南京大学
NANJING UNIVERSITY

研 究 生 毕 业 论 文

（申请硕士学位）

论 文 题 目　安卓应用自动化测试中台系统的设计与实现

作 者 姓 名　李灏宇

专 业 名 称　软件工程

研 究 方 向　软件工程

指 导 教 师　陈振宇　教授

2020 年 5 月 23 日

学　　号：MG1732006

论文答辩日期：2020 年 5 月 23 日

指 导 教 师：　　**（签字）**

Design and Implementation of Middle-end System for Android Application Automation Test

By

Haoyu 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 Science

Software Institute

May 2020

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：安卓应用自动化测试中台系统的设计与实现

工学硕士（软件工程领域）专业 2017 级硕士生姓名：李灏宇

指导教师（姓名、职称）：陈振宇 教授

摘 要

随着安卓智能设备的快速普及，“碎片化问题”极大的提高了安卓应用的测试成本。许多企业纷纷使用第三方的自动化测试服务。对于测试服务提供商来说，如何应对不同客户需求的差异，应对客户需求变化的风险，成了亟待解决的问题。与此同时，越来越多的互联网企业开始实施中台战略，以追求更为坚固的基础设施、更为通用的公共服务、更为灵活的项目架构。但是很少有人关注测试中台系统，尤其是对于安卓应用测试这种强依赖于物理设备资源的测试类型。

基于以上背景，结合自动化测试的高效率与测试中台系统通用性和灵活性的思路，本文提出并实现了一个基于 Appium 的安卓自动化测试中台系统。本系统首先会对需要进行测试的 APK 进行分析，之后选取当前空闲的安卓设备，通过 Appium 框架与 ADB 工具，执行自动化的遍历测试。本系统提供了默认的基于 DFS 的自动化遍历算法，通过对界面上所有 UI 控件的操作，以发现一些正常流程之外的异常情况。同时本系统通过 Groovy 支持动态执行自定义 Java 脚本以增加系统的可扩展性。在遍历执行期间，本系统还会收集若干测试中间数据，如设备信息、屏幕截图、设备 Logcat 日志、设备运行时状态等。这些中间数据会作为元数据，供后续的报告生成服务使用。通过指定不同的报告生成服务，系统可以从性能、功能性、稳定性、健壮性和兼容性等角度进行分析，生成多维度测试报告。同时，系统也支持自定义测试报告分析工具且分析工具可以热插拔。

本文选取了市面上常见的 206 个安卓应用，在 12 台安卓设备上进行了集成验收测试。除 19 个应用因自身原因无法正常进行自动化测试外，其余应用均可通过本系统进行自动化测试，测试通过率高达 90.78%。测试结果表明本系统已对不同安卓设备和安卓应用有了较高的支持度。

本系统提供了安卓应用泛化的测试能力。通过提供通用的接口、可配置的测试脚本与分析工具，本系统做到了对上游系统无依赖，且上游系统可以任意切换。目前，本系统已落地于慕测科技云平台，为教育版提供着 APK 评分服务，同时为企业版提供着稳定可靠的自动化测试服务。

关键词：测试中台，安卓自动化测试，Appium，深度优先遍历

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Design and Implementation of Middle-end System for Android Application Automation Test

SPECIALIZATION: Software Engineering

POSTGRADUATE: Haoyu Li

MENTOR: Professor Zhenyu Chen

Abstract

With the rapid popularization of Android smart devices, the "fragmentation problem" has greatly increased the cost of testing Android applications. Many companies have used third-party automated testing services. For test service providers, how to deal with the differences in the needs of different customers and the risk of changing customer needs has become an urgent problem to be solved. At the same time, more and more Internet companies have begun to implement the Middle-end strategy to pursue a more solid infrastructure, more general public services, and more flexible project architecture. However, few people focus on the construction of the testing Middle-end system, especially for Android application testing, which is a type of test that strongly depends on physical device resources.

Based on the background, combined with the high efficiency of automated testing and the idea of universality and flexibility of testing Middle-end system, this paper proposes and implements an Appium-based Middle-end system for Android application automation test. This system first analyzes the APK file that need to be tested, and then selects the currently idle Android devices and performs an automated traversal test through the Appium framework and ADB tools. This system provides a default automatic traversal algorithm based on depth-first search. Through the operation of all UI controls on the interface, it can find some abnormal situations outside the normal process. At the same time, the system supports dynamic execution of custom Java scripts through Groovy to increase system scalability. During the traversal execution, the system also collects several test intermediate data, such as device information, screenshots,

device Logcat logs, device runtime status, execution information, and so on. This intermediate data is used as metadata for subsequent report generation services. By specifying different report generation services, the system can analyze the intermediate data from the perspective of performance, functionality, stability, robustness, and compatibility to generate multi-dimensional test reports. Also, the system supports custom test report analysis tools and the tools can be hot plugged.

This paper selects 206 Android applications that are common in the market, and performs an integration acceptance tests on 12 Android devices. Except for 19 applications, which cannot be used for normal reasons, other applications can be automated tested through this system, and the test pass rate is 90.78%. The result shows that this system has high support for different Android devices and Android applications.

This system provides generalized testing capabilities for Android applications. By providing a universal interface, configurable test scripts, and analysis tools, the system has no dependencies on the upstream system, and the upstream system can be switched arbitrarily. At present, the system has landed on the Mooctest cloud platform, providing APK grading services for the education version, and stable and reliable automated testing services for the enterprise version.

Keywords: Testing Middle-end system, Android automated testing, Appium, Depth-first search

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景及意义

随着科学技术的进步和移动互联网的兴起及成熟，智能手机在我们的生活中扮演着越来越重要的角色。根据 App Annie [1]、newzoo [2]等多家市场分析公司的报告，2019 年全球共有 32 亿智能手机用户，38 亿部活跃智能手机，用户每日使用智能手机的时长达到 3.7 个小时，全球 App 下载量高达 2040 亿次。其中，安卓手机以 87% [1] 的市场份额占据着移动市场的绝对优势。同时，各大安卓手机厂商为适应不同用户的需求，也纷纷推出了不同型号的安卓设备，仅 2019 年，国内新上市的智能手机数量就为 424 款 [2]。再加上不同厂商均有自己的深度定制化操作系统，安卓系统的碎片化问题日益严重 [3] [4]。

与此同时，随着移动互联网市场的成熟，用户也已经不仅仅满足于应用的功能，他们更多在意的是应用的质量以及使用体验。这就需要开发团队在应用上线前，对应用进行更加完备的测试。但是由于安卓系统的碎片化问题与安卓系统版本的快速迭代，安卓应用在上线前就需要耗费非常多的测试成本，以尽可能的适配市面上的主流机型与系统。

目前，很多企业选择向第三方的测试服务提供商寻求帮助，以降低因安卓碎片化问题，而骤升的测试成本。但是，对于测试服务提供商来说，上游企业的测试需求各不相同。以慕测平台为例，企业用户的需求是快速的对待测 APK 进行自动化测试，并得到一份详尽的 Bug 报告；教育用户的需求是，对学生的课程作业 APK 进行评价与打分，以辅助自己的教学安排。如何使用统一的解决方案解决上游企业需求的差异问题，如何适应上游需求快速变化的情况，如何同时支撑多种复杂场景下对安卓应用自动化测试的不同需求、降低开发成本，这些都成了亟待解决的问题。

随着中台战略逐渐走进所有开发者的视野，上述问题便有了一个较好的解决方案。区别于传统的前台为用户提供实时响应的功能，后台为职能人员提供配置管理的功能，所谓中台，其核心在于提供较为通用的基础设施、基础服务或公共资源，避免在项目架构中重复造轮子情况的出现。中台目前可以简单的分为四类：业务中台、技术中台、数据中台和算法中台。但是，目前少有公司关注于测试中台系统的搭建与使用。所谓测试中台，比较偏向于技术中台的概念，

[1] https://www.appannie.com/cn/

[2] https://newzoo.com/

其目的是为不同的团队提供通用的测试服务。但是，目前在项目测试阶段，大部分团队还都是闭门造车，所生产的工具还都仅仅针对部门内部的业务场景，这就造成了大量资源的浪费。尤其是对于安卓应用测试这种强依赖于物理设备资源的测试类型，搭建一个通用的测试平台就显得尤为重要。

在此背景下，依靠于慕测科技的丰富高校资源与企业资源，结合当前中台战略的蓬勃发展，本文设计并实现了一套安卓应用自动化测试中台系统。本系统通过提供抽象良好的接口、可配置的测试脚本与分析工具，实现了移动应用泛化的测试能力，实现了对上游服务无依赖，上游服务可以随意切换的特性。所以本系统可适应各类需求与业务场景，为企业、高校或任意第三方提供稳定可靠的安卓应用自动化测试服务。在本系统的基础上，任何人都可以可以快速搭建符合自身业务场景的安卓应用自动化测试服务与系统。对于中小企业、高校、科研团队或者任意第三方团队来说，只需进行基础的功能开发，便可以快速接入本系统，搭建安卓自动化测试平台进行各自的测试与数据分析服务。目前，本系统已经落地于慕测科技企业版与教育版云平台，提供着稳定可靠的安卓自动化测试服务与 APK 评分服务。

1.2　国内外研究现状

1.2.1　安卓应用自动化测试的现状

移动应用自动化测试一直都是学术界比较热门的研究话题之一。特别是对于安卓应用来说，由于其操作系统的开源性以及较高的市场占用份额，使得研究人员对于安卓应用自动化测试技术进行了大量的研究，产出了许多自动化测试工具。

这些测试工具中，其中一大类是针对应用本身进行自动化遍历测试的工具。这类工具共有两类常见的遍历策略：随机策略与基于模型的测试 [5]。随机策略以 Monkey 工具为代表，十分适合用于进行压力测试。Monkey 是安卓开发者套件中提供的原生测试工具 [6]，其测试策略是将安卓应用视为一个黑盒，然后随机生成用户指定数量的系统级指令，对应用进行操作并监控应用状态，直到应用异常退出或者所有指令执行完毕。其优点是不需要进行其他安装工作，其缺点也显而易见，就是纯黑盒与纯随机的指令并无什么实际意义，且测试的停止依赖于超时策略。A. Machiry 等人提出了一种基于伪随机策略的测试工具 Dynodroid [7]，该工具可以根据事件执行频率或者事件与上下文之间的关系生成操作事件。

基于模型策略的核心思想大都是基于有限状态机，它将页面作为状态，将事件作为过渡，通过动态的构建有限状态机来进行自动化遍历测试。D. Amalfi-

tano 等人提出了一种基于当前应用状态与 DFS 算法的自动化遍历工具 GUIRipper [8]。该工具可以获取当前应用状态并动态的构建模型，且可以通过人工干预，从任意页面开始自动化测试。T. Azim 等人提出了一种基于更抽象模型的测试工具 A3E-Depth-First [9]，它忽略了每个页面内部组件的状态，仅将每个页面进行抽象，但是这样做可能会造成在遍历过程中错过一些非常容易实现的操作。W. Yang 等人提出的工具 ORBIT 在 GUIRipper 工具思路的基础上，加入了对应用源码的静态分析 [10]，期望通过静态分析的结果，发现特定 UI 事件与特定页面之间的关联，以对自动化遍历测试进行辅助。Yu 等人提出了一种基于图像识别与脚本生成技术，辅助安卓应用进行自动化测试的工具 LIRAT [11]。上述工具对于环境都较为敏感，无法做到开箱即用。如 Dynodroid 仅适用于安卓 2.3 版本，GUIRipper 仅能在 Windows 环境下执行且安装十分困难，ORBIT 并没有开源等。同时由于安卓碎片化的问题以及安卓版本的快速迭代，这些工具很难适应当前环境下对于安卓自动化测试的需求。

另一大类测试工具，他们则主要关注于移动应用众包测试流程的优化 [12]。R. Hao 等人提出了一种基于图像识别的测试工具 CTRAS [13] [14]。该工具通对报告中截图的识别与分析，可以识别出重复报告，将这些重复报告汇总成一份更全面且更利于理解的报告。Li 等人提出了一种基于报告推荐，进行测试引导的协同测试工具 CoCoTest [15]。该工具基于实时的报告去重与报告推荐，可以对众包测试进行引导，降低后续报告分析和汇总成本。X. Chen 等人提出了一种测试报告增强工具 TRAF [16]。该工具使用自然语言处理，通过三种不同的策略，分别对报告的上下文、输入以及描述进行增强。

1.2.2　中台技术的现状

中台战略为当前企业技术架构转的非常重要的方向之一。所谓中台，就是将应用在技术平台的支撑下进行封装或者重构，从而形成面向业务场景的共享服务以支撑前端业务快速创新的平台。中台的出现，是为了通过可复用的应用架构设计，来支撑易变复杂的数字化业务。当前，不同公司对于中台架构的发展方向有所不同，总体来说可以分为四类：业务中台、技术中台、数据中台和算法中台。（1）业务中台，即它的划分服从于业务需求，将不同项目的共通业务下沉为通用的服务平台，比如用户中心、订单中心、广告中心等；（2）技术中台，它的目的是为了避免重复造轮子，向不同团队提供通用的底层框架、引擎等；（3）数据中台，它的目的是为不同团队提供数据采集与分析功能，比如数据治理、日志分析、用户画像等；（4）第四类为算法中台，它的目的是为了向不同团队提供公共的算法服务，比如语音识别服务、图像识别服务、推荐搜索算法等。

目前，各大互联网公司也都纷纷进行了自己的中台改造。阿里巴巴通过对其业务特点的分析及划分，形成了用户中心、会员中心、订单中心等业务中台；通过对其通用技术能力的沉淀，形成了其特有的中台架构理念以及方法论 [17] [18]。网易通过对其内部微服务架构的重构与变革，成功将大数据、AI、容器等通用技术形成了技术中台于算法中台，提供给全集团的研发团队使用，避免了重复造轮子而造成了资源浪费 [19]。

同时，中台解决方案也是当前 toB 类企业级应用较为成熟的解决方案之一。如 VisionMind [3]智能视频中台系统，通过其对外开放的视频 AI 的能力，成功落地于城市交通、城市治理、岗位管理、城市客流管理与互联网内容安全等多个领域。中台还在改变着医疗卫生领域。通过将中台系统应用于医院后勤管理中，可以将医院内繁复庞杂各自为政的系统整合起来，形成高效统一的数据流，形成标准规范的业务流程，从而提升医院的工作效率。

1.3　本文主要工作

本文结合当前安卓应用自动化测试的特性，分析了安卓应用自动化测试的难点与痛点，同时为了适应慕测科技企业版与教育版并存、公有云服务与私有云服务并存的复杂业务场景，结合当前中台解决方案的蓬勃发展，利用其通用、灵活、可复用的特点，提出并实现了一个基于 Appium 的安卓应用自动化测试中台系统。目前本系统已落地于慕测科技企业版与教育版云平台，为教育用户提供着高效准确的 APK 评分服务，同时为企业客户提供着稳定可靠的安卓应用自动化测试服务。

