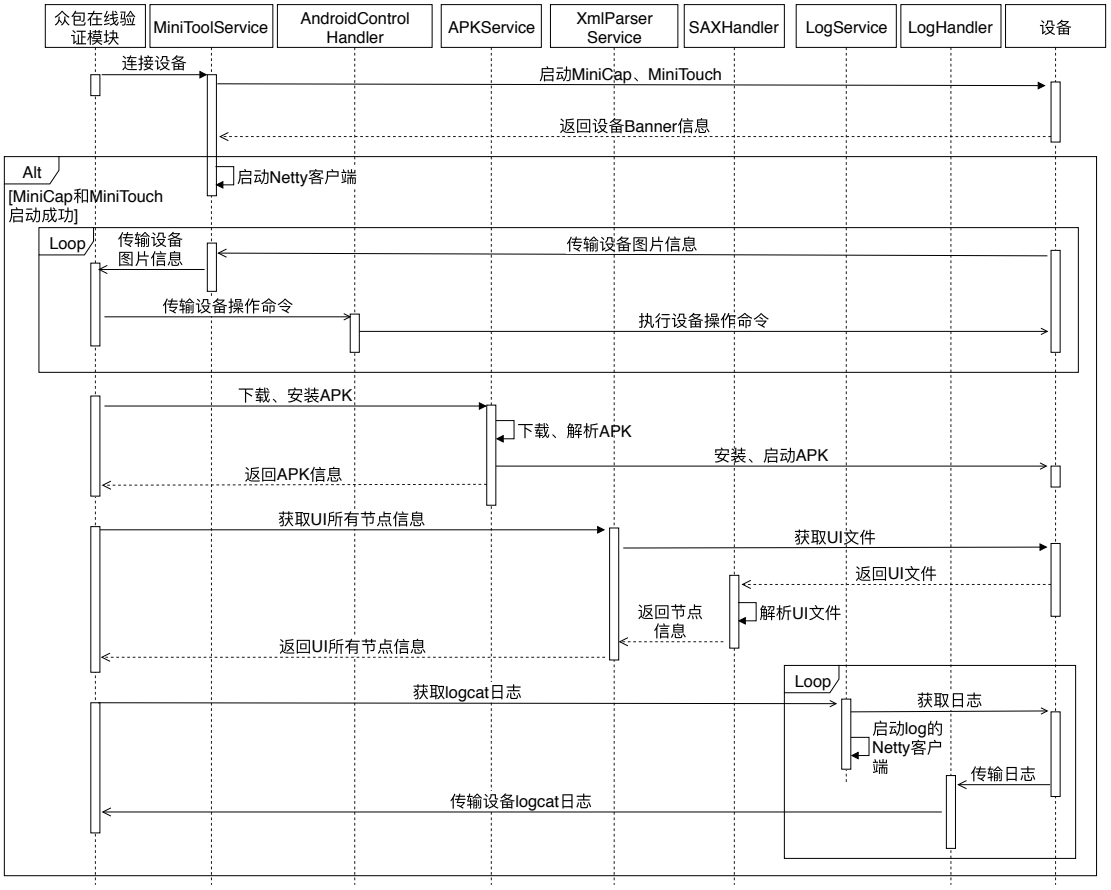

图 4.4: 设备交互子模块时序图

如图 4.4所示是设备交互子模块的时序图，该模块作为设备微服务模块的核心部分，保持设备与主平台之间的通信，保证设备操控的正常性。当众包在线验证模块发送设备连接请求过来后，该模块就会创建MiniToolService的一个实例，然后执行在设备上启动MiniCap和MiniTouch的命令。在这两工具正常启动后，启动Netty客户端并与主平台的Netty服务端建立长连接，通过连接让主平台与设备进行数据传输。之后APKService负责对待测应用进行下载和解析，并安装到设备上启动，XmlParserService则调用SAXHandler获取设备UI结构文件并解析成所有节点信息传给主平台。LogService则根据请求获取设备中的logcat日志，并启动另一个Netty客户端在LogHandler中处理和发送数据。该模块主要涉及到的类描述如表 4.2所示。

表 4.2: 设备交互主要类

类名称	分类	作用
DeviceController	控制类	接收众包在线验证模块的发送请求并响应
MiniToolService	服务类	将设备与众包在线验证模块进行连接并保持
AndroidControlHandler	数据处理类	发送图片信息，接收设备操作命令并执行
APKService	服务类	对APK进行下载、安装、启动等操作
XmlParserService	服务类	获取设备UI结构文件并获取解析内容
SAXHandler	数据处理类	对UI结构问件进行解析，得到节点信息列表
LogService	服务类	开始或停止获取设备logcat日志
LogHandler	数据处理类	传输logcat日志信息
APKUtil	工具类	对APK进行解析，获取APK信息
PortManagementUtil	工具类	对MiniCap和MiniTouch 在服务器上的映射端口进行管理
ApkInfo	数据类	APK信息实体
Banner	数据类	设备相关物理数据实体
Command	数据类	设备操作命令实体
UINode	数据类	UI控件节点信息实体

DeviceController作为控制层类接收HTTP请求并返回结果，MiniToolService、APKService、XmlParserService和LogService类分别是包含与设备进行交互、对待测应用进行下载安装、对设备界面UI结构文件进行解析和获取设备logcat日志所需实现逻辑的服务类。数据处理类AndroidControlHandler是Netty建立连接后在ChannelPipeline中添加的ChannelHandler实现类，在双方通信通道中接收或发送数据，SAXHandler类则是对界面UI结构文件进行解析，APKUtil和PortManagementUtil工具类分别对APK和设备数据通信端口进行管理，ApkInfo、Banner、Command和UINode类则是实体类。这些类的主要关系如图 4.5所示。

DeviceController向众包在线验证模块提供HTTP调用接口，根据不同请求调用不同服务类方法。连接设备进行交互由MiniToolService实现，对每个连接设备都有一个MiniToolService实例与其对应。该类的连接设备函数中首先会执行一条ADB验证命令判断设备中是否有MiniCap和MiniTouch，若无则根据设备的SDK版本以及CPU的ABI架构从工具文件中找到匹配的minicap.so 文件以及MiniCap和MiniTouch的可执行文件，然后将它们通过ADB的push命令上传到设备上启动。由于需要获取到MiniCap传输的数据和使用MiniTouch，因此在启动后还需要进行端口转发，将设备中MiniCap和MiniTouch使用的端口映射到服务器端口，这时使用PortManagementUtil对端口进行分配和回收来保障服务器端口之间不被占用。PortManagementUtil在初始化时默认每个服务器最多连接100台设备，并分配1300~1399端口给MiniCap，1400~1499端口给MiniTouch使用。

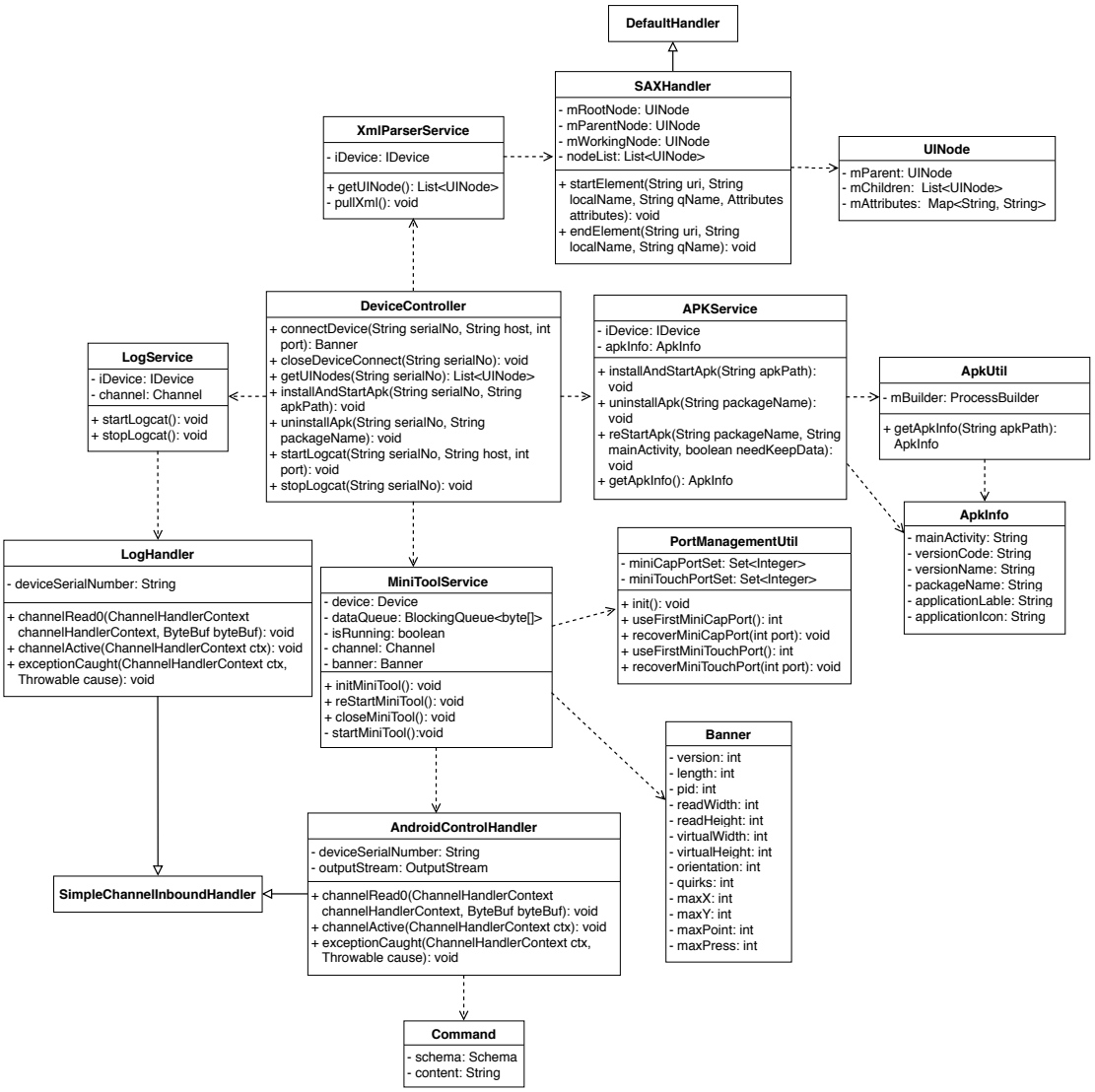

图 4.5: 设备交互子模块类图

在完成MiniCap和MiniTouch的端口映射之后，MiniToolService相关函数中就会启动三个线程：第一个线程配置并启动Netty客户端，与众包在线验证模块中启动的Netty服务端建立连接，这时Netty会分配一个channel作为表示该客户端的唯一标志，并由系统将该channel变量保存起来；第二个线程负责监听MiniCap映射的服务器端口来获取图片信息并保存到队列中；第三个线程则负责每次从队列中拿出图片数据，对数据进行处理，并在Netty连通后通过channel发送数据。由于MiniCap获取设备图片信息以及传输该数据很快，如果处理不及时，会造成在一次数据获取时会将它与之后又发送过来的数据一起处理，造成图片解析错误，数据获取与数据处理传输通过两个线程分别实现就

保证了数据传输正常。其中MiniCap传回的信息有三种：Banner模块、携带图片大小信息和图片二进制信息模块以及只携带图片二进制信息模块。所以需要对信息进行处理获取真正图片数据，再传输数据。处理和传输数据的相关代码如图 4.6所示。

```
//图片数据处理、传输
public void run() {
    while (isRunning) {
        …//从队列中获取图片数据代码省略
        for (int cursor = 0;cursor < length;) {
            int ch = buffer[cursor] & 0xff;
            if(readBannerBytes < bannerLength) {
                cursor = parserBanner(cursor, ch);     //Banenr 模块信息读取
            } else if(readFrameBytes < 4) {
                // 第二次的缓冲区中前 4 位数字和为 frame 的缓冲区大小
                frameBodyLength += (ch << (readFrameBytes * 8));
                cursor++;
                readFrameBytes++;
            } else {
                if(length - cursor >= frameBodyLength) {
                    byte[] subByte = subByteArray(buffer, cursor,
                        cursor + frameBodyLength);
                    frameBody = byteMerger(frameBody, subByte);
                    if(frameBody[0] != -1 || frameBody[1] != -40)
                        return;     //framebody 不是以 JPG 头格式开始
                    final byte[] finalBytes = subByteArray(frameBody, 0,
                        frameBody.length);
                    //通过 Netty 的 channel 传输数据
                    if(channel != null)
                        channel.writeAndFlush(Unpooled.copiedBuffer(finalBytes));
                    …//数据重新初始化代码省略
                }
                …//一张图片的数据解析完成后操作代码省略
            }
        }
    }
}
```

图 4.6: MiniCap图片数据处理和传输相关代码

AndroidControlHandler作为Netty客户端的ChannelHandler的一个实现类对传输数据进行处理。它添加在Netty客户端的ChannelPipeline中，Netty客户端在刚开始建立连接后就会通过该类主动告知服务端当前设备序列号，让服务端将设备序列号与属于该客户端的channel变量对应并保存起来，保证数据