本文首先分析了慕测科技云平台企业版的业务场景与需求，设计并实现了一套通用的 HTTP 接口，提供了可配置的脚本执行与分析服务，真正实现了对上游系统无依赖，且上游系统可以任意切换的特性。同时，本系统对外提供了多种不同的接入方式，以适应不同的物理场景与需求。考虑到系统的可扩展性与可维护，本系统使用 Spring Boot 作为业务开发框架。其次，本文通过从不同角度对系统进行需求分析与拆解，将系统划分为多个功能模块，之后重点讨论了各个功能模块的详细设计与实现。同时，在设计时，本文尽可能的将模块做的通用且可替换，提供了较高的兼容性与可扩展性，为今后的迭代打下了坚实的基础。最后，本文对系统各方面的测试工作进行了简单的介绍与总结，通过多方面的测试，可以有效的保障系统各个部分的可用性、稳定性等非功能需求，为上游服务提供可靠稳定的安卓应用自动化测试服务。

[3]https://supremind.com/

1.4　本文组织结构

本文共分为六个章节，组织结构如下。

第一章为引言，介绍了目前国内外移动互联网的相关状况，安卓生态的相关情况与安卓应用测试的重要性以及痛点和难点。针对上述情况，结合当前各大公司的中台战略，本文提出并设计开发了一套安卓应用自动化测试的测试中台系统。

第二章为相关技术概念介绍。介绍了本系统所使用到的各类相关工具与技术，包括开发框架 Spring Boot，测试框架 Appium，存储技术 Redis，容器技术 Docker、脚本语言 Groovy 等。

第三章为本系统的需求分析与概要设计。本文首先对系统进行了整体的概述，简述了其功能性需求与非功能需求，之后用图表的形式，从用例、系统架构、逻辑视图、开发视图、进程视图与物理视图多个角度，对系统进行了拆解与分析。最后介绍了系统所使用到的关键实体对象。

第四章在第三章需求分析的基础上，介绍了系统的详细设计与实现部分。主要包括了本系统各个模块的详细设计、进程图、类图与关键代码或伪代码实现。

第五章为测试部分，介绍了本系统所进行的各项测试工作，并展示了部分系统运行截图。

第六章为总结与展望。本章节简单总结了本文与本系统的各项工作与贡献，同时针对系统的不足之处进行了分析，提出了今后的改进方向。

第二章　相关技术概述

2.1　Appium 框架

Appium 是一个隶属于 OpenJS 基金会的开源自动化测试框架，它主要针对移动平台上三种类型的应用进行测试：安卓设备、iOS 设备和 Windows 桌面上的，原生、移动 Web 和混合的应用 [20]。原生应用是指通过安卓、iOS 或者 Windows SDK 开发的应用程序，可以理解为我们移动设备上或者 PC 上的应用程序。移动 Web 应用是指需要通过手机浏览器才可以访问的应用。混合应用是指通过 Webview 组件访问 Web 内容的应用。使用 Appium 就可以对全平台进行自动化测试工作，这使 Appium 已经成为时下最流行自动化测试框架，同时它还有着非常多的贡献者以及非常活跃的社区，Appium 已经成为了自动化测试首选的开源框架。

Appium 框架在设计上旨在满足移动端自动化测试特定的需求。它的设计理念主要包括如下四点：（1）你不应该为了执行自动化测试而对应用进行任何修改或重新编译；（2）你不应该被特定的编程语言或者框架所局限；（3）移动应用自动化测试框架不应该在 API 上重复造轮子；（4）移动应用自动化测试框架应该在名义、精神和实践上都是开源的。为了践行这些设计理念，Appium 主要有以下的特点。

首先，Appium 使用不同平台内置的自动化测试框架来作为自己的底层实现 [21]，如 iOS 平台的 XCUITest 和 UiAutomation，安卓平台的 UiAutomator 和 UiAutomator2。这么做的目的是不将任何 Appium 或者第三方的代码编译进待测应用，这样就可以保证待测应用在测试期间和发布后是一致的 [22]。其次，Appium 在不同平台的测试框架上进行了统一的封装，将它们包裹进了同一套 API——WebDriver API 当中，然后使用 C/S 架构来实现与 Appium Server 的通信。通过 C/S 架构，我们可以使用任何编程语言或者编程框架编写客户端，向 Appium 服务端发送对应的 HTTP 请求，以实现特定的操作和功能。Appium 服务端是使用 Node.js 进行编写的，它可以很方便的通过 NPM 工具安装到不同的操作系统当中。对于 Appium 客户端，Appium 官方也已经提供了大部分流行编程语言的实现[1]。如此一来，Appium 就可以做到不局限于特定的语言和框架，开发者可以选择自己熟悉和擅长的编程语言进行开发工作。最后，Appium 目前已经

[1]http://appium.io/downloads

完全的免费且开源[2]。同时 WebDriver 已经成为了 Web 自动化测试事实上的标准，基于 WebDriver 的 Appium 框架，也已经成为了移动端自动化测试的标准框架。同时，Appium 团队还提供了丰富的工具来帮助开发人员进行开发，比如 Appium Desktop 提供了可视化的界面辅助测试，Appium Doctor 用于检测 Appium 相关配置环境是否有遗漏。

2.2 Spring Boot

Spring Boot 是 Spring 框架大家族中十分重要的一员，由 Pivotal 团队开发并维护，它是快速构建所有基于 Spring 框架应用的起点。Spring Boot 旨在通过最少的前期配置，使开发者尽快的搭建和启动 Spring 应用 [23]。Spring Boot 框架有如下特点：

（1）Spring Boot 提供自动化配置功能 [24]，使用 Spring Boot 官网[3]或者 IDE 中插件提供的引导页面，可以在几秒钟内构建一个可运行的 Spring 项目，而不需要进行复杂的配置工作。

（2）Spring Boot 提供了一系列 starter 依赖包，将不同版本的 Spring 及其依赖进行整合，使用 starter 包之后，框架就会自动将相关的依赖及其版本处理好，使开发人员无需处理错综复杂的 Maven 依赖关系 [25]，可以将工作重心放到系统功能的开发上。

（3）Spring Boot 内嵌了各种常见的 Servlet 容器，比如 Tomcat，Jetty 等，打包的可执行 jar 包可以直接运行在各种 Java 环境下，所以服务器上除了 Java 环境无需其他额外配置，这样就真正做到了一次编译多地执行，极大减少了项目迁移的成本。

（4）Spring Boot 提供了一系列企业级应用程序中经常用到的公共非功能特性，并且对于这些特性和功能提供了良好的封装和完整的支持，比如 SQL 和 NoSQL、缓存、安全、RESTFul API、监控等。

本系统作为安卓应用自动化测试系统，需要长时间运行在安卓设备机柜中，会进行较为频繁的机柜迁移、机柜扩展等操作，所以对部署的简易程度有一定的要求；另外，设备机柜系统环境较为复杂，不宜进行过于侵入的配置；同时，本系统的上游系统，慕测科技云服务平台，正在进行微服务改造，所以本系统选用 Spring Boot 框架作为开发框架。

[2]https://github.com/appium/appium

[3]https://start.spring.io

2.3　ADB 工具

ADB（Android Debug Bridge，安卓调试桥）工具是 Google 公司为安卓开发者提供的一种具有丰富功能的命令行工具，可以让开发者与安卓设备进行通信 [26]。ADB 命令可以十分方便的执行各种设备操作，比如安装和卸载应用、查找设备等，同时还提供了对 Unix Shell 的访问权限，开发者可以通过 Unix Shell 在安卓设备上执行各种命令 [27]。

ADB 工具的架构与 Appium 框架的架构十分相似，都是 C/S 架构的程序。ADB 工具主要包括三个部分：（1）客户端用于发送命令，它运行在开发机器上，开发者可以在命令行中，通过发送 adb 命令调用客户端。（2）守护进程 adbd 用于在安卓设备上运行命令，它在每台安卓设备上作为后台进程运行 [28]。（3）服务端用于管理客户端和守护进程之间的通信，它在开发机器上作为后台进程运行。ADB 支持服务端通过 USB 与守护进程进行通讯，同时采用 TCP 协议以保证通信的可靠性与稳定性 [29]。

ADB 工具包含在 Android SDK 平台工具软件包中，可以通过 SDK 管理器进行下载安装。ADB 工具作为安卓开发套件中最常用的工具之一，可以帮助系统和安卓设备进行通信，进行设备管理，是安卓应用测试必备的工具之一。

2.4　Redis 数据库

Redis 是一个使用 C 语言编写的，遵守 BSD 开源协议的高性能 key-value 型数据库 [30]。它可以用来作为数据库、缓存和消息中间件使用。Redis 数据库具有以下特点。

第一，Redis 提供了非常丰富的数据结构供开发者使用，包括字符串，列表，数组，哈希，集合，有序集合和 HyperLogLog [31] 等。这是它与其他缓存数据库（如 Memcached）最大的区别。

第二，Redis 中所有数据都存放在内存中的，且 Redis 采用单线程的模型，避免了多线程上下文切换以及维护线程之间同步的开销，所以拥有极高的读写性能，读取速度可以达到 11 万次/秒，写入速度可以达到 8 万次/秒。另外，因为 Reids 是单线程模型，所以 Redis 中所有的操作都具有原子性。

第三，Redis 通过 Master-Slave 主从模式、Redis 哨兵以及 Redis 集群等多种方式，来保证 Redis 实例的高可用性。

最后，Redis 支持多种方式以及不同级别的数据持久化，可以帮助我们将数据从内存持久化到硬盘上，以避免因设备断电等问题导致数据丢失的情况。目前 Redis 支持两种数据持久化方式，RDB 快照和 Append-Only-File（AOF）。这两

种数据持久化方式各有特点。RDB 快照是 Redis 默认的数据持久化方式，正如其名字，这种方式就是将当前内存中的数据以快照的形式存入二进制文件，进行持久化。它使用 fork 命令创建子进程，通过子进程进行内存快照的写入工作。但是这种方式有个缺点，因为快照是存在时间间隔的，所以在两次快照期间的数据，是无法得到持久化保障的。与之对应的是另外一种方式 AOF。当 Redis 数据库每执行一次 Write 操作写入数据时，Redis 就会将这条写命令记录在日志中，然后通过操作回放来恢复数据。与 RDB 相比，AOF 可以保证数据的时效性，但是日志记录会逐渐变大占用磁盘空间。所以目前主流的解决方案是将两种方式结合，AOF 只记录上次 RDB 快照之后的数据，这样就可以尽可能的在控制 AOF 日志大小的同时保障数据不丢失。

因为本系统最终的部署环境为安卓设备机柜，需要始终维护着安卓设备的状态，包括设备是否在线，是否在执行任务，执行任务的进度等等。因为设备可能出现掉线等情况，所以使用传统关系型数据库来存储设备状态的意义并不是很大。同时设备状态的更新是一个非常频繁的操作，且设备执行完任务后，之前的状态就会同步到企业版云服务平台中，本系统无需继续保有之前的状态，故本系统选用 Redis 作为缓存数据库，以保证系统的高效执行和数据存储。

2.5　Docker 与容器技术

在传统软件开发过程中，由于开发人员的开发环境与应用最终运行的生产环境或者测试环境存在天然的差异，所以开发人员与运维人员，开发人员与测试人员会因应用运行环境的区别而产生分歧，影响团队整体的效能。但是，容器技术的出现，改变了这一现状。

容器技术可以简单的理解为一个打包了项目代码及其运行环境的标准化单元，所以在容器中的代码可以完全脱离容器所在物理设备或者云服务器的特定环境，独立运行、实现功能，进而我们的应用可以非常快速便捷的从一个计算环境快速可靠的转移到另一个计算环境，而无需关注新旧计算环境之间的差异。有了容器技术，开发人员就可以专注于业务逻辑，而运维人员就可以专注于代码的打包与部署工作。容器技术是一种内核轻量级的操作系统层虚拟化技术，与之对应的是另外一种较为成熟的虚拟化技术——虚拟机。

虚拟机是将一台计算设备变为多台计算设备的物理硬件抽象 [32]，虚拟化 hypervisor 技术使多台虚拟机可以运行在同一台物理设备中，每一台虚拟机包含独立的操作系统、应用程序、必要库文件的完整副本，这些通常要占用高达几十 GB 的硬盘空间。与容器技术相比，虚拟机技术也是用于资源的隔离与分配，但是除此之外，他们别无其他的相似点。容器技术是一种在操作系统层面的虚拟

化技术，它允许在同一个操作系统中同时运行多个容器，所以它更加的轻量，启动更加快速且基本不占用任何宿主机的资源 [33]。与此同时，容器技术会在操作系统层面上虚拟化 CPU、网络、内存、存储等资源，然后提供沙盒化的操作系统开放接口给开发者使用，因此，容器之间可以更加方便快捷的进行通信与合作，进行各方面的业务配合。当前 Docker 已经成为了容器的代名词，它具有非常良好的开发者生态及社区，同时也已经是各大互联网公司虚拟化改造的首选技术，也是持续集成与持续交付技术的首要选型，故本系统采用 Docker 作为容器技术的选型。

Docker 是一个使用 Go 语言开发的，遵守 Apache2.0 开源协议的容器应用引擎。通过 Docker，我们可以使用 Dockerfile，非常便捷的将我们的代码以及相关运行环境打包到一个标准化镜像之中，发布到镜像仓库上。之后该镜像就可以在任何版本的 Linux 系统中执行，而无需侵入系统本身的环境 [34]。这样就可以真正的做到忽视宿主机本身进行项目的部署，完全保证系统代码的稳定与运行环境的统一 [35]。虽然镜像也可以在 Windows 系统中运行，但是其本质也是在 Windows 系统中运行一个 Linux 虚拟机，然后在虚拟机上运行 Docker。所以为了避免没有必要的性能损失，我们选用 Ubuntu 等 Linux 稳定发行版进行安装与部署。

2.6　Groovy

Groovy 是一种基于 JVM 虚拟机的动态语言 [36]，它拥有类似 Python、Ruby 和 Smalltalk 等动态语言中的一些特性，如动态类型转换、闭包等。同时它也是一种成熟的面向对象的编程语言。对于 Groovy 语言来说，它既可以用于面向对象编程，又可以单纯的作为一种脚本语言使用。

Groovy 语言在设计初始便充分考虑了与 JAVA 的集成，它可以直接动态编译成 JAVA 字节码并运行在 JVM 虚拟机上 [37]。因此 Groovy 语言可以很好的与 JAVA 语言组合在一起进行混合编程。同时它的基本使用方式与 JAVA 相同，也支持大部分的 JAVA 语法。所以 Groovy 可以无缝与其他 JAVA 代码或者库进行交互操作。对于一些需要对 Java 代码进行动态编译的场景，使用 Groovy 进行混合编程对项目进行扩展，是目前非常主流的一种解决方案。

2.7　遍历算法

与传统的 Web 应用不同，对于安卓应用来说，应用中的所有页面都是通过 Activity 这个组件进行关联和展示的，然后通过对按钮的点击或者对特定事件的反应，进行页面之间的跳转，进行 Activity 之间的转换。对于一个应用来说，它

的所有 Activity 与在 Activity 上的操作，可以构成一个有向图结构。图的节点就是每一个 Activity 对象，图的边则表示 Activity 之间的跳转关系。对这个图结构进行覆盖，其实就相当于对这个应用进行遍历。常见的遍历算法有两种：深度优先遍历与广度优先遍历。下面将针对图 2.1中所展示的示例结构，对这两种算法进行说明。

2.7.1　深度优先遍历

所谓深度优先遍历（DFS），就是沿着图的深度遍历其节点，尽可能深的搜索图的分支，同时每一个节点只能被访问一次 [38] [39]。它是对树的前序遍历的推广。

对于图 2.1所展示的图结构来说，首先访问初始节点 A，然后依次从节点 A 的所有没有被访问过的邻接节点出发，再次进行深度优先遍历，也就是对 B 和 C 节点进行深度优先遍历。假设选择节点 B，那么则会再对节点 D、E 进行深度优先遍历。这里假设先访问节点 D，因为节点 D 所有的邻接节点都已经被访问过了（B 已经被访问过了），则进行回溯，回溯到节点 B。之后以此类推，访问节点 E，回溯到 B，A，最后访问节点 C 和 F。至此，所有从初始节点 A 可以达到的节点均已访问过了。

此时，如果存在还未访问的节点，则从一个未被访问的节点出发，再次进行深度优先遍历，直到所有节点都已经被访问到了。因为图 2.1中所展示的图为连通图，不存在还未访问的节点，则跳过此步骤。所以对于图 2.1所示的图结构，其深度优先遍历的结果为：A-B-D-E-C-F。

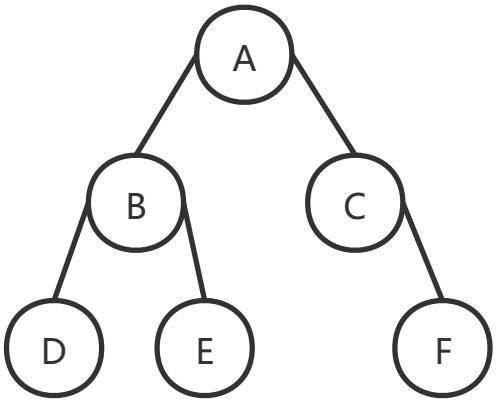

图 2.1: 遍历算法示例图结构

2.7.2　广度优先遍历

所谓广度优先遍历（BFS），就是对树的层次遍历的推广 [38]。它的基本思路是，首先访问源节点 V，并标记其已被访问，然后依次访问 V 的所有邻接节点，V1,V2,...,Vn，并标记他们已被访问。然后再按照 V1,V2,...,Vn 的顺序，依次访问所有他们没有被访问过的邻接节点，然后重复此过程，直到图中所有和源节点 V 存在路径相连的节点都被访问过为止。

对于图 2.1所示的图结构来说，首先访问初始节点 A，之后一次访问 A 的所有邻接节点 B 和 C，然后按照 B、C 的顺序，访问他们未被访问过的邻接节点 D、E、F。至此，所有和初始节点 A 存在路径相连的节点都已被访问过，遍历停止。所以对于图 2.1所示的图结构，其广度优先遍历的结果为：A-B-C-D-E-F。

2.7.3　算法选择

对于安卓应用来说，深度优先遍历的逻辑更符合用户日常使用 App 的操作逻辑，即进入一个新页面后，用户并不会立刻返回上一页面，而是对新页面进行相应的操作。同时，本系统还需要对安卓 XML 布局文件进行解析，XML 这种数据结构也是一种天然适用于深度优先遍历的数据结构。所以本系统选用深度优先遍历算法作为遍历算法的基本模型。

2.8　本章小结

本章主要概述了项目所使用到的算法和相关技术框架，通过对项目特点的介绍与分析，阐述了选择相应框架与技术的理由。首先，对系统核心技术 Appium 进行了概述，主要介绍了 Appium 框架的设计理念、架构以及特点。第二，介绍了项目后端主要的技术选型 Spring Boot 框架。其次，介绍了另外一款常见于安卓自动化测试的 ADB 工具，这个工具可以辅助我们进行更加多样化的操作。之后，阐述了项目所使用的数据存储引擎 Redis 缓存数据库。接着，对于项目部署所用到的关键技术和框架 Docker 进行了简单的介绍与说明。然后简单介绍了本系统所使用的动态语言 Groovy。最后，阐述了常见的遍历算法及其特点，以及选择原因。

第三章　安卓应用自动化测试系统需求分析与概要设计

3.1　系统整体概述

随着移动互联网的发展与智能手机的普及，智能手机和上面功能多样的APP 已经和我们的生活密不可分了。在我们日常生活中，不管是购物、理财、出行、娱乐都已经离不开我们的智能手机了。作为智能手机中占有绝对领先地位的安卓系统，更是和我们的生活息息相关。随着安卓手机厂商的不断入场，以及各大公司对于安卓应用的不断开发与投入，如何保证安卓系统上应用的质量，就成了各大应用角逐的关键所在。但是，由于安卓系统本身是开源的，安卓系统的碎片化问题始终是困扰开发者的难题之一。由于不同手机厂商对于安卓系统的个性化定制，以及安卓系统版本迭代过快，使得市面上必定同时存在使用不同机型以及不同安卓版本的用户，如何保证多样化用户的使用体验，就成为了开发团队所关注的重中之重。

为了保证安卓移动应用上线前的质量，目前，许多企业所采用的方案是使用第三方提供的自动化测试服务，以降低自身测试的成本。同时，随着移动互联网的逐步发展，越来越多的高校都开设了安卓应用开发课程，其课程作业大都是开发一个安卓应用。对于老师而言，很难对学生的课程作业进行评审，传统的人工评审工作量大、效率低，而且会受到主观因素的影响。所以现在高校也都希望采用自动化的方式进行准确、高效且公正的评分。如何应对上述两种场景不断变化且各不相同的需求，如何提供稳定、高效、统一的自动化测试解决方案，成为了我们当前亟待解决的问题。

图 3.1所示的为慕测科技企业版云平台流程图，是本文设计与实现的安卓自动化测试中台系统的落地场景之一，整个系统面向的用户是软件公司的开发人员、测试人员与管理人员。本测试中台系统的另一个落地场景依附于慕测科技教育版云平台，可以为安卓开发课程提供 APK 评分的功能，面向的用户为高校老师与学生。

不论是需要进行测试的 APK，或是需要打分的 APK，都可以通过统一的接口提交到本系统中，然后调用测试服务进行测试。然后我们会在所关联的安卓设备上执行自动化测试。如果用户提供了测试脚本，则会执行测试脚本；如果用户未提供，则会使用默认的测试脚本进行探索性测试，以发现应用正常流程之外的异常情况。在测试过程中，本系统会生成一系列的测试中间数据，包括应用截图，系统 Logcat 日志，测试操作以及应用执行过程中设备的网络、电量、

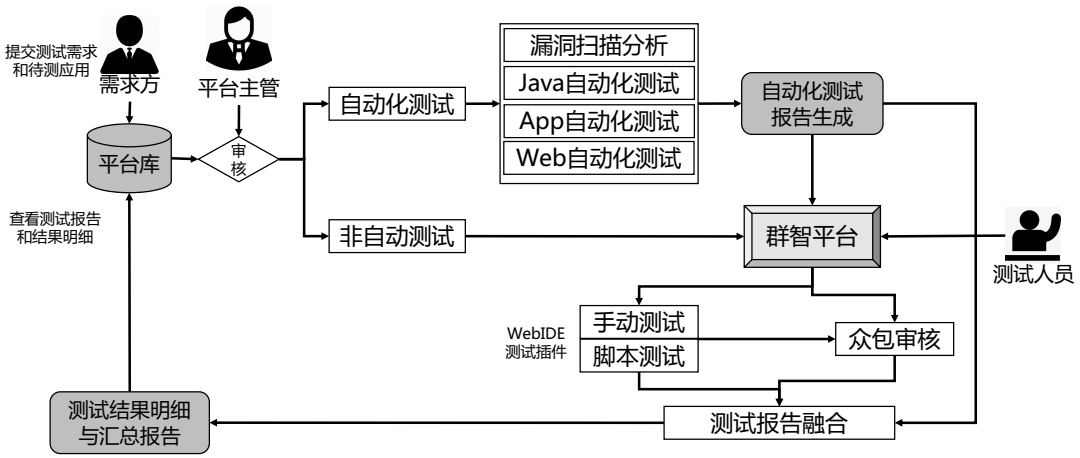

图 3.1: 慕测企业版云平台流程图

内存、CPU、帧率等信息。通过上述的中间数据，系统会调用不同的报告生成服务，从性能、功能性、稳定性、健壮性和兼容性等角度进行分析，生成最终的测试报告供测试需求方参考。用户还可以调用中间数据分析工具，对测试中间数据进行二次分析，如生成众包测试任务以派发到相关的众包测试平台，进行进一步的众包测试和验证。

3.2　系统需求分析

3.2.1　系统涉众分析

本系统作为慕测科技安卓自动化测试中台系统，所涉及的涉众如表 3.1所示，只包含测试需求方这一单一涉众。对于测试需求方来说，他们可以通过本系统发布测试任务，查看任务的执行情况，停止正在进行的任务，查看任务测试报告，对任务中间结果进行分析等。他们或是具有一定的开发能力，同时有能力根据测试报告中 Bug 的相关描述信息，对报告中潜在的 Bug 进行判断和验证、对真实 Bug 进行定位和修复的能力；或是拥有一定的决策能力，能根据报告结果对被测试应用进行评价与决策。他们所期待的是，自动化测试系统能对他们提测的应用在尽可能丰富的设备与机型上进行遍历测试，同时根据测试的中间结果，生成可读性较高，可参考性较高的测试报告。

3.2.2　功能性需求分析

根据系统的涉众分析以及期望分析，可进一度针对系统进行功能性需求分析，本系统主要需求如表 3.2中所示。

表 3.1: 系统涉众分析结果

涉众名称	涉众特征与期望
测试需求方	作为测试的需求方，他们期望在提交待测 APK 之后，本平台可以在尽可能多的安卓机型上进行自动化遍历测试，并提供应用的测试报告。测试需求方具有一定的应用开发能力，具有根据应用的测试报告，判断或验证报告 Bug 是否真实的能力，具有对真实 Bug 进行定位和修复的能力。具有根据测试报告进行评分与决策的能力。

表 3.2: 系统功能需求列表

需求 ID	需求名称	需求描述
R1	启动测试	待测应用的提测方可以在填写一部分基础信息后，启动对应用的自动化测试。
R2	停止测试	待测应用的提测方可以随时终止对应用的自动化测试，在终止后，可以获取根据目前的测试结果生成的各项报告。
R3	查询测试状态	待测应用的提测方可以随时查看自动化测试任务的状态，以及执行该任务的各台安卓设备的执行状态。任务状态包括：初始化、执行中、报告生成中、完成、下载失败、APK 解析失败等。设备执行状态包括：初始化、准备中、执行中、结束中、完成。
R4	查询测试结果	在测试完成后，提测方可以查看测试任务的测试报告，报告包括：测试任务的基本信息、每台设备的执行信息、Bug 列表、每个 Bug 的详细信息、应用评分等。
R5	执行二次分析	提测方如对 Bug 测试报告不够满意，可以申请对测试中间结果进行二次分析，生成二次分析数据与报告。
R6	查看二次分析状态	对已经开始进行二次分析的任务，提测方可以随时查看二次分析的执行状态。二次分析的状态包括：执行中、上传中、执行失败、上传失败、成功。

在应用提测方的业务流程中，他们首先可以在慕测企业版云平台中创建安卓应用自动化测试任务，上传 APK 并填写相关任务配置。之后，他们可以随时启动自动化测试任务。启动后本系统会收到执行任务的请求，随后使用机柜中空闲设备，对待测应用进行自动化遍历测试。在测试过程中，提测方可以随时查看任务的执行状态，或者终止该测试任务。如果提测方未进行终止任务的操作，那么该任务则会在所有设备执行自动化测试结束后自动终止。不论任务是手动或自动终止的，在任务结束后，提测方都可以获得一份可读性很高的自动化测试 Bug 报告，报告是对测试结果的总结，包括应用的基本信息、任务的基本信息、每台设备的执行信息、Bug 列表、每个 Bug 的详情等、系统对应用的评分等。最后，提测方如对测试结果并不满意，则可以提交二次分析测试中间数据的

任务，提交后，本系统将会根据所有测试中间数据，对其进行二次分析，生成二次分析的数据与报告。

3.2.3　非功能需求分析

考虑到本系统定位是为慕测科技企业版云平台提供安卓应用自动化测试服务，所以本系统在可用性、稳定性、可拓展性、性能等方面都有一定的要求。经过仔细的分析和深入的探讨，本系统的非功能需求如表 3.3所示。

表 3.3: 系统非功能需求列表

需求名称	需求描述
可用性	本系统应该保证全年 99% 的可用性。如因断电等意外情况导致系统掉线或不可用，本系统应该在宿主机恢复工作 15 分钟内重新启动并提供稳定服务。
稳定性	本系统应提供稳定的安卓应用自动化测试服务，由于系统需要使用手机等物理设备进行测试，存在许多导致测试流程异常退出的不可控因素，系统需要保证在任何不可控因素下，都能提供正常的服务，根据需求生成最终的测试报告。
可扩展性	本系统应对系统关键功能提供友好抽象，以方面后续进行更多功能的扩展，如：测试遍历算法、报告生成服务等。
安全性	本系统应该提供严格的身份识别功能，以保障请求来源的合法性；本系统应该保证所储存数据不可被随意篡改，以保障数据的完整性，提供可靠的服务。
性能	在可靠的网络环境中，本系统接口的返回时间应该不大于 200ms，本系统数据库的读写操作时间应该不大于 500ms。
兼容性	本系统应该可以在操作系统为任何 Linux 发行版的物理服务器上部署和执行。
伸缩性	当测试任务过多超过系统负载时，应可以通过快速部署新的机柜系统提问更多的测试服务，且该扩展对上游系统透明。

因为本系统为部署在物理机柜上的系统，且需要对安卓手机等物理设备进行操作，所以与常见的部署在云服务器上的系统有所不同。因机柜对其所在环境较为敏感，所以本系统需要有较高的可用性，以保障服务的正常提供。同时因为需要对安卓手机进行测试，所以需要保证测试过程的稳定性，以忽略因设备失联等问题导致测试终止的情况。因物理机柜可能会随时增减，所以系统应该提供较好的可伸缩性，且具有可以在不同环境的机柜中快速部署、启动、提供服务的能力。同时机柜中的安卓设备也可能会随时增减，所以系统应在不影响当前正在执行任务的情况下，动态的增减安卓设备。最后，因为本系统是一个测试中台系统，后续还需要提供更多的测试与分析服务，所以需要对测试与分析这

两个关键的功能点进行良好的抽象与设计，以方便团队进行后续的功能扩展和相关研究。

3.3　系统用例分析

3.3.1　系统用例图

通过涉众分析、功能与非功能需求分析，现得出系统的用例图如图 3.2所示。本系统只涉及到测试需求方一个涉众，共有开始测试、停止测试、查询测试状态、查询测试结果、分析测试中间结果等五个用例。