传输目的地正确。之后它会对Netty客户端的入站事件进行处理，主要就是接收并处理来自服务端的设备操作命令数据。目前设备操作命令格式有两种：minitouch://xxx和keyevent://xxx。命令执行方式根据前缀来区分，第一种通过socket传给MiniTouch执行，第二种则执行ADB命令。Netty客户端处理入站事件相关代码如图 4.7所示。

```
@Override
protected void channelRead0(ChannelHandlerContext channelHandlerContext,
                            ByteBuf byteBuf) throws Exception {
    …//数据解析成 Command 对象代码省略
    //keyevent 格式执行 ADB 命令，minitouch 格式传给 MiniTouch 工具处理
    if(command.getSchema().equals(Schema.KEYEVENT)) {
        IDevice device = DeviceManagementService.getInstance().
                getIDevice(deviceSerialNumber);
        if(device != null)
            CommandUtil.executeShellCommand(device, command.getContent());
    } else if (command.getSchema().equals(Schema.MINITOUCH)){
        outputStream.write(command.getContent().getBytes());
        outputStream.flush();
        String endCommand = "c\n";     //minitouch 每条命令后面加上 c 操作
        outputStream.write(endCommand.getBytes());
        outputStream.flush();
    }
}
```

图 4.7: Netty客户端处理入站事件相关代码

在设备连接成功并能在Web端显示设备界面后，众包在线验证模块会通过DeviceController通知对应设备微服务模块去下载、安装并启动APK，这时由设备微服务模块中的APKService根据APK地址下载APK到本地存储，并在APKUtil类中通过aapt工具命令对APK进行解析，获取APK的启动类和包名等，执行ADB命令安装并通过启动类名启动应用，最后再将存储APK信息的ApkInfo实体返回给Web端使用。

设备连接期间，用户可在录制时获取设备日志，这时LogService就被调用，启动另一个Netty客户端，与属于log的服务端建立连接，之后通过启动android库中的LogCatReceiverTask，通过监听器获取logcat信息，构成json格式后通过Netty发送给服务端，再由服务端发给Web端展示。在录制期间，用户每次操作后会调用XmlPaserService从设备中获取当前UI结构文件，命名为ui.xml，之后在SAXHandler中使用SAX解析ui.xml文件，最终获取UINode列表，并最终返回给Web 端使用。

4.1.4　脚本回放子模块详细设计与实现

脚本回放子模块负责配置和启动Appium，执行脚本。该模块的时序图如图4.8 所示。ScriptController为控制类，对外提供HTTP接口。ScriptService则包含执行回放的主要逻辑，通过调用AppiumManager配置信息并启动Appium Server，在Appium启动成功后将脚本交于它执行。

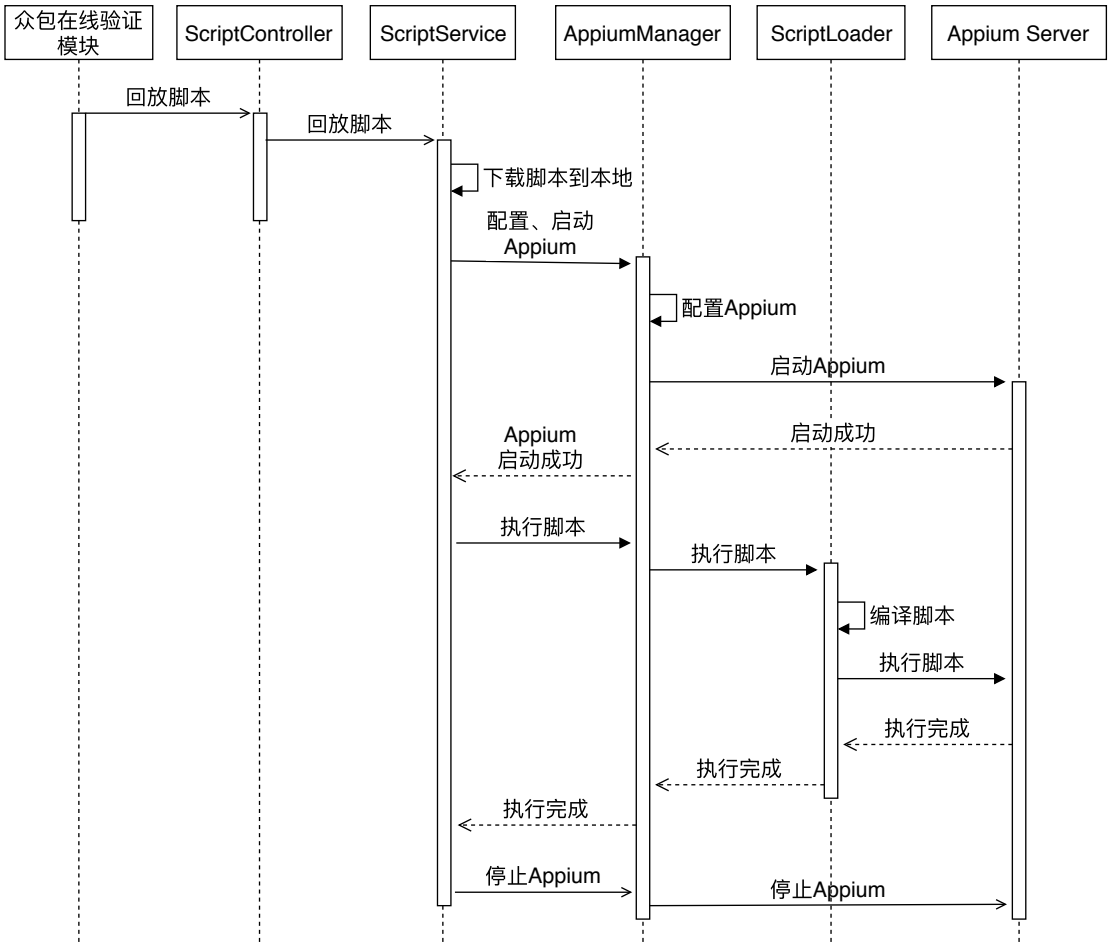

图 4.8: 脚本回放子模块时序图

脚本回放子模块中涉及到的主要类说明如表 4.3所示。ScriptController类接收用户发送的脚本操作相关的HTTP请求并响应。ScriptService服务类实现脚本回放的执行逻辑。AppiumManager类包含Appium客户端与服务端逻辑，负责配置和启动Appium。ScritLoader类则是对Java版测试脚本文件进行加载编译，交由Appium执行。

表 4.3: 脚本回放主要类

类名称	分类	作用
ScriptController	控制类	提供脚本操作的HTTP接口
ScriptService	服务类	提供脚本回放服务
AppiumManager	服务类	配置和启动Appium
ScriptLoader	服务类	对Java类文件加载编译，最后交由Appium执行

模块类图如图 4.9所示。开始进行脚本回放时，设备微服务根据URI找到ScriptController中的HTTP接口函数，调用ScriptService的playBackScript方法，在方法里首先对传输过来的MultipartFile类型脚本文件进行下载，存储到本地，之后通过AppiumManager配置AppiumDriver信息，并根据操作系统执行已写好的bat文件或者shell文件，启动Appium，在启动完成后提供脚本文件本地存储地址，由ScriptLoader对脚本文件进行编译，最后由AppiumDriver执行文件。

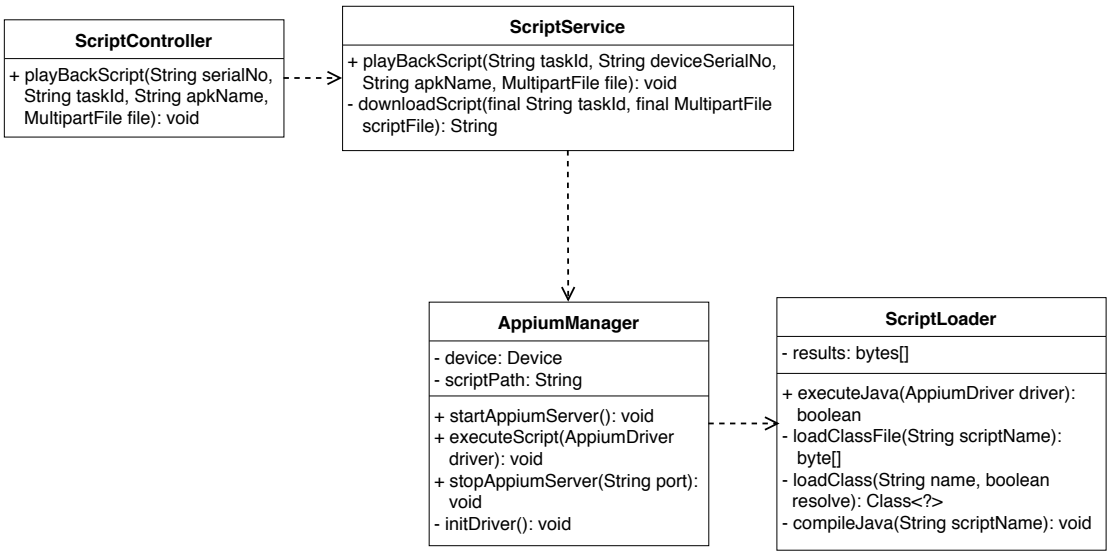

图 4.9: 脚本回放子模块类图

4.2　众包在线验证模块

4.2.1　众包在线验证模块介绍

众包在线验证模块作为系统主平台，包括Web端以及服务端，Web端与用户进行交互，服务端则向Spring Cloud注册中心注册，响应用户操作，根据需要从注册中心获取设备微服务实例并调用。该模块对所有设备信息进行统一管理，启动Netty服务端作为消息中心保证设备与Web端交互正常，让用户能正常操控

设备并看见同步设备界面。模块还在Web端提供脚本操作界面，并在最后将用户验证结果存储到MySQL数据库中。

4.2.2　设备管理子模块详细设计与实现

设备管理子模块对所有设备信息进行管理，保证当前设备信息都可用。如图 4.10 所示，每个正常运行的设备微服务会发送设备信息列表给该模块，并由DeviceServiceController类接收到，调用DeviceService存储服务-设备信息列表到DeviceServiceUtil的静态变量中。同时，模块也会通过定时器ServiceToDevicesHandleSchedule定期发送心跳到每个设备微服务，对无响应的微服务清除属于该服务的服务-设备信息列表数据。当用户要查看所有设备信息列表时，用户请求会由DeviceController响应，由它调用DeviceService从DeviceServiceUtil中获取所有设备信息列表，返回到界面上展示给用户。

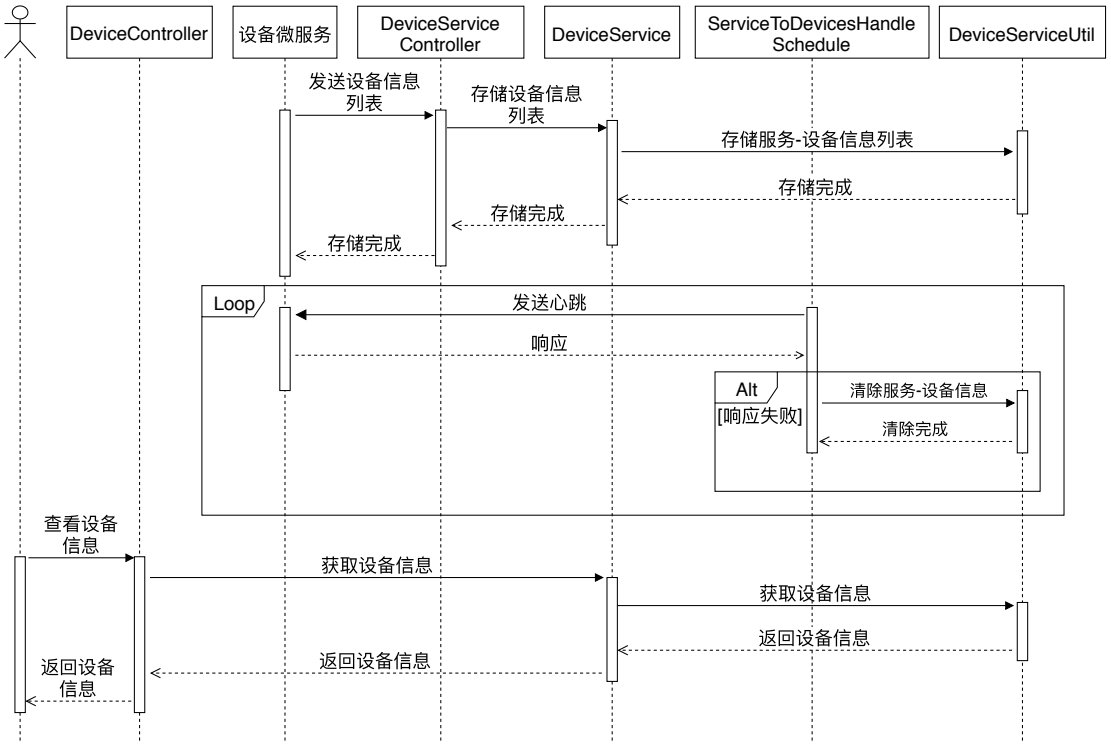

图 4.10: 设备管理子模块时序图

设备管理子模块中涉及到的主要类如表 4.4所示，描述类的名称、分类以及作用。DeviceServiceController控制类是接收来自设备微服务的请求消息，DeviceController类则是提供HTTP接口给Web端，响应用户请求。服务接口