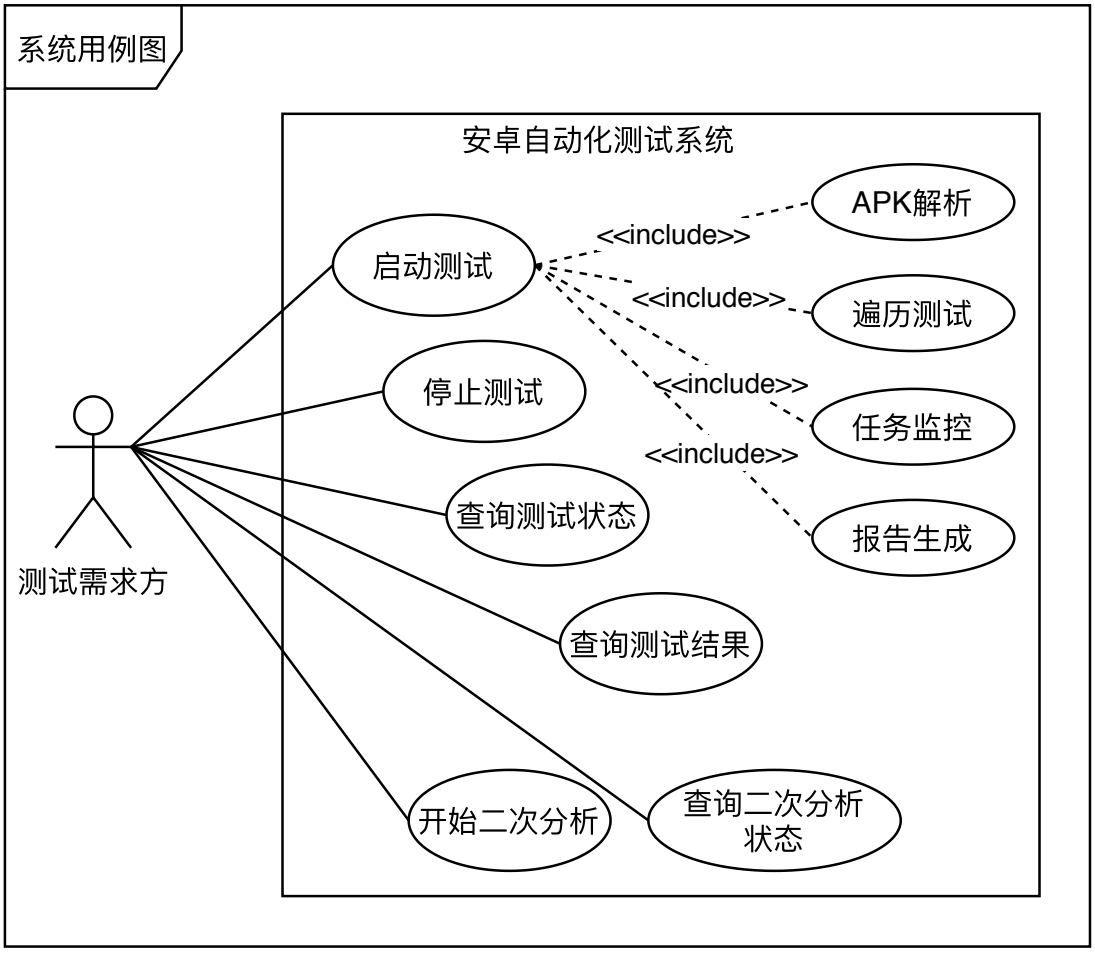

图 3.2: 系统用例图

3.3.2　系统用例描述

开始测试是本系统提供的最为关键的功能，测试需求方需要先创建安卓自动化测试任务，填写相关配置信息，启动本系统的测试功能。开始测试的详细用例描述如表 3.4所示。

表 3.4: 启动测试用例描述

ID	UC1
名称	启动安卓自动化测试
参与者	测试需求方
触发条件	用户点击开始测试按钮
前置条件	用户必须在上游服务中创建安卓应用自动化测试任务
后置条件	系统上游服务是否启动成功
优先级	高
正常流程	1. 用户已成功创建安卓自动化测试任务 2. 用户点击开始创建按钮 3. 本系统收到开始任务的请求，分配设备，执行自动化测试任务
异常流程	3a. 本系统收到的请求未通过权限验证 1. 系统拒绝该请求 2. 系统提示“权限验证未通过” 3b. 系统所需参数不完整 1. 系统拒绝该请求 2. 系统根据所缺参数作出相应提示

停止测试功能是向用户提供停止正在执行的自动化测试任务的能力。停止测试功能只向正在执行中的自动化测试任务开放，且必定成功，但是由于涉及到安卓物理设备，所以停止测试需要一定的收尾时间。停止测试的详细用例描述如表 3.5所示。

查询测试状态功能是向用户提供对正在执行的自动化测试任务的执行状态监控的能力。任务的状态包括以下几种：初始化、执行中、报告生成中、完成、下载失败、APK 解析失败等。查询测试状态的详细用例描述如表 3.6所示。

在任务结束后（不论是执行结束还是手动停止），用户可以查看当前任务的 Bug 报告，并下载任务报告以离线查阅。任务报告的内容包括：任务执行信息、任务基本信息、每台设备执行信息、潜在 Bug 列表、每个 Bug 的详细信息、APK 评分等。查看任务报告的详细用例描述如表 3.7所示。

二次分析功能在用户对自动化测试报告存疑或者有进一步需求的情况下，为用户提供了根据自动化测试结果，进行二次分析的能力。分析中间结果的详细用例描述如表 3.8所示。

二次分析状态查询是向用户提供对正在进行二次分析的服务状态监控的能力。任务状态包括以下几种：报告生成中、报告生成失败、报告上传中、报告上传失败、成功。二次分析状态查询的详细用例描述如表3.9所示。

表 3.5: 停止测试用例描述

ID	UC2
名称	停止安卓自动化测试
参与者	测试需求方
触发条件	用户点击停止测试按钮
前置条件	用户必须已成功开始任务
后置条件	系统提示停止成功
优先级	低
正常流程	1. 用户已成功启动安卓自动化测试任务 2. 用户点击停止测试按钮 3. 本系统收到停止任务的请求，停止对应 ID 的任务 4. 任务停止后生成相应的测试报告并上传
异常流程	2a. 本系统收到的请求未通过权限验证 1. 系统拒绝该请求 2. 系统提示“权限验证未通过” 3a. 所停止的任务 ID 不存在 1. 系统拒绝该请求 2. 系统提示“任务不存在” 3b. 所停止的任务已被停止 1. 系统拒绝该请求 2. 系统提示“任务已停止”
特殊需求	只能停止正在进行的任务

表 3.6: 查询测试状态用例描述

ID	UC3
名称	查询安卓自动化测试执行状态
参与者	测试需求方
触发条件	用户查看自动化测试任务详情
前置条件	用户必须已成功开始任务
后置条件	无
优先级	高
正常流程	1. 用户已成功启动安卓自动化测试任务 2. 用户点击任务详情按钮 3. 用户进入任务详情界面，并可以看到任务的执行状态
异常流程	3a. 所查询的任务 ID 不存在 1. 系统拒绝该请求 2. 系统提示“任务不存在”
特殊需求	只能查询已经创建成功的任务

表 3.7: 查询测试报告用例描述

ID	UC4
名称	查询安卓自动化测试执行报告
参与者	测试需求方
触发条件	用户查看自动化测试任务报告
前置条件	自动化测试任务已经结束
后置条件	用户可查看自动化测试任务报告
优先级	高
正常流程	1. 自动化测试任务已正常结束 2. 用户点击任务报告按钮 3. 用户进入任务报告界面，查看任务的详细执行报告
异常流程	2a. 所查询的任务 ID 不存在 1. 系统拒绝该请求 2. 系统提示“任务不存在”
特殊需求	只能查询已经执行结束的任务

表 3.8: 分析测试中间结果用例描述

ID	UC5
名称	分析测试中间结果
参与者	测试需求方
触发条件	用户点击二次分析按钮
前置条件	自动化测试任务已经结束
后置条件	生成二次分析报告
优先级	中
正常流程	1. 自动化测试任务已正常结束 2. 用户点击二次分析按钮 3. 系统调用二次分析服务，执行二次分析
异常流程	2a. 任务的中间结果已失效 1. 系统拒绝该请求 2. 系统提示“中间结果已失效”
特殊需求	只能操作已经执行结束的任务并成功生成报告的任务

3.4　系统总体设计

3.4.1　系统架构

本系统的系统架构图如图 3.3所示。目前，本系统的上游服务主要为慕测科技企业版与教育版云平台系统，企业版主要为企业客户提供自动化测试、众包测试等服务。教育版主要为高校客户提供测试课程相关的云服务。本系统采用 Spring Boot 开发框架，该框架具有非常强的快速上手能力以及扩展能力，所以可以快速搭建起后端项目以及引入相应外部依赖。在接入层中，系统主要对外

表 3.9: 二次分析状态查询用例描述

ID	UC6
名称	二次分析状态查询
参与者	测试需求方
触发条件	用户查看二次分析详情
前置条件	自动化测试任务已经结束
后置条件	无
优先级	中
正常流程	1. 自动化测试任务已正常结束 2. 用户点击查看二次分析按钮 3. 系统进入二次分析页面并可以看到分析的执行状态或结果
异常流程	2a. 所查询的任务 ID 不存在 1. 系统拒绝该请求 2. 系统提示“任务不存在”
特殊需求	只能操作已经执行结束的任务并成功生成报告的任务

提供了开始任务、任务状态与结果查询、结束任务、结果二次分析等功能。上游服务可通过 HTTP 协议发送响应请求到服务端。因为每次任务都比较耗时，所以所有的接入层接口均设计为异步执行，采用 Java 原生线程解决方案。

APK 服务提供 APK 包的下载及解析服务，aapt 工具为安卓开发套件提供的资源打包工具，它也可以用于解析 APK 的基础信息。自动化测试服务提供 APK 基础测试、设备监控以及脚本执行功能。基础测试包括基础安装、覆盖安装、应用卸载、应用冷启动等。设备监控包括监控设备测试中的 CPU、内存、刷新率以及网络情况。脚本执行会通过 Appium client 与外部的 Appium server 进行交互，控制安卓设备从而进行测试。在测试执行的过程中，监控服务会对所有设备执行线程进行监控，待任务结束后调用报告生成服务生成最终的报告。本系统外部存储使用了 Redis 缓存，Linux 文件系统以及阿里云对象存储 OSS 服务。Redis 用于储存任务状态、设备执行状态的临时信息，这些信息会随着任务结束而变得无意义，使用缓存可以大大减少数据库的压力以及提高数据更新效率。Linux 文件系统主要用于存储设备执行任务阶段所产生的中间数据，如各类日志、截屏、执行信息等。阿里云对象存储 OSS 服务主要用于对最终结果的储存，包括 Bug 报告、相关截图等。阿里云 OSS 可以提供十二个九的可靠性，且效率较高成本较低，是目前云存储的主流解决方案。

3.4.2　4+1 视图

本文参考 Philippe Kruchten 教授提出的“4+1”视图方法给出系统的架构设计 [40]。“4+1”视图是从五个不同的角度来描述软件的逻辑架构，包括逻辑视

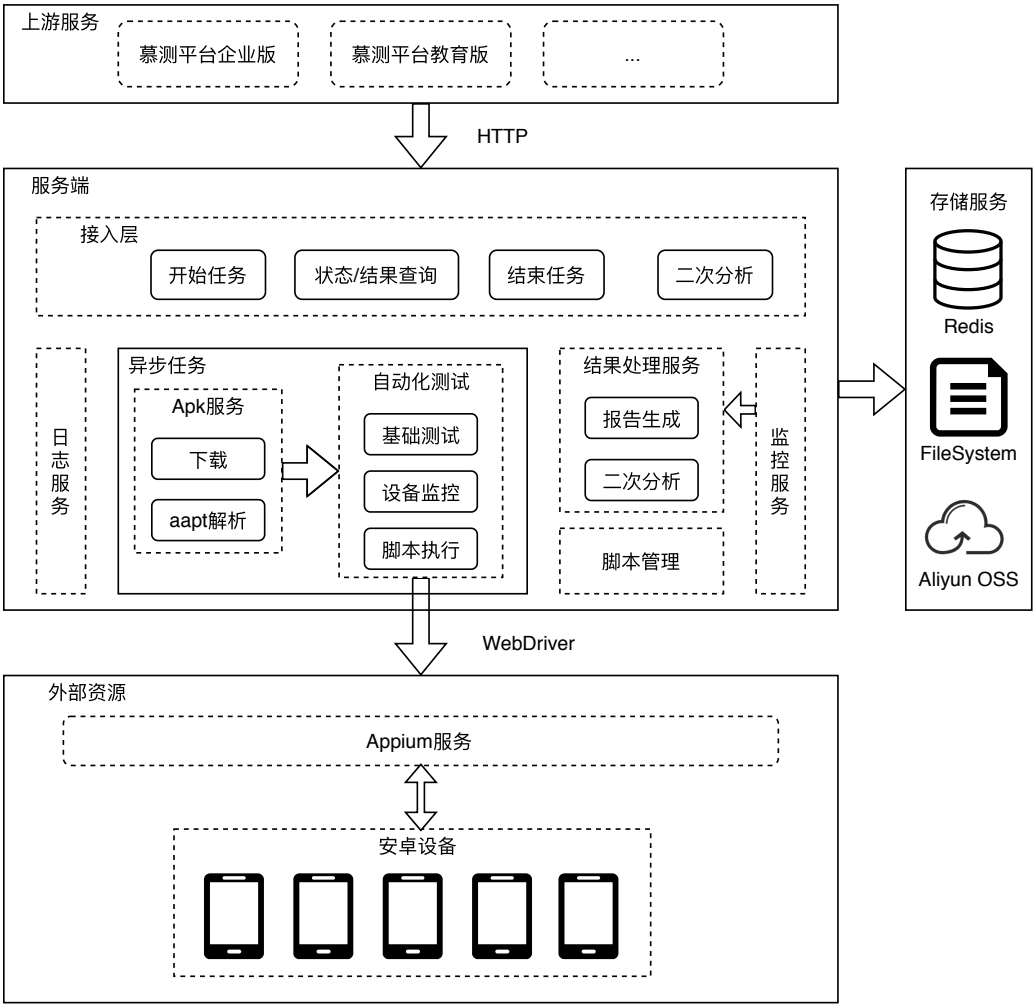

图 3.3: 系统架构图

图、开发视图、进程视图、物理部署视图和场景视图。场景视图已在第 3.3 章节用例分析中进行了详细的描述，故不再赘述。本章节重点从其他四个视图的角度，对本系统的体系结构进行描述。

（1） 逻辑视图

逻辑视图所面向的对象是最终用户，其需要向最终用户直观的表达出系统向用户提供了哪些功能性需求。在逻辑视图中，系统分解为一系列的功能抽象，所以在像 Java 这种面向对象的语言中，常常使用 UML 类图来描述逻辑视图。本系统的逻辑视图如图 3.4所示。

AutoTestService 为本系统最核心的服务，所提供是与自动化测试相关的功能，包括开始、停止、查询结果与状态。ApkService 提供了 Apk 文件下载与解析