类DeviceService提供设备操作的接口，具体接口逻辑由DeviceServiceImpl类实现。ServiceToDevicesHandleSchedule则是定时器，通过配置定期向每个设备微服务发送心跳并对响应结果进行处理。DeviceServiceUtil对服务与设备信息列表的映射关系进行存储管理，Device是设备实体类。

表 4.4: 设备管理主要类

类名称	分类	作用
DeviceServiceController	控制类	提供HTTP REST接口，响应来自设备微服务的请求
DeviceController	控制类	提供HTTP的REST接口，响应来自Web端的请求
DeviceService	服务接口类	提供与设备相关操作的方法接口
DeviceServiceImpl	服务实现类	实现DeviceService 中的接口方法
ServiceToDevicesHandleSchedule	定时器	定期向设备微服务发送心跳
DeviceServiceUtil	工具类	对所有服务-设备信息列表进行管理
Device	数据类	设备实体，保存设备信息

这些类的关系如图 4.11所示。设备微服务会有定时器定期发送该服务器上的连接设备信息列表给众包在线验证模块，这时模块的DeviceServiceController类会接收到这些设备信息，之后调用DeviceService处理，过程中构建接口实现类DeviceServiceImpl的实例实现具体的设备信息更新处理逻辑。

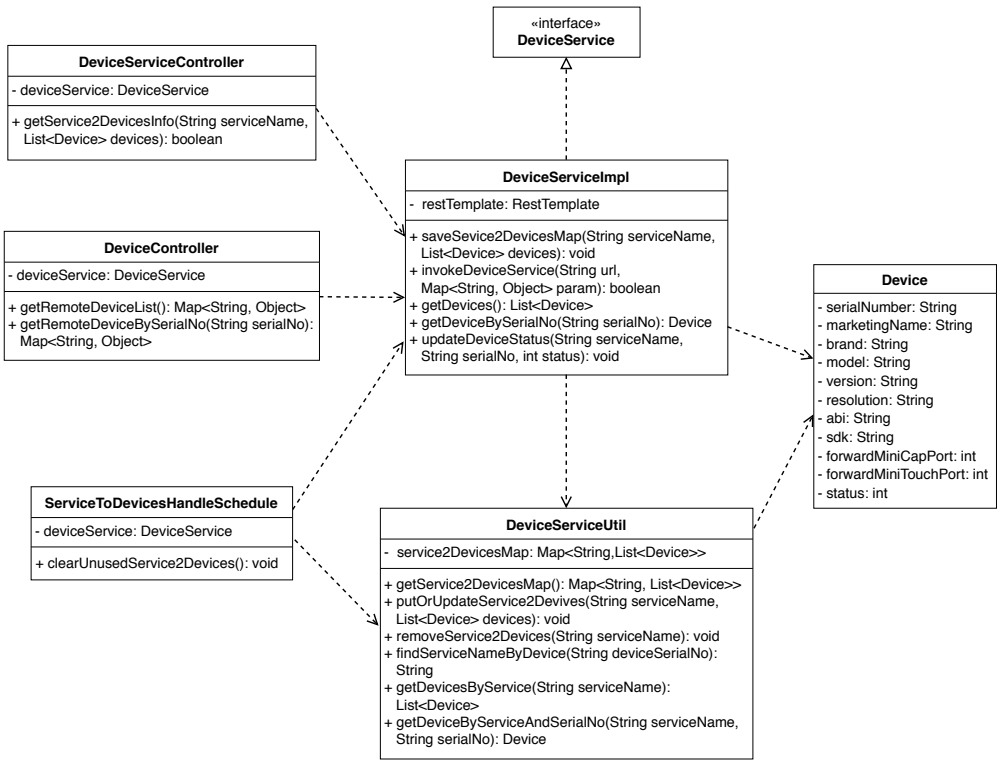

图 4.11: 设备管理子模块类图

DeviceServiceUtil内部保存了Map<String, List<Device>>类型的服务-设备信息映射总表，Map的键为在Spring Cloud微服务框架上注册的服务名，值为该服务的设备信息列表。DeviceServiceImpl就是通过DeviceServiceUtil提供的相关操作方法对总表信息进行更新。DeviceController则会响应用户查看设备信息列表请求，从DeviceServiceUtil中获取所有的设备信息列表List<Device>，返回给Web端显示。ServiceToDevicesHandleSchedule则定期发送心跳，对响应失败的设备微服务则清除总表中的服务-设备信息列表信息。该定时器的主要代码如图 4.12所示。

```
//配置定时器，配置轮询时间为 8 秒
@Scheduled(cron="0/8 * *  * * ? ")
public void clearUnusedService2Devices() {
    try {
        //遍历设备微服务名并发送心跳
        for (Map.Entry<String, List<Device>> entry : DeviceServiceUtil.
                getService2DevicesMap().entrySet()) {
            if(!deviceService.invokeDeviceService("http://" + entry.getKey() +
                    ApiConstants.HEART_BEAT, null)) {
                DeviceServiceUtil.removeService2Devices(entry.getKey());
            }
        }
    } catch (Exception e) {
        log.error("ServiceToDevicesHandle schedule error");
    }
}
```

图 4.12: 设备管理定时器相关代码

4.2.3　设备操控子模块详细设计与实现

设备操控子模块在设备微服务与设备进行通信连接后，作为消息中心将图片信息和设备操作命令中转到它们的目的地，保证设备微服务与Web端可以时刻保持数据交换直到操控流程结束。该模块的时序图如图 4.13所示，系统在启动后就会通过AndroidControlServer类分别开启与设备微服务模块以及与Web端通信的Netty服务端。用户选择设备使用后，Web端发送HTTP请求调用DeviceController 接口，通过服务类找到设备对应的设备微服务并让设备微服务中的设备交互子模块开始运行。流程全部运行完成后，设备与Web端就间接地建立起连接通道，并分别保存设备序列号与Netty分配的代表Web端的channel和代表设备端的channel之间的映射关系在DeviceChannelUtil中。在连接完成后，用户需要在设备上操作安卓APK应用，通过调用HTTP接口

发送APK下载地址给设备微服务，由设备微服务对APK进行下载安装，并在设备中启动，应用在下载安装过程中可在Web端查看到进度。最后则由SocketServerHandler和WSSocketServerHandler处理Netty入站事件，对来自两边的数据进行中转，传往目的端处理。

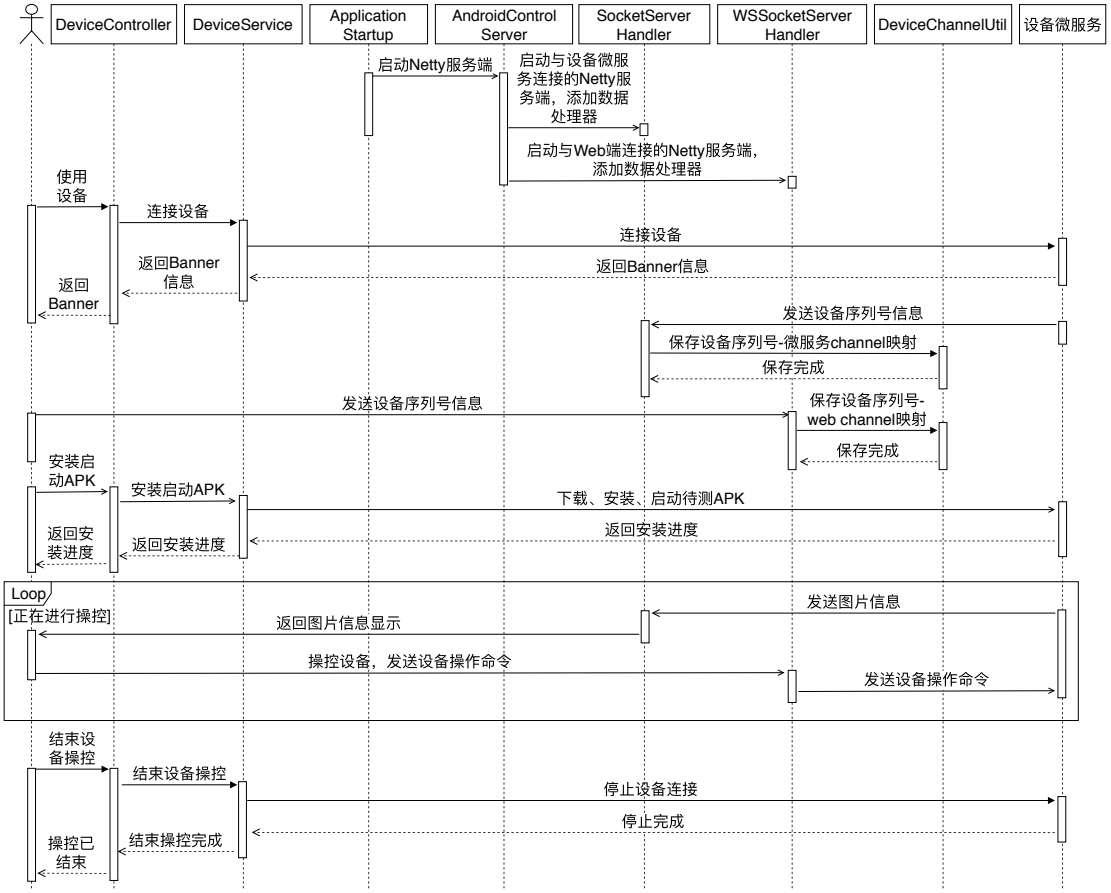

图 4.13: 设备操控子模块时序图

系统中实现该模块所涉及到的主要类说明如表 4.5所示。分别介绍了能提供与设备操作相关的HTTP接口的控制类DeviceController，实现监听器功能的ApplicationStartup类，将设备相关操作功能封装并提供给DeviceController调用的DeviceService服务接口类，实现具体接口功能的DeviceServiceImpl类，启动Netty服务端的AndroidControlServer类，接收和处理Netty传输数据的SocketServerHandler类和WSSocketServerHandler类以及保存管理设备序列号与channel映射的工具类DeviceChannelUtil。

表 4.5: 设备操控主要类

类名称	分类	作用
DeviceController	控制类	提供设备操作相关的HTTP 接口
ApplicationStartup	监听器	对系统启动进行监听，在系统启动完成后执行相关逻辑
DeviceService	服务接口类	提供与设备相关操作的方法接口
DeviceServiceImpl	服务实现类	实现DeviceService 接口
AndroidControlServer	服务类	启动设备操控的Netty 服务端
SocketServerHandler	数据处理类	接收和处理来自设备微服务的数据
WSSocketServerHandler	数据处理类	接收和处理来自Web端的数据
DeviceChannelUtil	工具类	管理设备序列号与channel的映射

如图 4.14所示展示该模块的主要类关系。ApplicationStartup监听系统启动，在系统启动后创建线程开启Netty服务端，启动的具体逻辑在AndroidControlServer里面实现。系统共启动两个服务端，一个作为Web端的Netty服务端，一个作为设备微服务的服务端。两个服务端分配两个不同端口绑定，并配置ChannelPipeline。SocketServerHandler和WSSocketServerHandler则是配置在ChannelPipeline中的ChannelHandler，分别接收并处理Web端和设备微服务端的数据。

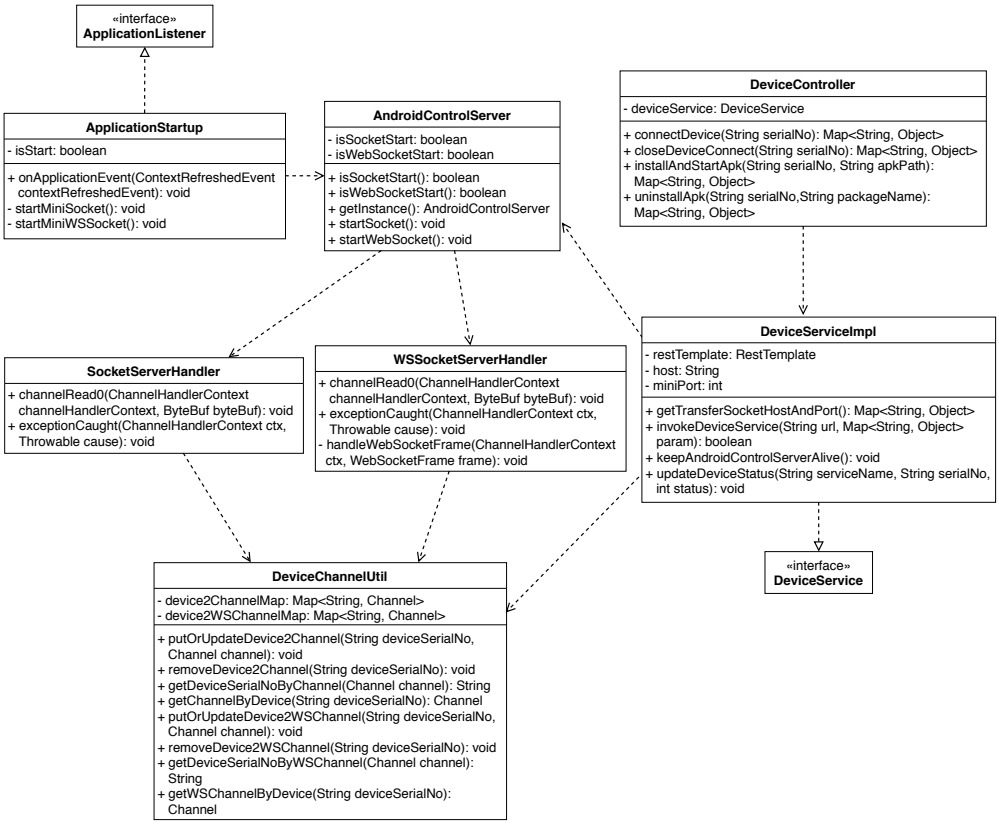

图 4.14: 设备操控子模块类图

在每次用户开始使用设备时，设备操控子模块都会先判断启动Netty的线程是否已经中断结束，若已中断则调用AndroidControlServer中的方法重新启动Netty服务端，之后Web端以及微服务端在连通服务端后第一次会发送设备序列号数据，由两个数据处理类接收并同发送该数据的channel变量被保存到DeviceChannelUtil的静态Map变量中，等待系统对其进行操作。

用户使用设备进行操控后，Web端会创建WebSocket实例，与对应Netty服务端建立连接，接收或传输数据。设备微服务端发送过来的图片信息是一串字节流，在Web端显示时需要将其进行转换缓存后才能显示在网页上。界面使用canvas作为图片显示区，在里面添加Image对象更新替换上一个对象，Image对象的src则为缓存图片地址，具体实现逻辑如图 4.15所示。

```
//二进制数据转为 image/jpeg 显示在 canvas 上
onBinaryData(data:any) {
    var th = this;
    var cvs = <HTMLCanvasElement> document.getElementById("cvs");
    var blob = new Blob([data], {type: 'image/jpeg'});
    var URL = window.URL;
    var imgObj = new Image();
    imgObj.src = URL.createObjectURL(blob);
    //待图片加载完后，将其显示在 canvas 上
    imgObj.onload = function(){
        var ctx = cvs.getContext('2d');
        if(!th.imgWidth) {
            th.imgWidth = imgObj.width;
        }
        if(!th.imgHeight) {
            th.imgHeight = imgObj.height;
        }
        th.cvsHeight = Math.round((th.cvsWidth / th.imgWidth) * th.imgHeight);
        ctx.drawImage(imgObj, 0, 0, th.cvsWidth, th.cvsHeight);
        imgObj.onload = null;
        imgObj = null;
        blob = null;
    }
}
```

图 4.15: Web端设备屏幕展示相关代码

canvas不断刷新图片实时展示设备屏幕，在Web端还对它进行鼠标和键盘事件监听，实现用户触屏模拟操作。在每次鼠标按下、移动、释放事件触发后，Web端会将事件转为MiniTouch可执行的命令格式: d、m、u、c命令并添加minitouch://前缀表示其类型，最后通过建立的WebSocket将命令传

给Netty服务端，再由另一个服务端传给设备微服务执行。Web端还提供菜单键、HOME键、返回键和App Switch键按钮，在用户点击它们或键盘事件触发时，Web端将相应ADB命令添加keyevent://前缀同样通过WebSocket传输过去。操控期间模块通过不断重复这些操作，保证用户在Web端正常操控设备。

4.2.4　脚本操作子模块详细设计与实现

脚本操作子模块负责对脚本进行操作以及录制期间提供logcat查看功能。主要功能体现在Web端，即与用户的交互上，相关后端是对用户与脚本和log正常交互的支撑与保障。该模块的时序图如图 4.16所示。

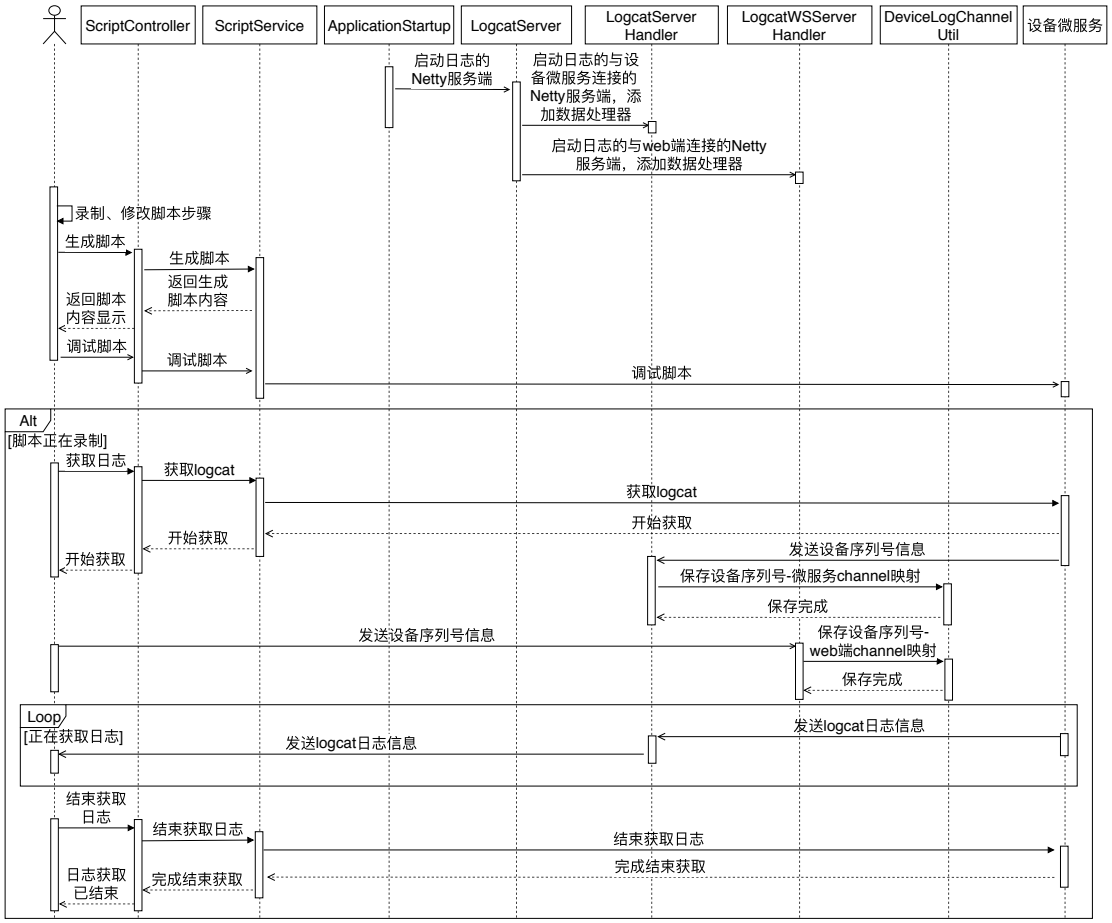

图 4.16: 脚本操作子模块时序图

脚本步骤的录制以及修改在Web端实现，后端不涉及。在用户选择生成脚本后，ScriptController接口被调用，脚本步骤以及APK部分信息传输到后端，之后ScriptService就会根据模板生成脚本文件，并返回脚本内容给Web端显示，同

时提供脚本地址下载。与设备操控时类似，log的获取也是开启Netty服务端，通过与Web端以及设备微服务端进行通信，保证logcat信息能传到Web端展示。脚本操作子模块中涉及到的主要类功能上与设备操控中的涉及类相似，它们的关系如图 4.17所示。

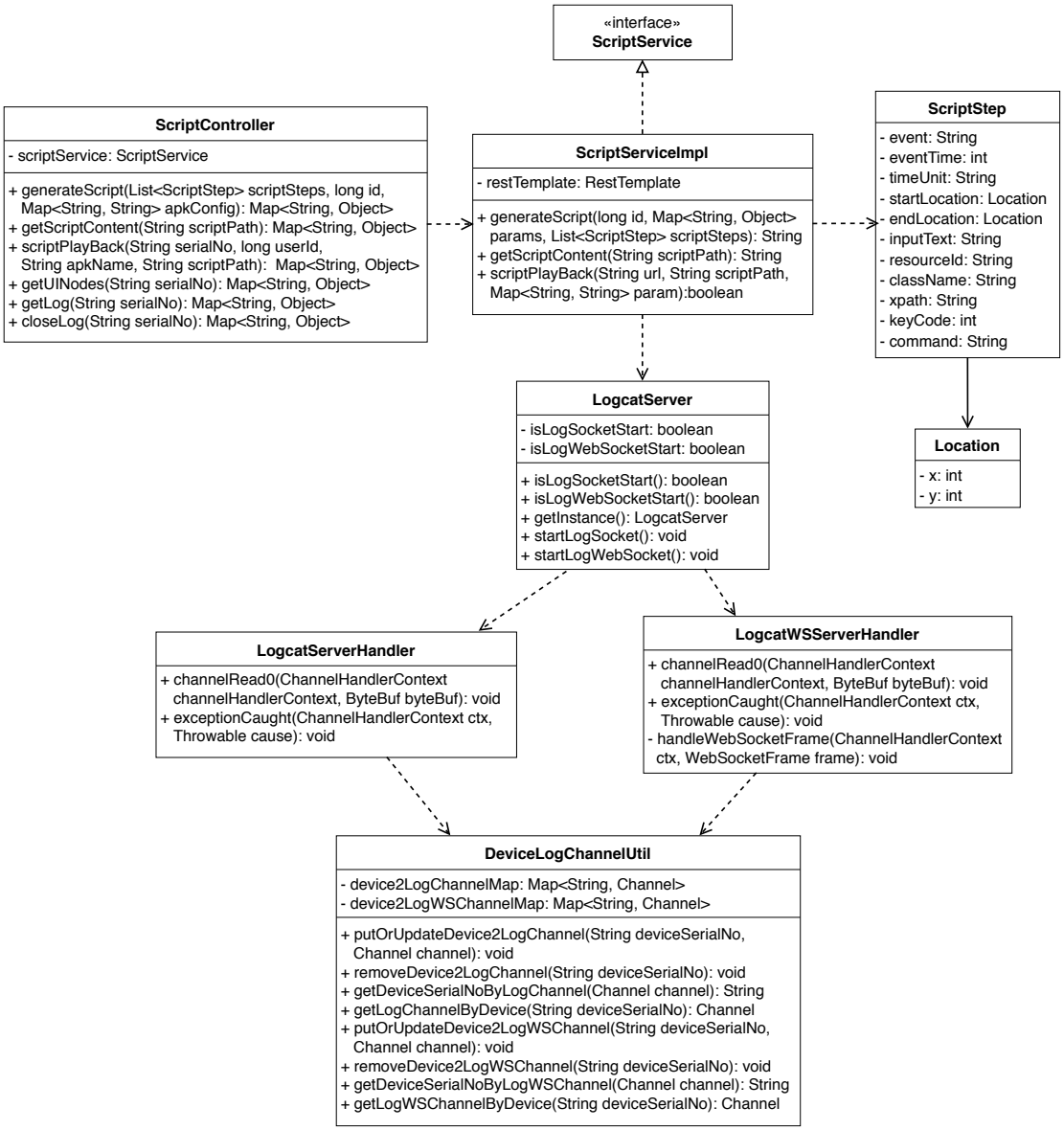

图 4.17: 脚本操作子模块类图

在设备操控正常情况下，用户可进行脚本步骤录制，在录制期间，用户每次用户操作都会在Web端的对应事件监听函数中被转化为一条脚本步骤，步骤实体定义为ScriptStep，主要包含事件类型、界面显示内容、坐标以及控件信

息等。Web端支持5种事件类型：CLICK （点击控件），LONGPRESS （长按控件），SWIPE（滑动），SLEEP（等待），INPUT（输入）以及FASTCUT_PRESS（快捷键按下）。录制期间Web端会在每次用户操作后发出请求，最终由设备微服务的设备交互子模块获取与解析UI结构文件并返回节点信息列表，然后根据鼠标当前坐标获取匹配的控件节点信息，在界面上用红色框突出控件并展示当前控件重要属性。获取匹配控件信息的主要代码如图 4.18所示。CLICK、LONGPRESS和FASTCUT_PRESS事件触发后，该代码就会被执行，并将node部分信息如xpath、text、resource_id等存在ScriptStep中，其中CLICK事件维持时间超过了1秒就定义为LONGPRESS。

```
//获取匹配的控件节点信息
findLeafMostNodesAtPoint(x:number, y:number) {
    var foundNode = null;
    if(this.nodes) {
        this.nodes.forEach(node => {
            if(node.hasBounds) {
                if(node.xPosition <= x && x <= (node.xPosition + node.width)
                && node.yPosition <= y && y <= (node.yPosition + node.height)) {
                    if(foundNode === null) {
                        foundNode = node;
                    } else {
                    if((node.height * node.width) <
                    (foundNode.height * foundNode.width))
                            foundNode = node;
                    }
                }
            }
        });
    }
    return foundNode;
}
```