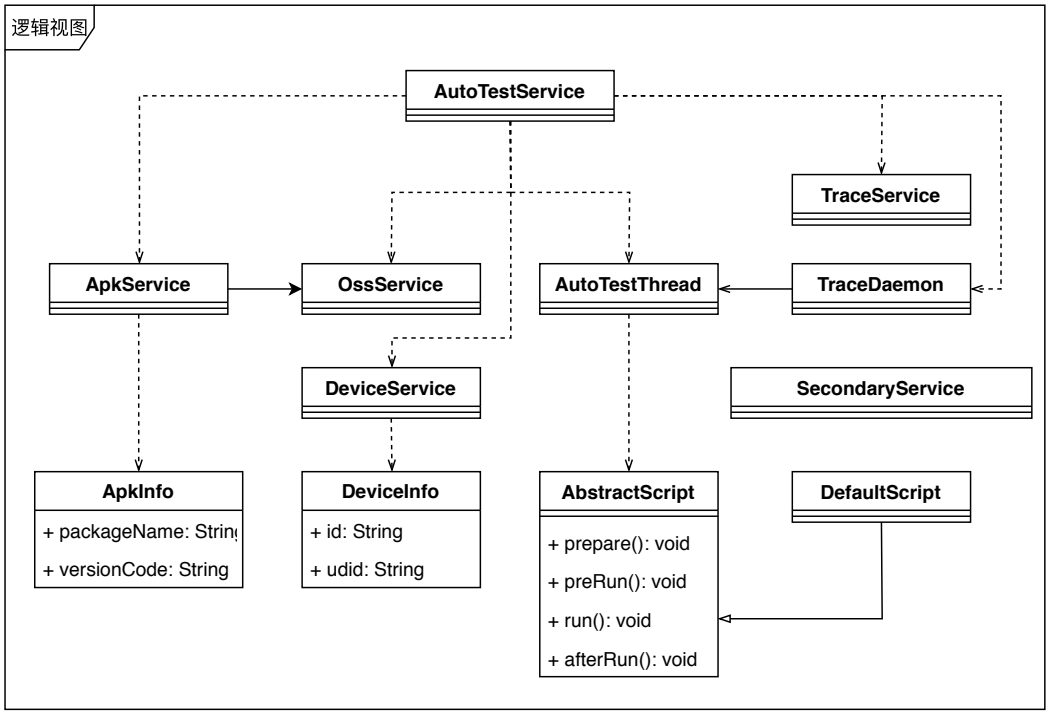

图 3.4: 系统逻辑视图

相关的服务，它会将需要测试的 APK 文件解析成更为抽象的 ApkInfo 对象，供后续测试使用。OssService 提供了与阿里云云存储 OSS 进行交互的服务，主要是对上传与下载进行封装与抽象。TraceService 提供了与任务相关的服务，包括获取、更新任务状态，获取任务所选设备及信息等功能。DeviceService 提供了与安卓设备相关的服务，包括获取、更新设备状态，获取设备详细信息，获取空闲设备列表等。DeviceInfo 为每台安卓设备的抽象，其包括每台安卓设备的各种基本信息。AutoTestThread 为单台设备自动化测试的线程对象，提供了单台安卓设备上自动化测试服务。TraceDaemon 提供了自动化测试单次任务的监控服务以及任务结束后的收尾服务，它会每 5 秒对所有设备的运行状态进行检查，如所有设备均已执行结束，则会进行收尾工作。AbstractScript 为执行自动化算法的抽象对象，其只提供一个抽象模板方法，具体算法均由相关实现类实现，DefaultScript 为本系统所提供的默认实现类，通过 DFS 深度优先遍历实现对应用的遍历测试。SecondaryService 提供了中间结果二次分析的服务。

（2） 开发视图

开发视图所面向的对象为专业的软件开发人员，主要侧重于展现软件系统模块的包组织与管理。本系统的开发视图如图 3.5所示。

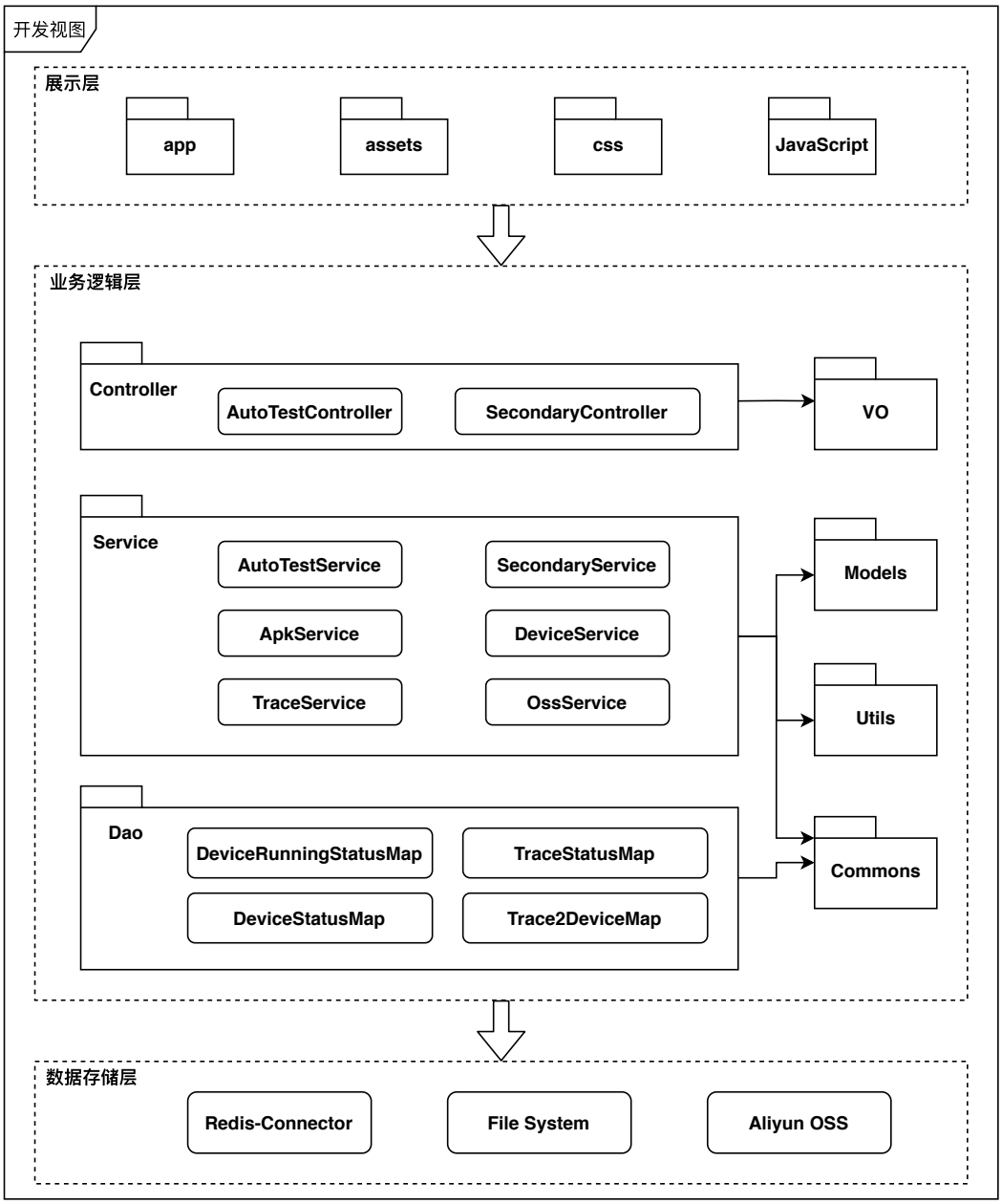

图 3.5: 系统开发视图

本系统为分层架构，开发视图总共分为三层。在展示层中，app 包中存放是的前端代码，其中代码会以功能进行在细分；assets 包中存放的是前端静态资源文件，包括图片、json 文件、i18n 资源等；CSS 目录存放的是前端的样式文件；JavaScript 文件存放的是前端所使用到的第三方依赖文件。在业务逻辑层中，Controller 包负责接收前端请求并进行路由分发；Service 包负责处理具体的业务逻辑，并对 Controller 层提供业务逻辑支撑；Dao 包中负责对底层数据存储进行

封装与抽象，并对 Service 层提供数据支撑。其他资源文件中，VO 包储存的是于前端进行交互的 VO 对象；Models 包存储的是系统所用到的对象实体；Utils 包中提供的是各种静态工具类，包括执行系统命令、打印日志、解析 XML 文件等；Commons 包中负责存储的是各种静态常量，包括字符串常量、枚举值、异常对象等。数据存储层主要提供的是一些数据存储服务，包括 Redis、文件系统及阿里云 OSS 云存储。

（3） 进程视图

进程视图主要是从系统处理请求过程的角度，描述系统的并发与同步特性，旨在解决进程、线程、并发、同步、通信等方面的问题。本系统的进程视图如图 3.6所示。

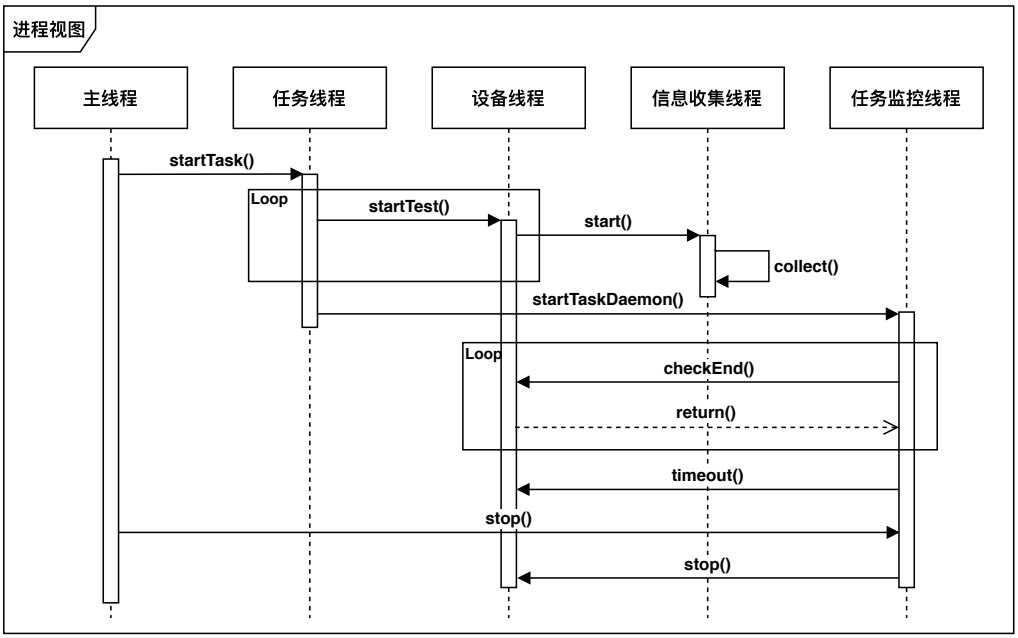

图 3.6: 系统进程视图

当主线程收到开始执行任务的请求，因为执行自动化测试任务是一个耗时操作，所以会为此次任务创建任务线程，成功后直接返回给上游服务。在任务线程中，会为每一台被选中的安卓设备创建设备执行线程，每个执行线程负责一台特定安卓设备的自动化执行，之后，任务线程会创建监控线程以监控此次任务中所有的设备线程。在每个设备线程开始时，会分别创建截取屏幕、收集 Appium 日志、收集安卓 Logcat、处理设备实时信息等四个线程，用于对该台安卓设备执行时信息的收集与处理。在任务监控线程中，会以每五秒一次的速率检查所有设备线程的状态，如果所有设备线程均已结束，则执行任务收尾操作；

同时，任务监控线程会记录任务执行时间，如果任务超时，则会向所有设备执行线程发送超时命令，强行停止所有设备的执行。另外，如果主线程收到任务停止的请求，也会向任务监控线程发送立即停止指令，任务监控线程则会向所有设备执行线程发送停止命令，以停止所有设备的执行。

（4） 物理视图

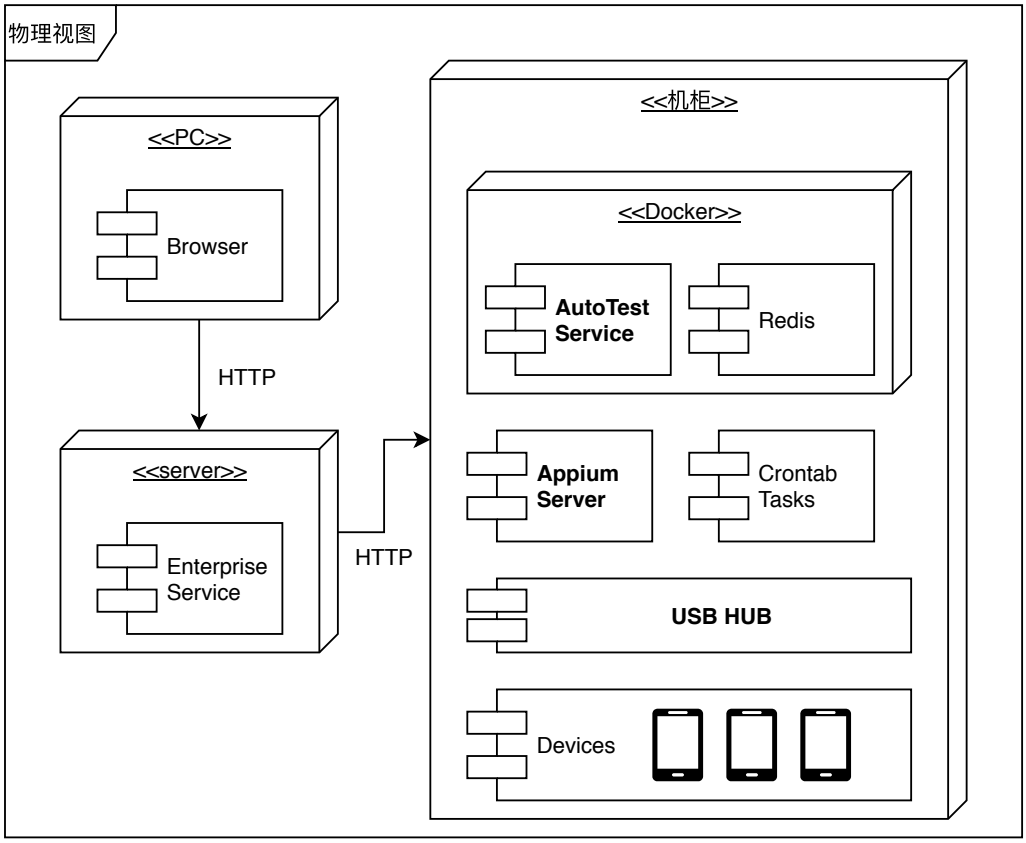

图 3.7: 系统物理视图

物理视图是从软件部署的角度，用来描述系统软件到硬件的映射关系，旨在解决系统安卓、系统部署等问题，主要面向的对象系统的运维人员。本系统的物理视图如图 3.7所示。

用户通过个人电脑上的浏览器，访问慕测企业版云平台系统，也就是本系统的上游系统，然后上游系统通过 HTTP 请求向本系统发送请求。本系统（AutoTestService）通过 Docker 部署在物理机柜上，通过 Docker 进行部署可以实现系统的快速部署与迁移。同时 Redis 缓存数据库也使用 Docker 的方案进行部署。另外，机柜上还部署了 Appium Server 服务以及若干定时任务，Appium Server 是本系统和各个安卓设备之间通信的中间层，各个定时任务是为了解决