图 4.18: 获取匹配控件节点信息相关代码

若ScriptStep的事件类型是SWIPE时，则只需保存起始点坐标及结束点坐标即可。INPUT则需要查看上一步步骤信息，若是CLICK则新生成脚本步骤，保存输入内容以及控件节点信息，若是INPUT则在原步骤基础上追加输入内容并保存。SLEEP则是在每次其它事件类型的脚本步骤生成前自动生成，时间数据为本次步骤生成与上次步骤生成之间的时间间隔。脚本步骤生成后，Web界面还提供步骤编辑、移动、删除功能。当至少有一条脚本步骤时，用户可选择将它们转为一个脚本文件，此时由服务实现类ScriptServiceImpl调用FreeMarker包

获取已定制的Appium支持的Java脚本模板appium.ftl，根据脚本步骤类型逐个将它们转为Appium的Java脚本可执行的函数，并插入到appium.ftl文件对应位置中，在相应变量位置替换真正的APK信息，最终生成Java文件并统一命名为Main.java，保证Appium能正确执行脚本。脚本内容则返回给Web端代码高亮显示，并提供下载链接给用户。生成脚本后，用户可对脚本进行调试回放，通过设备微服务的脚本子模块启动Appium执行实现。

录制期间，Web端会显示log展示框并提供log获取按钮，log获取过程与设备图片信息传输类似，LogcatServer负责在系统启动后就启动传输log的Netty服务端，并由LogcatServerHandler和LogcatWSServerHandler分别处理两端数据，获取期间设备中的logcat信息会逐一传到界面上，为了保证界面流畅性，默认每过500条清除当前数据。

4.2.5　缺陷验证子模块详细设计与实现

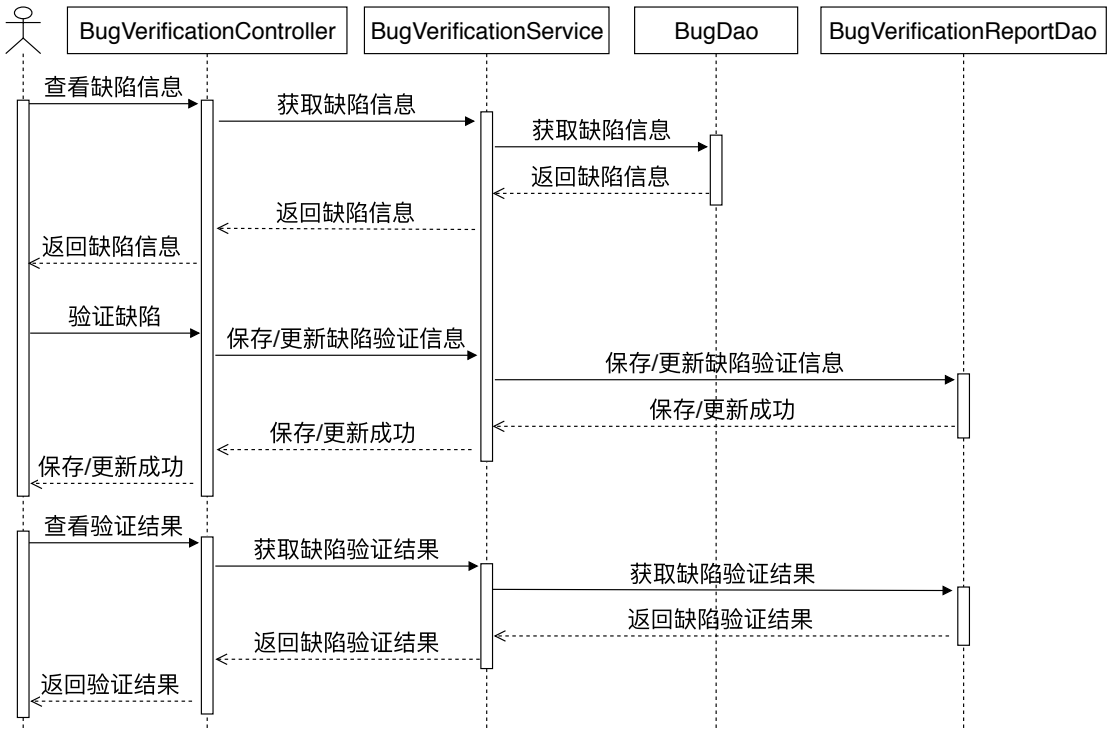

图 4.19: 缺陷验证子模块时序图

缺陷验证子模块的功能是查看缺陷以及对缺陷进行验证，这也是本平台的最终目的，以上子模块的实现都是作为验证手段供用户使用。该子模块的时序图如图 4.19所示。

BugVerificationController提供缺陷查看、缺陷验证信息查看以及验证信息保存与更新功能的接口。BugVerificationService为服务接口，它的实现类则是对这些功能的具体逻辑实现。缺陷信息和缺陷验证信息保存在数据库中，所以由DAO层两个类BugDao和BugVerificationDao访问访问数据库并对数据进行增删改查操作。模块中主要涉及到类说明如表 4.6所示。

表 4.6: 缺陷验证主要类

类名称	分类	作用
BugVerificationController	控制类	提供验证缺陷相关操作的HTTP接口
BugVerificationService	服务接口类	缺陷验证相关逻辑实现接口
BugVerificationServiceImpl	服务实现类	缺陷验证相关逻辑实现
BugVerificationReportDao	数据库访问类	访问数据库，获取表信息
BugDao	数据库访问类	访问数据库，获取表信息
BugVerificationReport	数据类	缺陷验证信息实体
Bug	数据类	缺陷实体

这些类的关系如图 4.20所示，它们主要实现缺陷查询以及缺陷验证结果的基本数据库操作。缺陷验证子模块涉及到众包任务请求者和众包工人两个角色，他们都有权限对缺陷信息进行查看，有区别的地方在于众包工人只能查看和更新自己的缺陷验证结果，而任务请求者有权限查看缺陷所有验证结果。

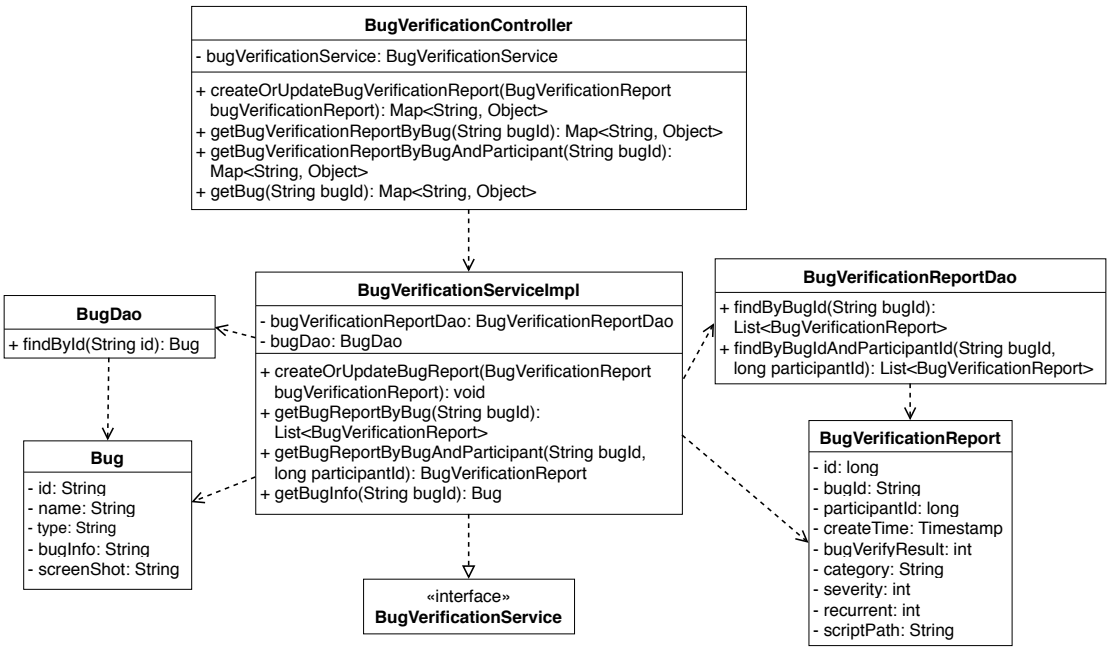

图 4.20: 缺陷验证子模块类图

在众包工人选择设备使用进入到缺陷验证界面后，Web界面会发送获取当前缺陷信息以及众包工人自己的验证信息请求，由BugVerificationController响应，并调用BugVerificationService接口，接口的具体实现逻辑则是通过它的实现类BugVerificationServiceImpl访问DAO层，由DAO层去访问数据库，从相关表中获取缺陷信息以及用户对这个缺陷的验证信息（无验证信息则返回空值）。之后在浏览器界面上提供是缺陷、不是缺陷、不确定三种选项，当众包工人选择“是”时，还需分别从BUG分类、复现程度、严重等级的下拉列表中选择某一个值，完善缺陷信息。众包工人若在验证过程中选择生成脚本过，界面会在他填写验证结果时提示是否选择将生成的脚本作为参考，参考脚本为最近生成的测试脚本文件，并在验证结束后同验证信息一起发送给BugVerificationController，最终存入MySQL数据库中。缺陷验证信息的存储相关代码如图 4.21所示。

```
//保存或更新缺陷验证结果
@Override
public void createOrUpdateBugReport(long userId,
        BugVerificationReport bugVerificationReport) {
    //对于判断不是缺陷或不确定的将缺陷的补充信息设为空
    if(bugVerificationReport.getBugVerifyResult() > 0) {
        bugVerificationReport.setCategory(null);
        bugVerificationReport.setRecurrent(0);
        bugVerificationReport.setSeverity(0);
    }
    bugVerificationReport.setParticipantId(userId);
    bugVerificationReport.setCreateTime(new
            Timestamp(System.currentTimeMillis()));
    bugVerifyService.createOrUpdateBugReport(bugVerificationReport);
    bugVerificationReportDao.save(bugVerificationReport);
}
```

图 4.21: 存储缺陷验证信息相关代码

众包任务请求者在创建缺陷众包项目后就可以在项目开始后时刻查看众包工人提交的缺陷验证结果信息。在任务请求者进入验证结果页面后，Web端会发送用户请求给后端的BugVerificationController处理，最终通过DAO层获取相应数据库数据并返回给Web端，由Web端对数据进行计算，统计缺陷的判断结果以及判断是缺陷时众包工人填写的BUG分类、复现程度和严重等级的占比，通过ECharts绘制相应饼图显示在界面上，同时也将原始验证结果以列表形式展示给任务请求者查看。

4.3　系统运行展示

平台提供给用户的主要有三部分页面。第一部分页面为设备列表页面，众包工人在参与缺陷验证项目后可进入该页面。该页面展示每个连接设备的基本信息如品牌、型号、分辨率等，并对当前可用设备提供“使用”按钮给众包工人，设备正被其他众包工人使用时则提供“占用中”按钮提示。设备列表界面运行展示如图 4.22所示。

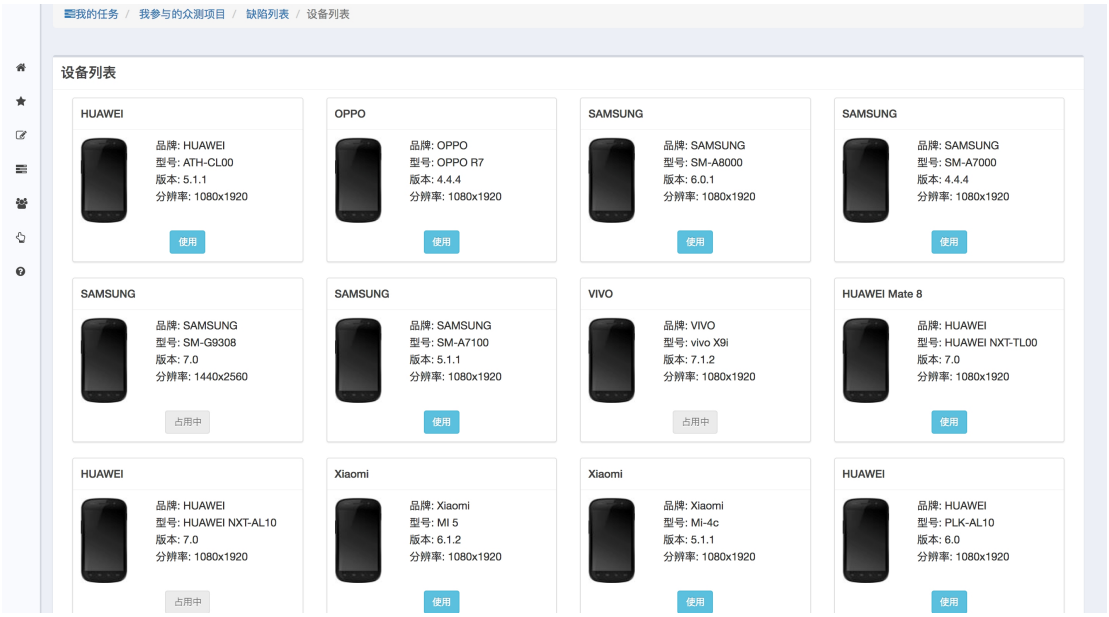

图 4.22: 设备列表界面

第二部分界面是缺陷验证界面，即众包工人进行设备操控和缺陷验证所在界面。该界面包含缺陷基本信息展示区、设备和测试脚本操作区、缺陷验证区以及测试脚本内容展示框。缺陷基本信息展示区展示缺陷的截图、缺陷类型、缺陷内容及自动化中产生缺陷的设备等基本信息。设备和脚本操作区则是体现平台重要功能的区域，也是众包工人主要与界面进行交互的区域，该区域展示设备当前界面并提供鼠标模拟操作和HOME键等四个快捷键，在测试脚本录制时显示脚本步骤，提供脚本步骤的修改、移动、插入、删除、生成和回放操作按钮。同时该区域还展示设备logcat日志信息，也提供“选择手机”按钮供众包工人切换手机使用。缺陷验证区是众包工人提交结果的交互区域，众包工人在缺陷验证界面进行一系列操作后能够得出结果时，就可以在界面的缺陷验证区对缺陷进行验证，填写相关信息，并点击“确定”保存验证结果，返回到众包项目的缺陷列表界面。缺陷验证界面的运行展示如图 4.23所示。

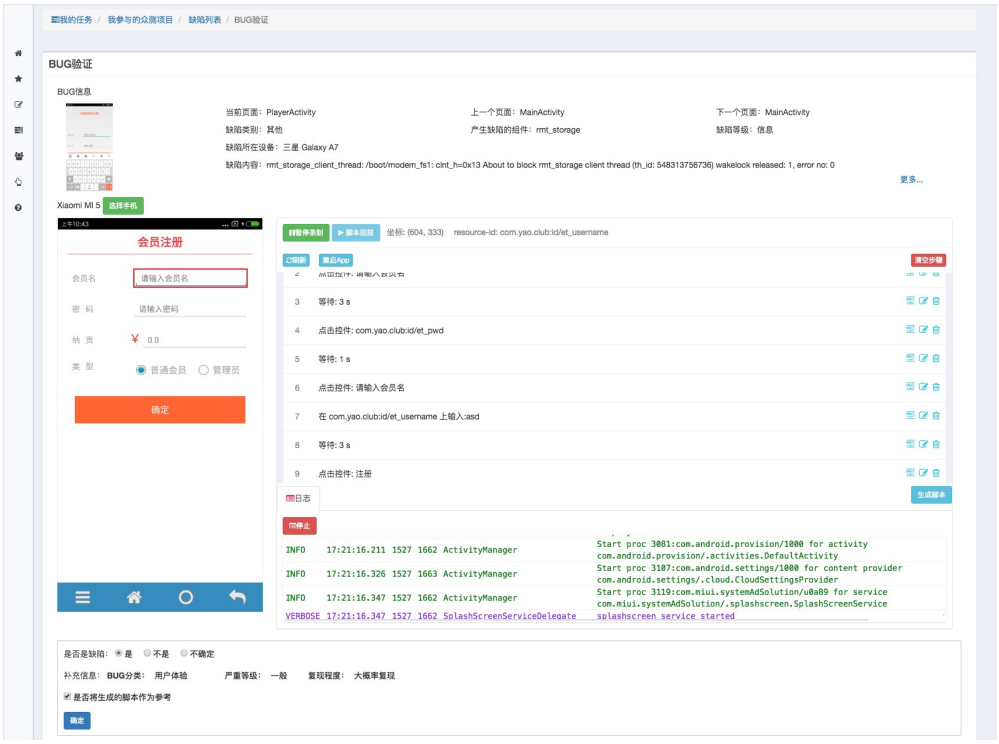

图 4.23: 缺陷验证界面

众包工人在选择生成脚本后，测试脚本内容会以Dialog框的形式展示在界面上给众包工人查看，其中代码会高亮显示。同时Dialog框中还提供“下载”按钮给众包工人来下载脚本到本地。脚本展示框的运行展示如图 4.24所示。

图 4.24: 缺陷验证界面脚本内容展示框

平台最后部分界面是缺陷验证结果界面，该界面只有众包任务请求者才能进入。界面除了给任务请求者展示缺陷的基本信息外，还会展示所有众包工人对该缺陷的验证结果统计图和结果列表，任务请求者可以根据统计图得到缺陷各个结果的数量以及比例，并可在结果列表中看到每个众包工人的提交结果和下载作为参考的测试脚本。该界面实际运行展示如图 4.25所示。

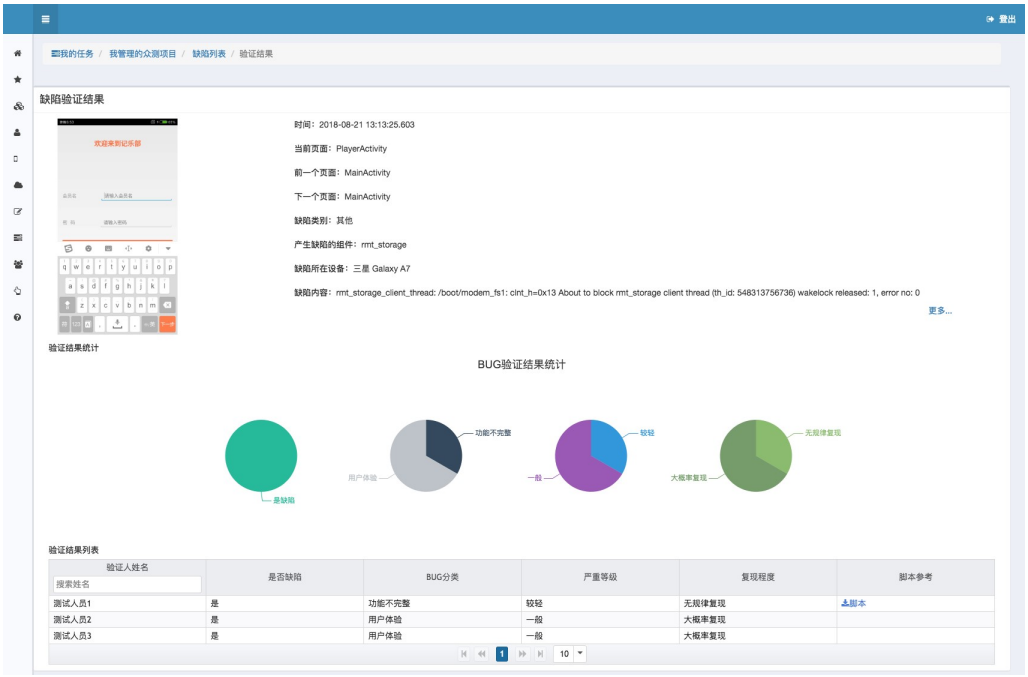

图 4.25: 缺陷验证结果界面

4.4　本章小结

本章主要对平台的设备微服务模块和众包在线验证模块进行介绍，并详细介绍了它们的子模块，通过时序图、类说明表以及类图对这些子模块进行详细设计与实现，并通过一些关键代码展示这些模块内部的具体功能实现细节。本章最后对平台上线后的关键界面运行图进行展示。

第五章　安卓众包在线验证平台的测试与实验设计

5.1　系统测试

5.1.1　测试目标

系统在设计与实现后需要对其基础功能进行单元测试和功能测试。单元测试主要分别测试设备微服务模块和众包在线验证模块中各个最小测试单元类和方法的正确性。功能测试则涉及用户界面的交互，主要是通过在公司服务器上部署安卓众包在线验证平台，由人工实际操作来测试功能是否正确。功能测试的主要目标包括：设备管理功能、设备操控功能、脚本操作功能以及缺陷验证功能，每个功能中有多个小功能点，通过验证所有功能点是否符合设计要求、功能是否正确以及之后对测试发现的缺陷进行修复，保证平台的功能完整性，确保平台可以正常上线运行。

5.1.2　测试环境

本平台在公司进行部署并测试，其中测试环境配置如表 5.1所示。系统测试环境中使用5台服务器，分别部署一个Spring Cloud的Eureka注册中心，一个众包在线验证模块以及三个设备微服务模块。服务器统一使用Ubuntu16.04系统并配置Java环境，在部署众包在线验证模块的服务器上部署MySQL数据库，配置Nginx反向代理服务，启动Nginx实例。平台共使用20台主流Android手机与设备微服务模块所在服务器进行USB连接，并打开USB调试模式。

表 5.1: 系统测试环境

设备与软件	备注
服务器	5台，每台4G内存10M带宽
系统环境	Ubuntu16.04，jdk1.8.10_131
数据库	MySQL5.7.18
部署软件	Nginx
服务注册中心	Spring Cloud的Eureka注册中心
设备	华为、小米、VIVO等20台主流Android手机

5.1.3　单元测试

为了保证代码健壮性，需要通过单元测试来验证系统中最小可测单元的正确性。本平台使用Junit 4作为单元测试框架，并通过Jenkins进行项目构建部署

得出单元测试结果。Junit 4引用了许多注解来替代以往命名约定方式，图 5.1是项目中使用Junit 4编写的单元测试代码示例，其中@Mock引入需要被Mock的对象，@InjectMocks创建一个实例，其余用@Mock注解创建的对象被注入到该实例中。代码中使用@Test标注是测试类的测试方法，测试逻辑在该方法中实现。在测试方法之前执行的步骤逻辑则通过注解@Before标注，用来初始化前置条件，例如初始化模拟HTTP请求的MockMvc对象。@After注解标注的方法则是在测试方法之后执行，一般用来释放资源或回滚测试中修改的数据。

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class DeviceControllerTest {
    private MockMvc mockMvc;
    private Device device;
    @InjectMocks()
    private DeviceController deviceController = new DeviceController();
    @Mock
    private DeviceService deviceService;

    @Before
    public void setupMockMvc() {
        initMocks( testClass: this);
        mockMvc = MockMvcBuilders.standaloneSetup(deviceController).build();
        device = new Device();
        device.setSerialNumber("c123a567");
        device.setBrand("HUAWEI");
    }