自动化测试过程中出现的若干特异性问题。最重要的是，机柜中还需通过 USB HUB 连接若干台安卓设备，用于执行自动化测试任务。USB HUB 可以提供充足的 USB 接口和提供稳定的供电。

3.4.3　自动化任务流程设计

图 3.8为系统默认状态下接收到自动化测试任务的流程图。当系统接收到自动化测试任务时，首先通过 APK 服务，对 APK 文件进行下载与解析；之后会通过设备服务获取到当前在线的所有设备；然后对获取到的设备进行筛选，如果筛选后并无空闲设备，则直接结束本次任务；如果有空闲设备，则继续对每台设备进行遍历。当遍历到某台设备时，首先通过 CAS 更新该设备的状态，如果更新成功，则对该设备启动自动化执行线程；如果更新失败，则说明该设备已被其他任务抢先占用，则跳过该设备。最后，启动监控线程，对所有的自动化执行线程进行监控。至此，刚刚接收到的任务已启动完毕。

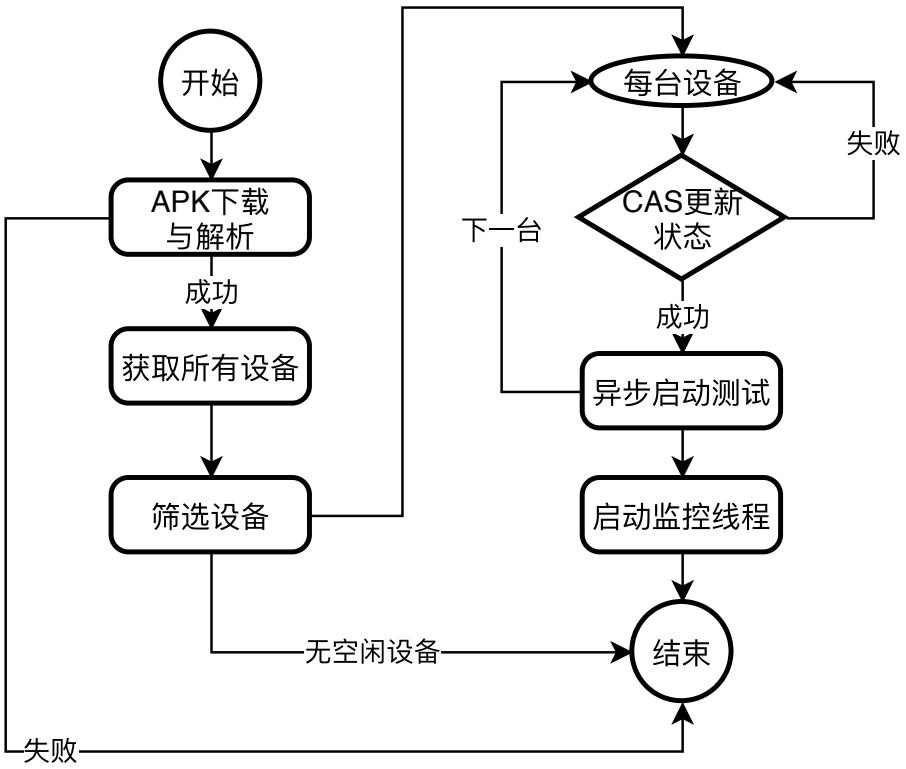

图 3.8: 接收任务流程图

另外，因为本系统为部署在物理机柜上的系统，所以可能存在物理机柜没有公网 IP 无法接收到 HTTP 任务请求的情况。在这种情况下，本系统需要支持

没有公网 IP 的物理机柜，较好的解决方案为本系统通过 HTTP 请求，主动从上游系统拉取并执行自动化测试任务，由上游系统控制任务的执行与分发。这种场景的流程图如图 3.9所示。

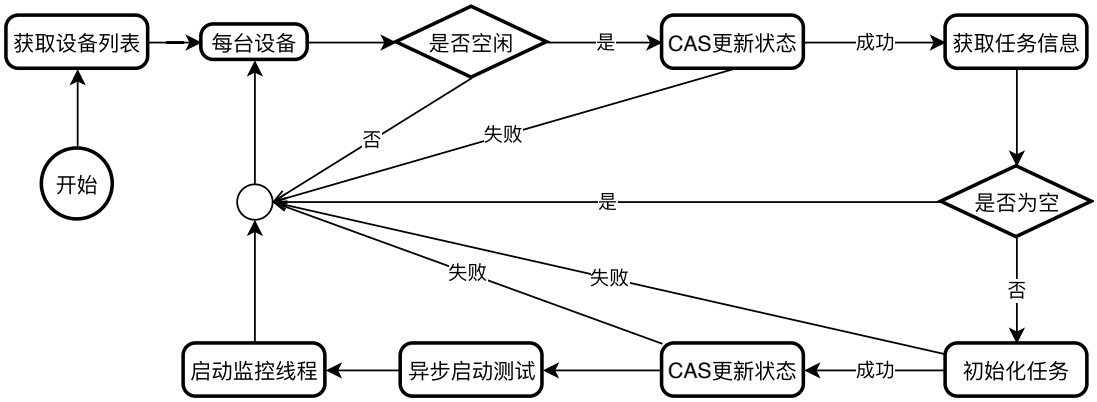

图 3.9: 拉取任务流程图

在此流程中，当系统启动时，会首先获取并遍历当前系统连接的设备列表。对于每台设备，首先检查设备是否空闲，如果空闲，则尝试通过 CAS 将设备状态更新为占用，如果更新失败，则说明该设备已经被占用，则跳过该设备。之后，系统便会获取设备的各类信息，包括品牌、机型、分辨率、安卓版本等，打包发送给上游服务，上游服务接收到请求后，会返回给本系统一个任务信息。如果任务信息为空，则说明上游服务暂时没有适合此设备的任务；如果任务信息不为空，则接着进行任务初始化操作。任务初始化成功后，会再次通过 CAS 操作，将设备的状态从占用更改为执行。更改成功后，则会与接收任务的场景下一样，启动异步线程执行自动化测试任务，最后启动监控线程对自动化测试任务进行监控。完成这一系列操作后，便会选择下一台设备，继续执行上述操作。

3.4.4　自动化遍历算法流程设计

本系统在默认情况下会提供一套默认脚本，用于执行自动化测试任务。该脚本的算法基于 DFS 深度优先遍历，其流程图如图 3.10所示。

首先，每一次 DFS 流程开始的时机为遍历到一个全新的页面，所以第一步需要先处理页面上的弹窗以及可能出现的权限申请操作，之后，通过 AppiumDriver 获取到当前页面的组件列表，通过组件列表判断当前页面是否还在 APP 中，如果不在，则退出当前的测试流程；如果在，则继续判断该页面是否出现过。如果页面已经出现过，则更新页面信息；如果页面未出现过，则将页面加入页面列表并更新信息。之后检查页面是否已被测试完成，如果测试完成，则点击 return

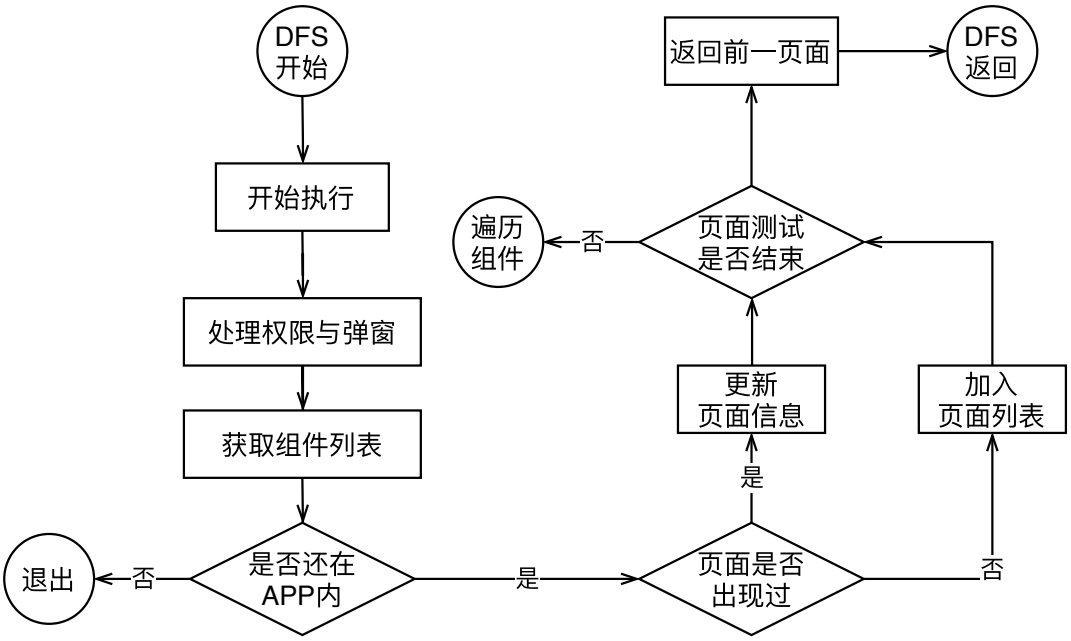

图 3.10: DFS 算法流程图

键，此次 DFS 返回；如果未完成，则对之前扫描到的组件列表进行循环点击，组件遍历操作的流程图如图 3.11所示。

首先，判断该组件是否已被点击过，或者被配置文件忽略，如果否，则继续。之后，根据该组件的类型，执行点击、输入等操作，操作完成后，需要再次处理权限以及系统弹窗，处理完成后，则需要对当前页面的状态进行判断。首先判断操作前后的页面是否相同，如果不相同，则说明此次操作进入了一个新的页面，那么，则需要进行如图 3.10所示的更深一级的 DFS 遍历；如果页面相同，则需判断页面上的组件是否发生了变化，如果组件未发生变化，则说明此次操作并未引起任何页面上的变化，那么则直接进行下一组件的遍历；如果发生了变化，则需要对组件列表进行更新，然后再进行下一个组件的遍历。当更深一层的 DFS 遍历返回时，此时说明因为刚刚操作进入的新页面已经遍历结束了，那么此时则需先处理安卓的弹窗，然后判断当前页面是否还属于待测 APP，如果已经跳出了 APP，则退出整个遍历流程；如果为跳出，那么则再次更新组件列表，更新之后继续遍历下一个组件。

整个遍历的流程中，DFS 的深度会随着新页面的出现而逐渐变深，每当进入到一个新页面，就会进行更深一层的 DFS 遍历；同时，在每个页面的遍历过程中，会对页面中的所有组件逐个进行操作，直至组件遍历完成。

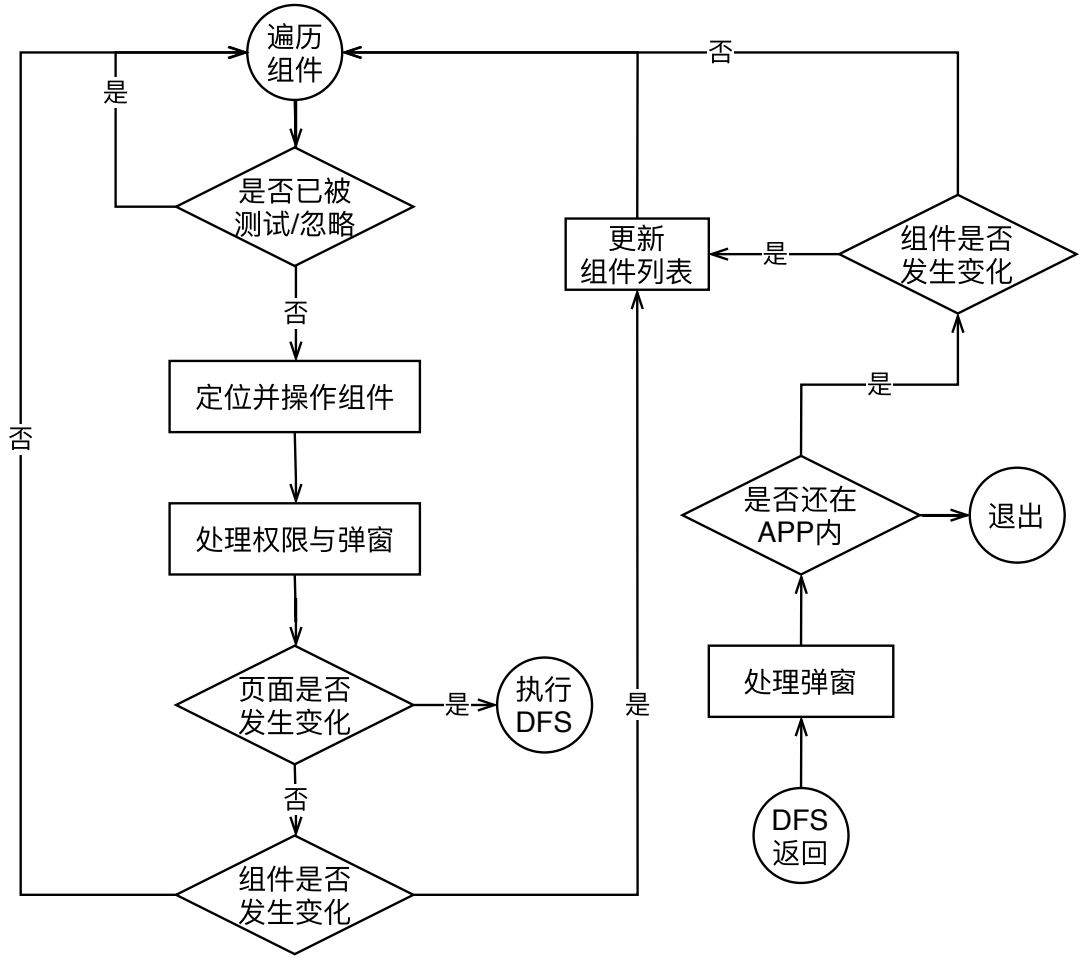

图 3.11: 组件遍历流程图

3.5　数据库与实体设计

本系统因为没有需要持久化的数据，所以并未使用 MySQL 数据库。任务和设备的临时状态信息均缓存在 Redis 数据库中，这些信息会随着任务结束而失去意义。Redis 数据库中缓存的信息如表 3.10所示。

系统中还有若干数据实体，包括 APK 信息、Device 设备信息、安卓 Activity 对象、安卓 Component 对象、自动化测试操作 Action 对象等。

APK 信息为 APK 文件本身的抽象对象，包括 APK 的应用名、版本号、包名、支持的安卓版本等若干静态信息。其设计如表 3.11所示。

Device 对象为当前连接到系统里的安卓设备的抽象。主要记录的是设备的唯一 ID、系统版本、型号、品牌、分辨率等若干静态信息，供自动化测试使用。其属性设计如表 3.12所示。