    @Test
    public void should_returnDevice_when_givenSerialNo() throws Exception{
        when(deviceService.getDeviceBySerialNo("c123a567")).thenReturn(device);
        MvcResult result = mockMvc.perform(get( urlTemplate: "/enterprise/device/c123a567"))
                .andDo(print()).andExpect(status().isOk()).andReturn();
        JSONObject jsonObject = new JSONObject(result.getResponse().getContentAsString());
        Assert.assertEquals( expected: "c123a567",jsonObject.get("serialNumber"));
        Assert.assertEquals( expected: "HUAWEI",jsonObject.get("brand"));
    }
}
```

图 5.1: 单元测试代码示例图

Jenkins是一种持续集成工具，在编写好单元测试后，可以通过Jenkins对项目进行构建，之后在构建页面中查看单元测试结果。公司内部已搭建好Jenkins环境，只需要在其中新建Job并添加配置就可以实现平台各模块的一键部署。平台分为众包在线验证模块和设备微服务模块，它们的逻辑代码分别在不同项目中编写实现，因此在Jenkins新建两个Job来实现各模块的构建和部署。在每个Job的配置中除了设置构建需要的条件外，选择构建后操作步骤中的Publish Junit test result report，配置报告生成路径，这样就可以在Job的每次构建页面中看到展示Junit测试结果的XML测试报告。

设备微服务模块的单元测试最终结果如图 5.2所示，对模块的controller、common、service、util和wrapper包中主要最小可测单元进行测试。报告显示总共运行130个单元测试，并且每个包中的所有单元测试都全部通过。

Test Result

0次失败 (±0)

130个测试 (±0)

花了

添加说明

所有的测试

Package	花的时间	失败 (区别)	跳过 (区别)	Pass (区别)	总数 (区别)
cn.acovp.deviceservice.controller	2.37 秒	0	0	25	25
cn.acovp.deviceservice.common	1.88 秒	0	0	20	20
cn.acovp.deviceservice.service	4.53 秒	0	0	44	44
cn.acovp.deviceservice.util	2.32 秒	0	0	27	27
cn.acovp.deviceservice.wrapper	1.39 秒	0	0	14	14

图 5.2: 设备微服务模块单元测试报告

众包在线验证模块的单元测试最终结果如图5.3所示，主要对其中controller、common、service和util包中的单元类和方法进行测试。总共运行116个单元测试，每个包中的单元测试通过率都为100%。

Test Result

0次失败 (±0)

116个测试 (±0)

花了

添加说明

所有的测试

Package	花的时间	失败 (区别)	跳过 (区别)	Pass (区别)	总数 (区别)
cn.acovp.cov.controller	3.04 秒	0	0	32	32
cn.acovp.cov.common	2.09 秒	0	0	22	22
cn.acovp.cov.service	4.48 秒	0	0	45	45
cn.acovp.cov.util	1.54 秒	0	0	17	17

图 5.3: 众包在线验证模块单元测试报告

5.1.4　功能测试

在测试环境上部署平台后，就可以在浏览器上对平台进行功能测试。通过编写测试用例并由测试人员按顺序执行用例中的每个步骤得出测试结果，与预期结果进行比较，若比较后结果一致则表示该测试用例通过，证明该功能点已达到目的，不通过则对其进行标记并修复。对平台的功能测试主要是测试用户交互流程，分为设备管理功能测试、设备操控功能测试、脚本操作功能测试和缺陷验证功能测试。下面分别对这四部分功能测试进行阐述，详细介绍测试用例要测试的功能，测试该功能需要执行的测试步骤，以及对执行后的预期结果和真实测试结果。

设备管理功能测试；设备管理功能主要是从设备连接和设备状态两方面进行测试，根据设备列表界面中的展示内容得到测试结果。如表 5.2所示描述设备管理功能的测试用例，包括测试功能、测试步骤、预期结果和测试结果。经过测试后发现测试结果与预期结果相符合，测试功能全部通过。

表 5.2: 设备管理功能测试用例

用例编号	测试功能	测试步骤	预期结果	测试结果
TC1	设备与服务器连接	1. 将设备连接到服务器USB端口，打开USB调试 2. 点击缺陷验证按钮进入设备列表界面	设备列表界面展示该连接设备的基本信息，包括设备的品牌、型号、系统版本和分辨率	通过
TC2	设备与服务器断开	1. 将已连接设备与USB端口断开 2. 点击缺陷验证按钮进入设备列表界面	设备列表界面不再显示该设备基本信息	通过
TC3	使用设备，设备状态发生变化	1. 用户1点击可使用设备的使用按钮 2. 用户2点击缺陷验证按钮进入设备列表界面 3. 用户1结束设备使用 4. 用户2刷新设备列表界面	用户1在使用设备时用户2的设备列表界面显示该设备处于占用中；用户1在结束设备使用后用户2的刷新后设备列表界面显示该设备处于可用中	通过

设备操控功能测试：如表 5.3所示描述了设备操控功能的所有测试用例。测试目的主要是查看设备在自动下载安装待测应用以及浏览器上用户对设备的操控和设备界面同步展示功能是否全都完整，符合要求。经过测试，可得知测试结果与预期结果相符，测试项全部通过。

表 5.3: 设备操控功能测试用例

用例编号	测试功能	测试步骤	预期结果	测试结果
TC4	设备自动下载安装并启动待测应用	1. 参与缺陷众包项目，点击某一缺陷的缺陷验证按钮 2. 选择设备并点击使用按钮	设备在界面上显示屏幕画面后自动下载安装并启动待测应用，界面显示安装进度条	通过
TC5	操控设备	1. 选择设备并点击使用按钮 2. 在界面上对设备进行点击、滑动、长按、输入操作	设备界面与浏览器界面同步，设备响应用户对浏览器界面上设备的各种操作	通过
TC6	点击设备快捷键	1. 选择设备并点击使用按钮 2. 点击浏览器界面提供的菜单键、HOME键、App Switch键和返回键	点击菜单键时设备弹出应用菜单，点击HOME键设备回到主界面，点击APP Switch键设备显示所有正在运行应用，点击返回键返回到设备上一步页面	通过

脚本操作功能测试：脚本操作功能测试主要涉及到脚本录制时测试脚本步骤的生成、修改、移动、插入和删除功能，以及测试脚本的生成展示和测试脚本的回放，还包括录制期间logcat日志的展示和待测应用的重启操作。表 5.4 描述脚本操作功能的测试用例，并通过测试得出测试结果，所有测试用例结果与预期结果相符，全部通过。

表 5.4: 脚本操作功能测试用例

用例编号	测试功能	测试步骤	预期结果	测试结果
TC7	测试脚本步骤生成	1. 选择设备使用 2. 点击开始录制按钮 3. 对设备进行模拟操作	每次设备操作会被转为测试脚本步骤并展示在界面上，包括步骤的触发动作以及控件属性	通过
TC8	测试脚本步骤修改	1. 点击任一脚本步骤的编辑按钮 2. 修改脚本信息	被修改的脚本步骤展示信变成修改后信息	通过
TC9	测试脚本步骤移动	1. 将任一脚本步骤拖动到其它脚本步骤位置	被拖动的脚本步骤从原处插入到移动处，脚本步骤顺序发生变化	通过
TC10	测试脚本步骤插入	1. 点击任一脚本步骤的后一步插入步骤按钮 2. 对设备进行模拟操作	设备操作后生成的脚本步骤在该步骤的下一步中显示	通过
TC11	测试脚本步骤删除	1. 点击任一脚本步骤的删除按钮	该脚本步骤被删除，不再显示在界面上	通过
TC12	测试脚本生成	1. 点击脚本生成按钮	测试脚本文件被生成，界面弹出Dialog框展示脚本内容并提供下载按钮	通过
TC13	测试脚本回放	1. 点击脚本生成按钮 2. 点击脚本展示框的返回按钮 3. 点击脚本回放按钮	生成的测试脚本被执行，设备回放脚本中的所有步骤	通过
TC14	logcat日志获取	1. 点击开始录制按钮 2. 点击log展示框中的获取按钮 3. 点击停止按钮	点击获取后log展示框中逐条显示获取的日志信息，点击停止后信息不再变化	通过
TC15	待测应用重新启动	1. 点击重启App按钮 2. 选择是否清除App所有数据 3. 点击确定按钮	应用会在设备上重新启动，并会在用户选择清除App所有数据时清除App在设备上的信息	通过

缺陷验证功能测试：缺陷验证功能是本平台最终目的，对该功能的测试也最重要。它的主要测试点包括缺陷验证结果的提交以及缺陷验证结果的统计显示功能，根据用户提交验证结果后的提示以及该结果是否在验证结果列表中显示来证明这些功能点是否通过。表 5.5是缺陷验证功能的测试用例，通过测试可知测试结果与预期结果相同，测试用例全部通过。

表 5.5: 缺陷验证功能测试用例

用例编号	测试功能	测试步骤	预期结果	测试结果
TC16	缺陷验证结果提交	1. 点击任一缺陷的缺陷验证按钮 2. 选择设备使用 3. 对设备和脚本进行操作 4. 选择是、不是和不确定中的一种结果 5. 选择是时分别选择一种BUG分类、复现程度和严重等级 6. 选择是否将测试脚本作为参考 7. 点击确定	缺陷验证结果提交成功，在下次众包工人再次验证该缺陷时可显示上次验证结果	通过
TC17	缺陷验证结果统计展示	1. 任务请求者点击缺陷的验证结果按钮	界面展示所有提交的验证结果并显示结果统计饼图	通过

5.2　系统实验设计

5.2.1　实验目标

在对本平台进行的单元测试和功能测试通过之后，平台功能已全部正确实现。之后平台正式部署在公司的生产环境中，作为公司移动应用测试平台的子平台进行上线，提供众包在线验证服务给用户使用。为了验证本平台的实际可用性，本文设计缺陷验证实验来对平台进行系统运行效果分析。本文在公司提供的安卓设备中选取50台进行实验，在Web界面分别使用它们进行远程操控，通过查看操作是否正常来验证平台提供的设备操控功能。之后，本文分别将7个有缺陷版本的应用进行安卓应用自动化测试，在得出测试报告后将各自报告中的全部缺陷都作为测试目标转为众包测试，设置不同人数项目组发布到项目广场中，有偿地让线上用户帮助验证。最后将内部专家验证结果作为标准，与不同人数项目组得出的验证结果进行比较得出结论，验证系统运行效果。在实验中，我们希望能够验证以下两个问题。

问题一：本文的平台对安卓设备的远程操控支持度如何？

问题二；本文的平台对缺陷的验证效果如何？

5.2.2　实验过程与结果

首先为了验证问题一，本文选取华为、小米、三星等50台安卓设备通过USB与服务器连接，其中某些主流机型设备有多台，并特意从公司设备库中挑选了Android系统中最高9.0版本以及最低4.4版本的设备，这些设备的具体信

息如表 5.6所示。之后在某个缺陷众包验证项目中分别使用这些设备进行远程操控，通过Web界面进行观察和操作来验证这些设备是否能够正常操控。

表 5.6: 实验安卓设备列表

品牌	安卓系统版本	数量（台）
华为	Android 9.0	2
华为	Android 8.0	4
华为	Android 7.0	6
小米	Android 9.0	3
小米	Android 6.0	4
小米	Android 5.1	5
三星	Android 7.0	3
三星	Android 6.0	5
魅族	Android 9.0	4
魅族	Android 8.0	3
魅族	Android 6.0	2
VIVO	Android 6.0	4
OPPO	Android 5.1	3
ZTE	Android 4.4	2

在对这些设备进行操控后，根据操作结果得知这些设备都能操控正常，部分设备远程操控界面图如图 5.4所示。经实验可知本平台对于各种主流安卓设备都能较好地支持，并且对于Android 4.4及以上系统的设备都能很好地进行远程控制，而对于Android 4.4以下版本设备由于公司不会再提供也不做具体实验分析。同时，对于目前新型设备（Android 9.0版本的华为、小米与魅族手机）本平台也能进行操控，这也验证平台对安卓设备的远程操控支持度很高。

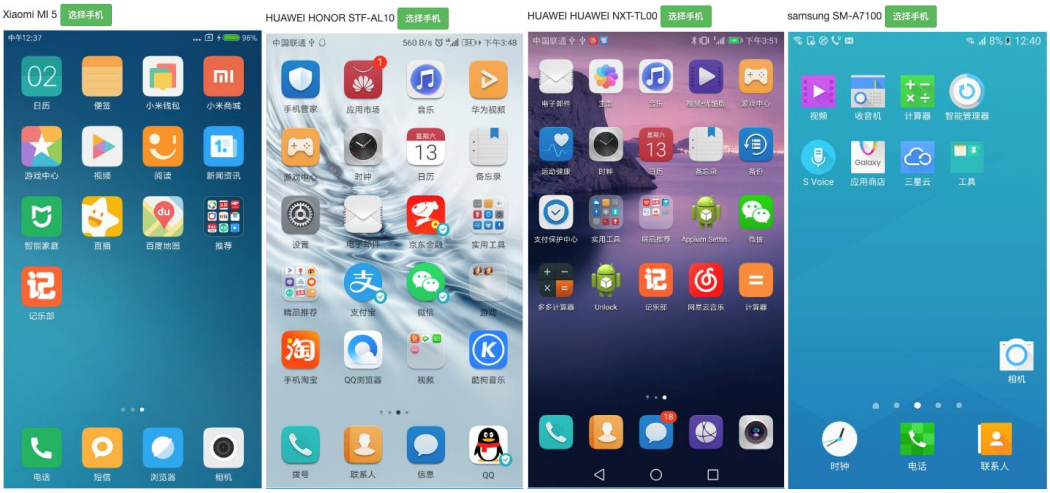

图 5.4: 部分设备远程操控界面图

为了验证问题二，本文选择7个有缺陷版本的安卓应用，将它们分别在公司移动应用测试平台中进行安卓应用自动化测试。这些应用类型各不相同，表5.7描述了它们的编号、名称、类型以及版本号。在测试完成后，将得到的自动化测试报告中所有缺陷都作为测试目标转为众包测试，分别设置10人项目组、20人项目组以及50人项目组发布到任务广场中，有偿地让平台上有一定技术基础的用户加入项目组并在安卓众包在线验证平台上进行缺陷验证。

表 5.7: 待测安卓应用信息表

编号	名称	类型	版本
A1	途牛旅游	旅游	9.51.0
A2	落网	音乐	5.3.6
A3	中华万年历	天气日历	6.8.5
A4	咕咚翻译	工具	1.8.0
A5	GnuCash	理财	2.3.0
A6	哔哩哔哩	娱乐	5.3.1
A7	起点读书	电子书	6.5.3

本文首先统计并保存每个应用的自动化测试报告中缺陷数量，然后让公司内部专家对每个测试报告中的所有缺陷进行验证，投票表决出缺陷验证结果，最终得出测试报告中的真正缺陷，并将内部专家们验证出的真正缺陷数作为实验衡量标准。之后在项目时间结束后对发布到任务广场的每个应用的10人组、20人组和50人组缺陷众包验证项目结果进行统计，实验中保证每个项目组都是满员状态，否则重新发布众包测试任务。在结果统计中分别查看每个项目组中的所有提交结果，通过这些验证结果获取验证出的真正缺陷数量。由于每个众包工人的验证结果可能不一样，这里按照公司的判断标准作为参考，即若验证某缺陷是真正缺陷的人数在项目组总人数占比中超过65%时，则将该缺陷视为真正缺陷。然后根据这个参考标准对提交结果进行统计，获取并记录每个应用的每个项目组通过安卓众包在线验证平台验证得出的真正缺陷数量。

各应用实验结果如表 5.8所示，展示每个应用的自动化测试报告中的缺陷数量，标准缺陷数量（内部专家验证得出的真正缺陷数量）以及这些应用在10人组、20人组和50人组的众包测试项目中验证得出的缺陷数量和这些缺陷中属于标准缺陷的数量。最后数据表给出了缺陷验证准确率，该准确率为通过安卓众包在线验证平台验证得出的真正缺陷中属于标准缺陷数与标准缺陷数量的比值。由表可知，受到测试工具等影响，测试报告中确实存在不准确缺陷。在通过创建众包测试并在本平台验证缺陷之后，虽然验证结果准确度存在误差（比如应用A2的10人项目组中验证得出的缺陷数与标准缺陷数相同，但其中有两个

不属于标准缺陷），但是可看出随着验证的众包工人数增加，误差在逐渐减少，并且验证得出的缺陷准确率也越来越高，平均保持在80%左右。

表 5.8: 各应用缺陷验证实验结果

应用编号	测试缺陷数	标准缺陷数	验证缺陷数(属于标准缺陷数)			缺陷验证准确率		
			10人组	20人组	50人组	10人组	20人组	50人组
A1	8	6	4(4)	5(5)	5(5)	67%	80%	80%
A2	11	8	8(6)	7(6)	8(8)	75%	75%	100%
A3	6	5	4(4)	4(4)	4(4)	80%	80%	80%
A4	7	6	6(4)	6(5)	6(5)	67%	83%	83%
A5	14	10	11(8)	10(8)	10(9)	80%	80%	90%
A6	5	4	4(3)	4(3)	3(3)	75%	75%	75%
A7	10	9	10(7)	9(8)	10(9)	78%	89%	100%

本文通过具体实验对本平台运行效果进行分析，验证了平台对安卓设备远程操控的高支持度以及在缺陷验证方面的可用性。在将缺陷作为测试目标转为众包测试后，当参与的众包工人有一定技术基础且数量足够时，可通过本平台得出高准确度的缺陷结果，帮助用户发现真正的缺陷信息。

5.3　本章小结

本章对安卓众包在线验证平台进行测试以及实验设计。对测试目标、测试环境进行详细描述，并对平台进行单元测试以及功能测试，展示测试结果。之后对平台进行实验设计，通过实验对平台的运行效果进行分析，验证了平台的可用性。

第六章　总结和展望

6.1　总结

随着移动互联网的快速发展，安卓应用也在爆发性增长，为了保证安卓应用的质量，安卓应用测试必不可少。当前主流测试平台针对安卓应用都提供了自动化测试功能，执行用户上传的测试脚本并最终生成包含应用缺陷信息的测试报告。但限于测试工具的测试准确性以及测试脚本质量问题，这些平台测试得到的应用缺陷并不能确保百分百准确。此时，一些用户想要验证这些缺陷时会有设备或人力资源不足的问题。因此，为了解决这一问题，本文设计开发了安卓众包在线验证平台，帮助用户快速验证缺陷，完善缺陷信息。