表 3.10: Redis 数据库设计

KEY	类型	VALUE 类型	说明
TRACE_STATUS_{traceId}	String	String	保存 TraceStatus 对象的 code，缓存任务状态
DEVICE_STATUS_{udid}	String	String	保存 DeviceStatus 对象的 code，缓存设备的占用状态
DEVICE_RUNNING_STATUS_{udid}	String	String	保存 DeviceRunningStatus 对象的 code，缓存设备的任务执行状态
TRACE_DEVICES_{traceId}	String	列表	保存执行此次任务的设备 udid
TRACE_SECONDARY_{traceId}_{toolName}	String	String	保存 SecondaryStatus 对象的 code，缓存二次分析状态

表 3.11: ApkInfo 对象属性表

字段	含义	类型	说明
versionCode	Apk 内部版本号	String	此属性对用户不可见
versionName	Apk 外部版本号	String	此属性对用户可见
packageName	Apk 包名	String	作为唯一标识区别应用
minSdkVersion	支持的最小 SDK 版本	String	低于此版本设备将无法安装
usesPermissions	APK 声明的权限	List<String>	
targetSdkVersion	APK 的目标版本	String	
applicationLable	应用名称	String	
applicationIcon	应用图标位置	String	
launchableActivity	应用入口 Activity	String	该属性有且只有一个，才可以正常进行测试

表 3.12: Device 对象属性表

字段	含义	类型	说明
udid	设备唯一 ID	String	此 ID 设备出厂时即确定且无法修改
os	设备安卓系统版本	String	
deviceModel	设备型号	String	
deviceName	设备名称	String	
brand	设备品牌	String	
resolution	设备分辨率	String	

Activity 对象为安卓应用中 Activity 实例的抽象，一般来说，一个 Activity 对应应用的一个页面，此对象主要记录页面的各类信息以及在自动化遍历过程中

的状态信息等。其属性设计如表 3.13所示。

表 3.13: Activity 对象属性表

字段	含义	类型	说明
name	名称	String	此字段在应用中唯一
fatherActivity	父页面	Activity	如无 father，则为空字符串
hash	Activity 对象的 Hash 值	Integer	值为 name 字符串的 Hash 值
comList	页面上的组件列表	List<Component>	
isDone	该页面是否遍历结束	Boolean	默认为 false

Component 对象为安卓应用中各类组件的抽象，如输入框、按钮等，此对象记录了安卓应用组件的类型、ID、XPath、内容、状态、位置等信息，这些信息可以用于自动化遍历过程中对各个组件的识别与操作。其属性设计如表 3.14所示。

表 3.14: Component 对象属性表

字段	含义	类型	说明
resource_id	组件 ID	String	此字段在应用不保证唯一
locator	组件定位符	String	此字段为组件 XPath 值或 resource_id
index	组件在页面上的索引值	String	
text	组件上文字内容	String	
classname	组件的类型	String	
packagename	组件所属的报名	String	与应用包名一致
content_desc	组件的描述	String	一般用于无障碍功能
checkable	组件是否可以勾选	Boolean	
checked	组件是否已被勾选	Boolean	
clickable	组件是否可以被点击	Boolean	
enabled	组件是否可用	Boolean	
focusable	组件是否可以获取焦点	Boolean	
focused	组件是否已获取焦点	Boolean	
scrollable	组件是否可以滚动	Boolean	
long_clickable	组件是否可以长按	Boolean	
password	组件是否是密码输入组件	Boolean	
selected	组件是否被选中	Boolean	
bounds	组件的绝对位置	String	
hasBeenTested	组件是否已被测试	Strings	
fatherComponent	组件的父组件	Component	

Action 对象本系统在自动化测试过程中操作的抽象，主要用于记录在执行了点击、滑动和输入等操作前后应用的状态。其属性设计如表 3.15所示。

表 3.15: Action 对象属性表

字段	含义	类型	说明
timeBeforeAction	操作前系统时间	String	此时间为设备系统时间
timeAfterAction	操作后系统时间	String	此时间为设备系统时间
type	操作类型	String	包括：点击、滑动、输入
activityBeforeAction	操作前页面	String	
activityAfterAction	操作后页面	String	

3.6　本章小结

本章主要概述了安卓应用自动化测试系统的需求分析部分。首先对本系统的涉众、功能性需求与非功能性需求进行了简要分析，在此基础上，用图表的形式给出了系统的用例图与各个用例的描述。其次从系统的整体架构设计、逻辑视图、开发视图、进程视图与物理视图等不同角度，对系统进行了更为详细的描述。之后，针对有无公网 IP 的两种场景，给出了两种自动化测试任务执行流程的设计方案。最后，介绍了系统内所使用的数据存储服务以及相关的对象实体设计思路以及字段描述。

第四章　安卓应用自动化测试系统详细设计与实现

4.1　自动化测试服务的设计与实现

4.1.1　自动化测试服务架构图

图 4.1为本系统自动化测试服务的架构图。对外，自动化测试服务通过 HTTP 协议提供测试启动、查询与停止的功能；对内，自动化测试服务主要负责串联各类服务、控制调度流程以及保障流程的畅通与可靠。本章节后面所提及的 APK 服务、设备服务、测试执行服务、任务监控服务均由自动化测试服务统一调度与管理。

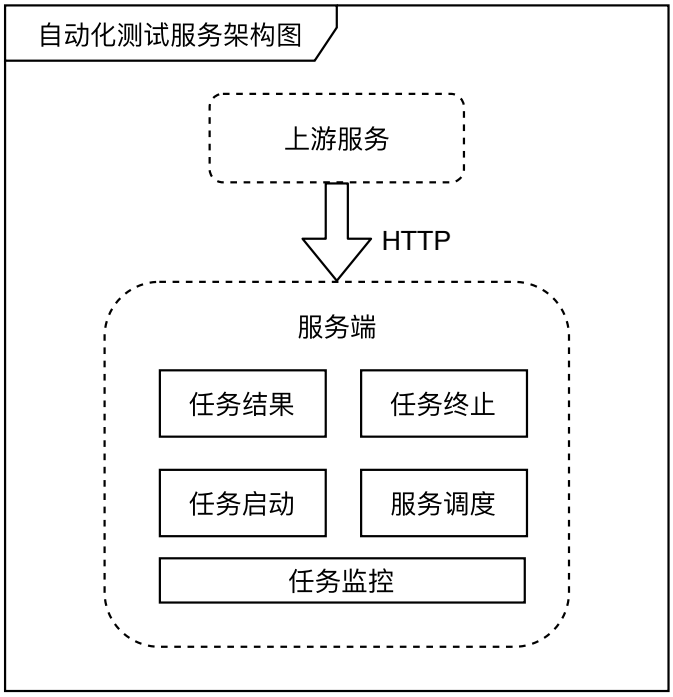

图 4.1: 自动化测试服务-架构图

4.1.2　自动化测试服务核心类图

图 4.2为自动化测试服务的核心类图。AutoTestController 为自动化测试服务对外提供 HTTP 服务与路由的控制类，其通过 AutoTestService 类封装的抽象接口来实现任务的开始、查询与终止功能。AutoTestServiceImpl 为 AutoTestService 的

实现类，用于实现接口对外提供的任务开始、查询、终止等功能。同时，AutoTestServiceImpl 也为 Apk 服务、设备服务、监控服务、OSS 服务等服务的调用方，通过调用不同的服务，推进自动化测试的流程。对于任务开始功能，由于任务执行较为耗时，所以在 AutoTestServiceImpl 中使用了内部线程类 RunTraceThread，用于异步启动自动化测试任务。另外，TraceMetaInfo 为任务元信息的抽象类，TraceStatusResult 为任务状态与结果信息的抽象类。

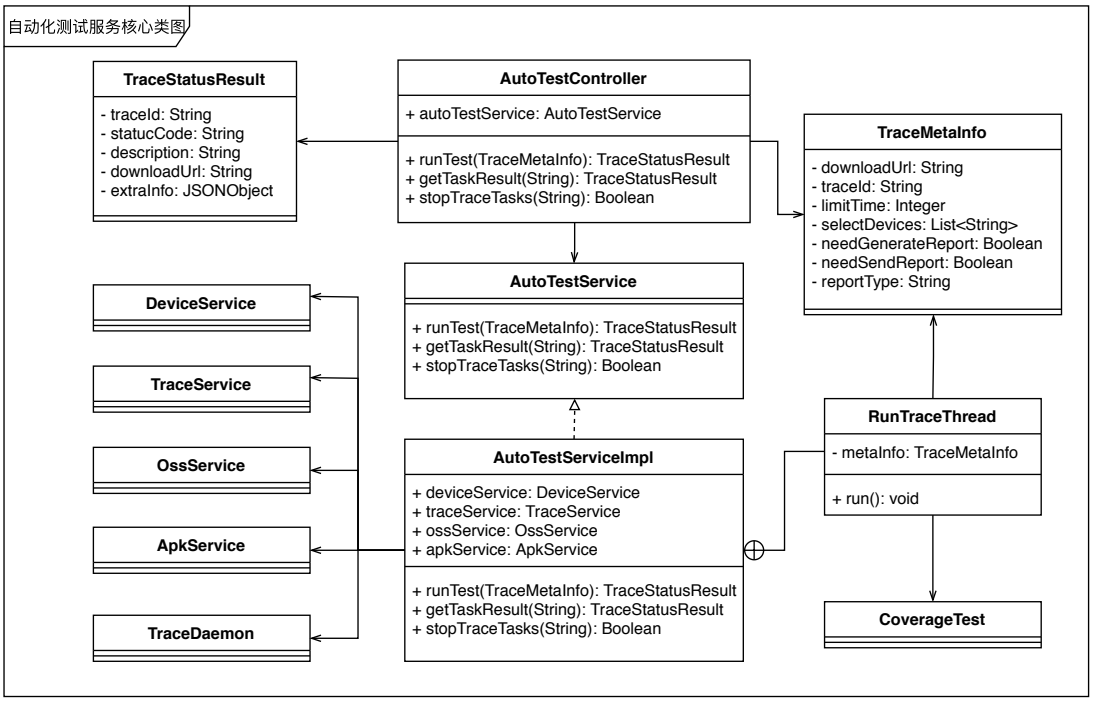

图 4.2: 自动化测试服务-核心类图

4.1.3　自动化测试服务顺序图

图 4.3为自动化测试服务的顺序图。当收到自动化测试任务开始的请求时，AutoTestController 会调用 AutoTestService 提供的 start() 方法开始自动化测试任务。在任务开始前，会先使用 ApkService 下载并解析 APK 文件，之后通过 DeviceService 获取到当前可用的设备列表，然后针对每一台设备构造出 CoverageTest 设备执行类，将任务分发给每一台设备，最后启动 TraceDaemon 任务监控类，对任务总体的流程与状态进行监控。同时，AutoTestService 还提供了 getStatus() 与 stop() 接口，用于提供查询任务状态与停止任务的功能。

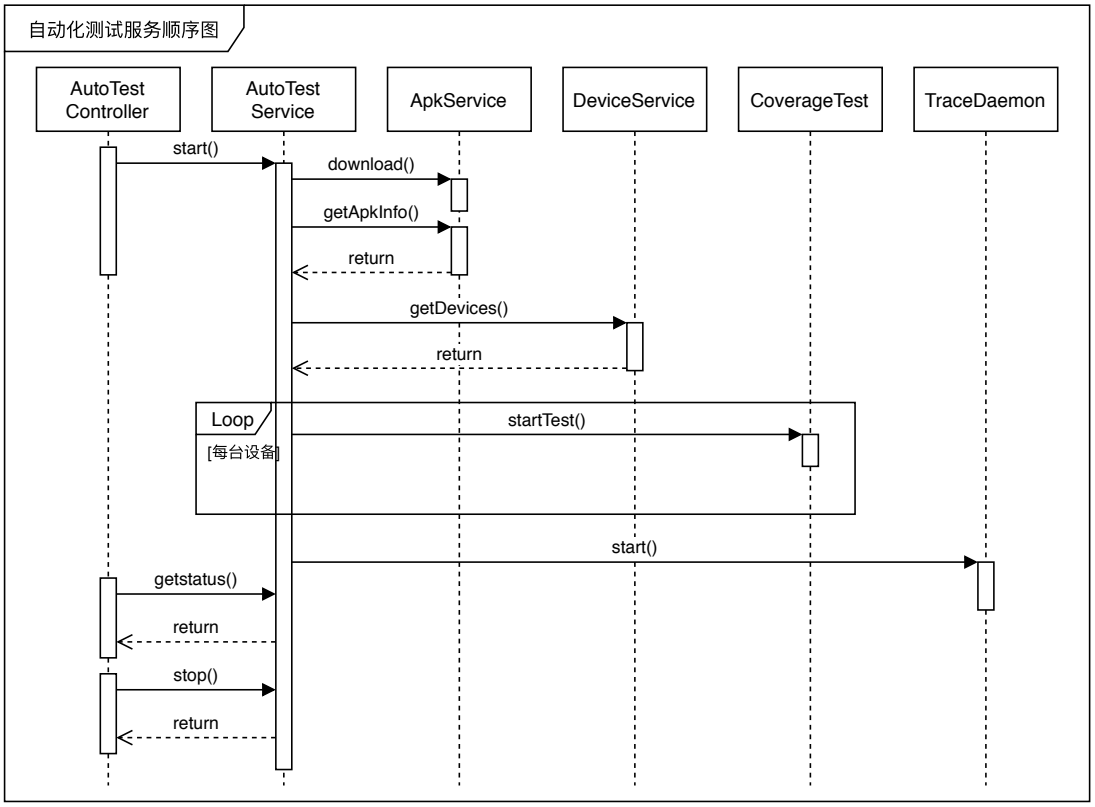

图 4.3: 自动化测试服务-顺序图

4.1.4　自动化测试服务关键代码

图 4.4为自动化测试服务任务开始的关键代码。在收到任务开始的请求后，首先通过 ApkService 进行 APK 的下载、解析与信息保存。之后通过 DeviceService 获取到当前在线的设备列表，获取到设备列表之后，通过和指定设备的对比，进行设备筛选。获取到设备筛选的结果后，使用 CAS 操作将设备的状态更新为占用中，并删除更新失败的设备，因为更新失败说明该设备已被其他任务占用。然后对每一台选中的设备，将设备状态更新为执行中，构造并执行测试执行线程 CoverageTest。最后将所有的测试执行线程 CoverageTest 传入任务监控线程 TraceDaemon 中，启动任务监控线程监控所有设备测试执行的状态。

图 4.5为自动化测试服务查询测试状态的关键代码。当收到查询任务状态的请求时，首先从 Redis 数据库中查询该任务的执行状态，并组装成 TraceStatus-Result 对象；之后根据任务 ID 去获取该任务所使用的设备，并获取每一台设备当前的执行状态；最后如果任务的状态为完成，则通过 OssService 获取到任务报告的下载链接。最终将任务状态的抽象对象 TraceStatusResult 返回。