本文首先介绍平台的项目背景以及众包验证平台和微服务的研究现状，然后介绍平台使用到的关键技术和工具，包括管理与维护整个平台的微服务框架Spring Cloud，为了支持实时浏览器与设备的双向通讯而使用的WebSocket协议和Netty框架，保证设备实时操控的设备交互工具MiniCap和MiniTouch，执行测试脚本的移动应用自动化测试框架Appium以及Web开发框架Angular2。之后本文对平台进行需求分析和设计实现，重点描述平台中各个子模块的设计与实现细节，并展示平台在公司上线后的实际运行界面图。最后，本文对平台进行单元测试以及对平台主要的设备管理、设备操控、脚本操作和缺陷验证功能进行测试，并对平台的运行效果进行实验设计与分析，验证了平台的可用性。

本平台在公司已稳定运行并开放给用户使用。平台让用户在设备与人力资源缺乏的情况下只需通过创建众包测试项目，由众包工人通过平台提供的在线真机进行缺陷验证，就能快速根据他们提交的验证结果获取真正和充分的缺陷信息，提高报告中缺陷准确率，最终帮助用户提高应用质量。平台也因此受到越来越多的用户青睐，为公司业务发展做出了贡献。

6.2　展望

当前平台仍然存在一些不足之处，部分功能也有待改进。

第一，当前平台在设备操控中对安卓设备传输的每张图片都经过压缩处理后展示在界面上，但目前由于服务器带宽的限制，大量用户同时在线操控设备时会略微出现卡顿现象。因此在后续改进中会使用更好的图片压缩方式，同时也会增加公司服务器带宽，保证每一个用户都流畅地操控设备。

第二，本平台目前在测试脚本操作中支持基本的点击、滑动、长按、输入等操作的录制，在下一步工作中平台会提供更多样的测试脚本命令，如if-else条件语句、断言和循环语句等。

第三，当前平台生成的测试脚本只有Java版本，脚本种类可以继续增加，提供更多的开发语言的脚本模板，让任务请求者可参考的脚本更加丰富。

第四，平台虽然目前只支持安卓设备的在线操控，但在设计之初已考虑到iOS设备的操控，并将设备操控相关逻辑统一封装成通用接口提供，使得平台具有可扩展性。在后续过程中平台会提供iOS设备在线操控功能，这使得平台提供的真机设备种类更加多样，让用户能够通过各种类型设备进行缺陷复现来验证缺陷，最终获取更准确的缺陷信息。

参考文献

[1] 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.

[2] K. Mao, L. Capra, M. Harman, Y. Jia, A Survey of the Use of Crowdsourcing in Software Engineering, Journal of Systems and Software 126 (2017) 57–84.

[3] N. Leicht, I. Blohm, J. M. Leimeister, Leveraging the Power of the Crowd for Software Testing, IEEE Software 34 (2) (2017) 62–69.

[4] M. Lease, E. Yilmaz, Crowdsourcing for Information Retrieval: Introduction to the Special Issue, Information Retrieval 16 (2) (2013) 91–100.

[5] 章晓芳, 冯洋, 刘峘, 陈振宇, 徐宝文, 众包软件测试技术研究进展, 软件学报29 (1) (2018) 69–88.

[6] M. Geogy, A. Dharani, Customising Android Automated Testing Framework to Enable Native Hardware and Software Support, International Journal of Engineering Research & Technology 2 (2) (2013) 1–3.

[7] S. Newman, Building Microservices, O'Reilly Media, 2015.

[8] H. Hawilo, M. Jammal, A. Shami, Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms, IEEE Network 33 (2) (2019) 202–210.

[9] G. Pallis, D. Trihinas, A. Tryfonos, M. D. Dikaiakos, DevOps as a Service: Pushing the Boundaries of Microservice Adoption, IEEE Internet Computing 22 (3) (2018) 65–71.

[10] P. D. Francesco, P. Lago, I. Malavolta, Architecting with Microservices: A Systematic Mapping Study, Journal of Systems and Software 150 (2019) 77–97.

[11] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, IEEE Software 33 (3) (2016) 42–52.

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

[13] 翟永超, Spring Cloud微服务实战, 北京：电子工业出版社, 2017.

[14] S. He, L. Zhao, M. Pan, The Design of Inland River Ship Microservice Information System Based on Spring Cloud, in: Proceedings of the 5th International Conference on Information Science and Control Engineering, 2018, pp. 548–551.

[15] 王磊, 微服务架构与实践, 北京：电子工业出版社, 2016.

[16] 周立, Spring Cloud与Docker微服务架构实战(第2版), 北京：电子工业出版社, 2018.

[17] J. Carnell, Spring Microservices in Action, Manning Publications, 2017.

[18] D. Coward, Java WebSocket Programming, McGraw-Hill Osborne Media, 2013.

[19] V. Pimentel, B. G. Nickerson, Communicating and Displaying Real-Time Data with WebSocket, IEEE Internet Computing 16 (4) (2012) 45–53.

[20] E. Bozdag, A. Mesbah, A. V. Deursen, A Comparison of Push and Pull Techniques for AJAX, in: Proceedings of the 9th IEEE International Symposium on Web Systems Evolution, 2007, pp. 15–22.

[21] K. Singh, V. Krishnaswamy, Building Communicating Web Applications Leveraging Endpoints and Cloud Resource Service, in: Proceedings of the 6th IEEE International Conference on Cloud Computing, 2013, pp. 486–493.

[22] M. Heinrich, M. Gaedke, WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata, in: Proceedings of the 11th International Conference on Web Engineering, 2011, pp. 387–390.

[23] K. Shuang, X. Shan, Z. Sheng, C. Zhu, An Efficient ZigBee-WebSocket Based M2M Environmental Monitoring System, in: Proceedings of the 12th IEEE International Conference on Dependable, Autonomic and Secure Computing, 2014, pp. 322–326.

[24] N. Maurer, M. A. Wolfthal, Netty in Action, Manning Publications, 2015.

[25] J. Yang, H. Zhang, L. Han, B. Cui, G. Dong, Design and Implementation of Software Consistency Detection System Based on Netty Framework, in: Proceedings of the 11th International Conference On Broad-Band Wireless Computing, Communication and Applications, 2016, pp. 343–351.

[26] S. Zhang, S. Zhu, Server Structure Based on Netty Framework for Internet-based Laboratory, in: Proceedings of the 10th IEEE International Conference on Control and Automation, 2013, pp. 538–541.

[27] 李林锋, Netty权威指南, 北京：电子工业出版社, 2014.

[28] D. Duan, M. Huang, Y. Mu, Research for Building High Performance Communication Service Based on Netty Protocol in Smart Health, in: Proceedings of the 9th International Conference on Signal Processing Systems, 2017, pp. 18–21.

[29] 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.

[30] L. Gomez, I. Neamtiu, T. Azim, T. D. Millstein, RERAN: Timing- and Touch-sensitive Record and Replay for Android, in: Proceedings of the 35th International Conference on Software Engineering, 2013, pp. 72–81.

[31] V. Garousi, W. Afzal, A. Çaglar, I. B. Isik, B. Baydan, S. Çaylak, A. Z. Boyraz, B. Yolaçan, K. Herkiloglu, Comparing Automated Visual GUI Testing Tools: An Industrial Case Study, in: Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing, 2017, pp. 21–28.

[32] S. Singh, R. Gadgil, A. Chudgor, Automated Testing of Mobile Applications using Scripting Technique: A Study on Appium, International Journal of Computer Engineering and Technology 4 (5) (2014) 3627–3630.

[33] M. Hans, Appium Essentials, Packt Publishing, 2015.

[34] T. Azim, I. Neamtiu, Targeted and Depth-first Exploration for Systematic Testing of Android Apps, in: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, 2013, pp. 641–660.

[35] A. M. Sinaga, A. W. P., A. Silalahi, N. Yolanda, Performance of Automation Testing Tools for Android Applications, in: Proceedings of the 10th International Conference on Information Technology and Electrical Engineering, 2018, pp. 534–539.

[36] N. Verma, Mobile Test Automation with Appium, Packt Publishing, 2017.

[37] D. Zun, T. Qi, L. Chen, Research on Automated Testing Framework for Multiplatform Mobile Applications, in: Proceedings of the 4th International Conference on Cloud Computing and Intelligence Systems, 2016, pp. 82–87.

[38] F. Belli, C. J. Budnik, L. White, Event-based Modelling, Analysis and Testing of User Interactions: Approach and Case Study, Software Testing, Verification and Reliability 16 (1) (2006) 3–32.

[39] M. Ramos, M. T. Valente, R. Terra, AngularJS Performance: A Survey Study, IEEE Software 35 (2) (2018) 72–79.

[40] M. Gechev, Switching to Angular 2, Packt Publishing, 2016.

[41] J. Zhu, Q. Min, J. Wu, G. Y. Tian, Probability of Detection for Eddy Current Pulsed Thermography of Angular Defect Quantification, IEEE Transactions on Industrial Informatics 14 (12) (2018) 5658–5666.

简历与科研成果

基本情况 周顺祥，男，汉族，1996 年 1 月出生，江西省吉安县人。

教育背景

2017.9～2019.6 南京大学软件学院 硕士

20013.9～2017.6 华中科技大学软件学院 本科

参与项目

1. 南京南瑞集团项目：移动众测平台软件（20170413），2017-2018
2. 江苏省博士后科研资助计划：基于持续融合模型的移动应用测试自动生成（2018K028C），2018-2019
3. 国家自然科学基金项目：基于可理解信息融合的人机协同移动应用测试研究（61802171），2019-2021

致　　谢

研究生生活即将结束，在这两年的学习时光里，我得到许多老师与同学们的关怀和帮助。在论文完成之际，我要向所有给予我指导、关心和帮助的人致以最诚挚的感谢。

首先我要感谢我的导师陈振宇老师以及房春荣老师，在研究生阶段的两年时间里，陈老师给我们提供了良好的学习环境，并在科研方面提供很多帮助和支持。在毕设项目的设计与开发过程中，陈老师和房老师从毕设选题到项目的开发实现中都给予我很多灵感，也为我的项目提出了许多宝贵的意见，让我在他们的悉心指导下，最终完成此项目和论文。同时陈老师和房老师严谨踏实的学术作风和勇于创新的科研精神也给我留下了深刻的印象，让我受益匪浅。

其次我要感谢在研究生期间一直帮助和鼓励我的实验室同学们以及我的家人，你们的鼓励让我在读研的路上一直勇往直前，更激励着我在未来道路上再接再厉。

最后，我要向评审论文和论文答辩委员会的各位老师致以最诚挚的敬意，感谢你们在这期间的付出。

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