图 4.6为停止自动化测试任务的关键代码。因为本系统大部分的执行逻辑都

```
@Override
public void run(){
    try {
        String filePath = apkService.downloadApk(downloadUrl, traceId);
        ApkInfo apkInfo = apkService.parseAPK(filePath, traceId);
        apkService.saveApkInfo(apkInfo, filePath, traceId);
    } catch (Exception e){}
    // 省略获取设备的代码，这里的流程为获取在线设备 ->筛选设备 ->CAS 更新设备状态
    // 省略任务状态更新的代码
    List<CoverageTest> oneTraceTasks = new ArrayList<>(finalDevices.size());
    for (Device device: finalDevices){
        deviceService.startRunDevice(device.getUdid());
        CoverageTest coverageTest = new CoverageTest(apkInfo, filePath, device, traceId);
        coverageTest.setName(Consts.AUTO_TEST_THREAD_NAME_PREFIX + device.getUdid());
        coverageTest.start();
        oneTraceTasks.add(coverageTest);
    }
    //启动监控线程监控该任务中所有执行线程
    TraceDaemon traceDaemon = new TraceDaemon(traceId,timeout, oneTraceTasks,true,
            false, traceInfo.getReportType());
    traceDaemon.setName(Consts.DAEMON_THREAD_NAME_PREFIX + traceId);
    traceDaemon.start();
}
```

图 4.4: 自动化测试服务-任务开始的实现

```
@Override
public TraceStatusResult getResult(String traceId){
    TraceStatus currentStatus = traceService.getTraceStatue(traceId);
    // 省略组装 TraceStatusResult 对象的代码
    // 获取执行该任务的设备状态
    List<DeviceStatusResult> deviceStatus = traceService.getTraceDeviceStatus(traceId);
    JSONObject extraInfo = JSON.parseObject("{}");
    extraInfo.put("deviceStatus", deviceStatus);
    traceResult.setExtraInfo(extraInfo);
    // 如果任务完成，则获取任务报告下载链接
    if (TraceStatus.FINISH.equals(currentStatus)){
        String downloadUrl = ossService.path(traceId, Consts.REPORT_FILE_NAME);
        traceResult.setDownloadUrl(downloadUrl);
    }
    return traceResult;
}
```

图 4.5: 自动化测试服务-查询任务状态的实现

是通过异步线程实现的，所以首先需要获取到当前系统中正在运行的线程，之后去查找指定 ID 的任务监控线程。查找到之后，通过调用其 stopImmediately() 发送停止指令，之后由监控线程负责向其监控的所有测试执行线程发送停止指令。如果未找到指定 ID 的任务监控线程，则抛出 404 异常。

```
@Override
public boolean stopTrace(String traceId){
    // 获取当前正在运行的线程
    ThreadGroup threadGroup = Thread.currentThread().getThreadGroup();
    Thread[] runningThreads = new Thread[threadGroup.activeCount()];
    threadGroup.enumerate(runningThreads);
    String targetName = Consts.DAEMON_THREAD_NAME_PREFIX + traceId;
    // 查找任务线程，并停止
    for (Thread t: runningThreads){
        if (t.getName().equals(targetName)){
            ((TraceDaemon)t).stopImmediately();
            return true;
        }
    }
    // 未找到则抛出异常
    throw new HttpNotFoundException(String.format("TraceId [%s] 不存在", traceId));
}
```

图 4.6: 自动化测试服务-终止任务的实现

4.2 APK 服务的设计与实现

4.2.1 APK 服务架构设计

APK 服务为系统接收到任务首先调用的服务，主要负责 APK 文件的下载与解析工作。其架构图如图 4.7所示。

系统接收到任务信息后，首先要对 APK 下载链接进行有效性校验。校验通过后，则会使用 Java 原生工具类 FileUtils 进行文件下载。因为阿里云 OSS 服务对网络环境较为敏感，所以此处加入了重试机制以保证 APK 下载的稳定性。下载成功后，则会使用安卓开发套件原生提供的 aapt 工具对 APK 文件进行解析。aapt 工具解析出的结果为半结构化的文本信息，只需要对该文本信息通过模式匹配进行二次解析，构建 ApkInfo 对象，供后续测试使用。

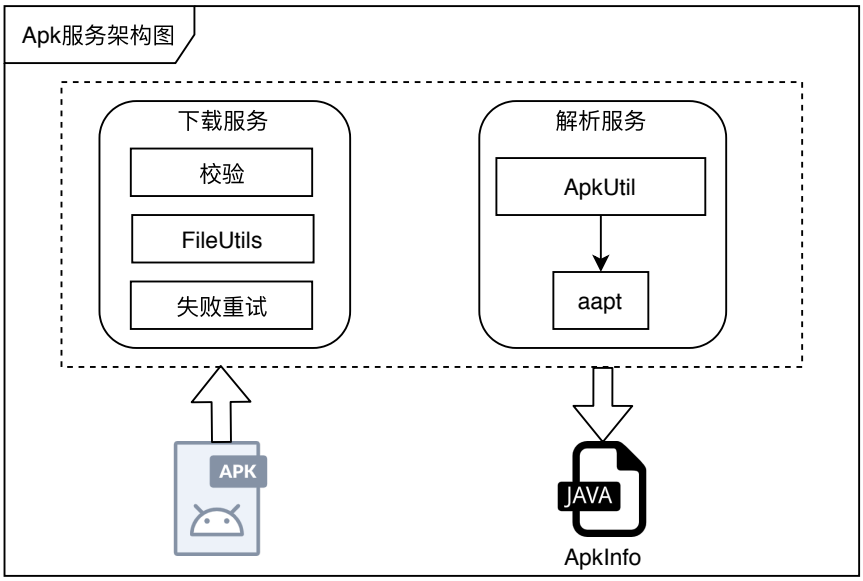

图 4.7: APK 服务-架构图

4.2.2　APK 服务核心类图

APK 服务的核心类图设计如图 4.8所示。AutoTestService 为 APK 服务的调度方自动化测试服务。ApkService 为 APK 服务对外提供的接口类，主要提供了下载 APK、解析 APK、保存 APK 信息三个接口。ApkServiceImpl 为 ApkService 的实现类，负责实现上述三个功能。OssService 则为 ApkService 提供 OSS 存储及上传服务，其主要为 APK 服务提供上传 APK 图标的功能。ApkUtil 是解析 APK 文件到 ApkInfo 类的工具类，其通过调用 aapt 子进程以及模式匹配，构造 ApkInfo 对象。

4.2.3　APK 服务顺序图

APK 服务的顺序图如图 4.9所示。自动化测试服务在收到自动化测试任务，并创建任务线程之后，任务线程会调用 APK 服务提供的下载接口进行 APK 下载，下载首先会进行下载 URL 的校验，校验成功后，通过 FileUtils 工具类进行下载。如果下载失败，则会至多尝试 10 次，如果均失败，则拒绝该任务。下载成功后，则会调用 APK 服务提供的解析接口进行 APK 文件解析。APK 服务解析接口则会调用 ApkUtil 工具类执行具体的解析工作。在 ApkUtil 工具类中，是通过启动 aapt 工具子进程，使用 badging 解包命令获取到 APK 文件的各种详细信息。获取到 aapt 子进程的结果字符串后，通过模式匹配构造 ApkInfo 对象，返回给 APK 服务，APK 服务继续返回给自动化测试服务。收到正确的 ApkInfo 对

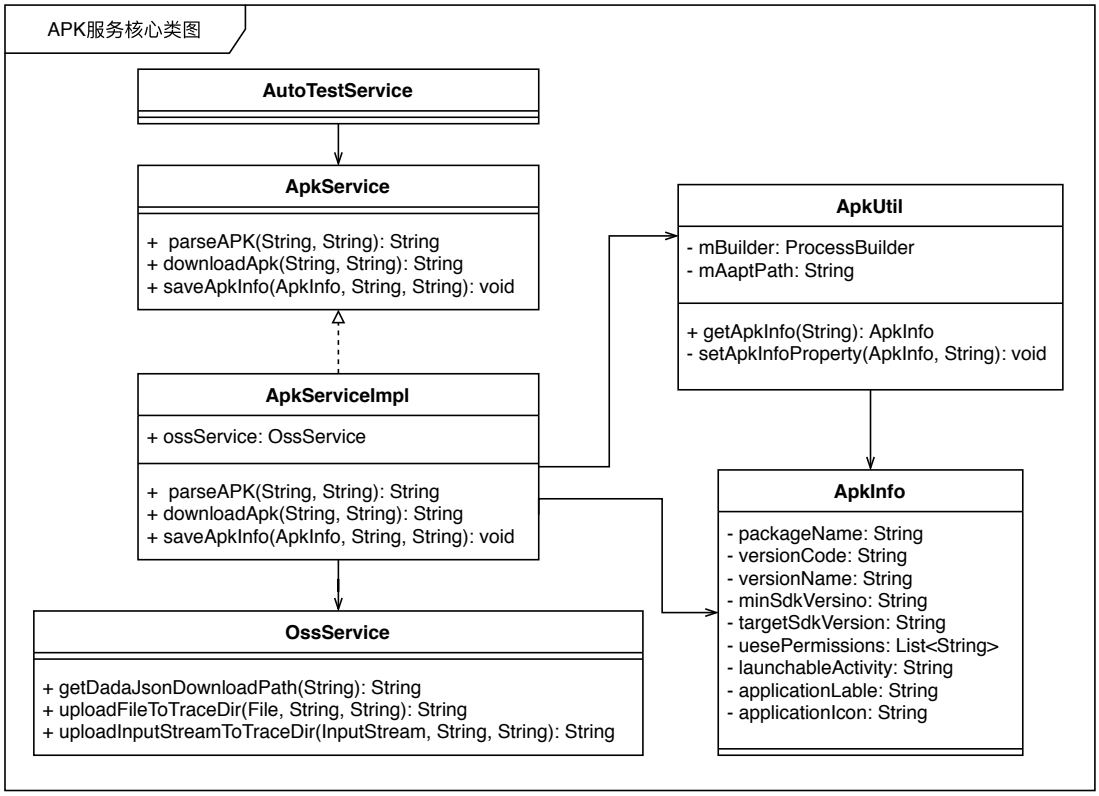

图 4.8: APK 服务-核心类图

象后，自动化测试服务则会调用 APK 服务的保存接口，将 ApkInfo 相关信息保存到对应的文件中。

4.2.4　APK 服务关键代码

图 4.10为 APK 服务解析服务的核心代码实现。其主要涉及了 ApkServiceImpl 及 ApkUtil 两个类。为方便阅读，故将两个类中的核心实现代码放在同一张图中展示。

ApkServiceImpl 的 parseApk 方法是对 ApkUtil.getApkInfo() 方法的简单封装，是为了处理各类异常情况的出现。APK 解析的核心代码为 ApkUtil 中的 getApkInfo() 方法。在该方法中，会通过 ProcessBuilder 构建一个子进程，执行 “aapt d badging apkPath” 命令，获取 APK 的静态信息。然后通过读取子进程的返回信息，按序处理每一行的数据，调用 setApkInfoProperty() 方法，该方法通过模式匹配以及正则匹配，获取 Apk 信息，并构建 ApkInfo 对象。每一行信息处理完成后，返回 ApkInfo 对象。

图 4.11为 APK 服务下载功能的核心实现代码。首先，需要检查下载链接的有效性与合法性，如下载链接无效，则会抛出 APK 下载失败异常。之后则

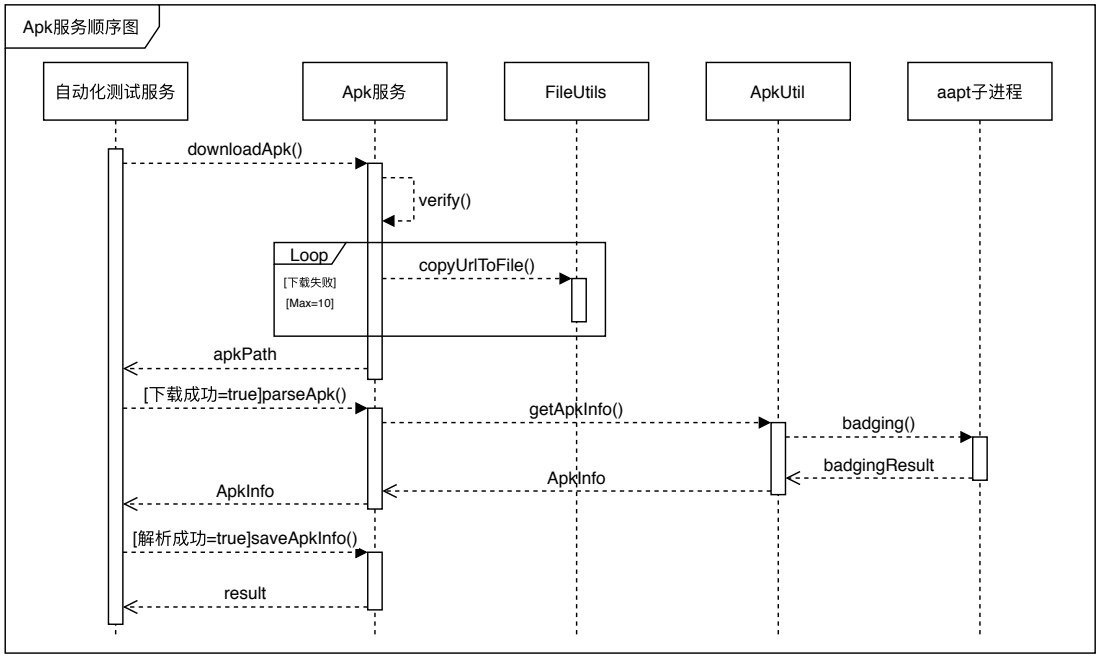

图 4.9: APK 服务-顺序图

会获取 APK 文件的存储路径。然后通过 FileUtils 工具类的 copyURLtoFile 方法进行下载。如果下载成功，则直接返回 APK 的本地路径；如果失败，则会重试 DOWNLOAD_MAX_ATTEMPTS（默认为 10）次，如均下载失败，则会抛出 APK 下载失败异常。

4.3　设备服务的设计与实现

4.3.1　设备服务架构设计

设备服务为本系统中进行设备管理的服务，其架构图如图 4.12所示。该服务主要提供获取当前连接设备、设备选取、设备状态维护等功能。获取当前连接设备需要使用安卓开发套件中的 adb 工具，通过 OsUtil 工具类执行 adb 命令获取。设备筛选则是将用户选择的设备与当前在线且空闲的设备取其交集，作为本次任务的执行设备。

另外，设备状态维护需要维护如图 4.13所示的设备的有限状态机，因为自动化测试设备为有限资源，存在多线程间的同步问题，所以需要通过 Lua 脚本，执行 CAS 操作，在 Redis 数据库中维护每个设备的占用状态；对于设备的执行状态，并无多线程同步的需求，所以可以直接通过 RedisMapper，在 Redis 数据库中进行维护。

```
// ApkServiceImpl
@Override
public ApkInfo parseAPK(String apkPath, String traceId) {
    try {
        return new ApkUtil().getApkInfo(apkPath);
    } catch (Exception e) { throw new ApkParseException(e.getMessage()) };
}

// ApkUitl
public ApkInfo getApkInfo(String apkPath) throws Exception {
    Process process = this.mBuilder.command(this.mAaptPath, "d", "badging", apkPath).start();
    BufferedReader br = new BufferedReader(
                                new InputStreamReader(process.getInputStream();, "utf8"));
    String tmp = br.readLine();
    try {
        if (tmp == null || !tmp.startsWith("package")) {
            throw new Exception("参数不正确，无法正常解析 APK 包。");
        } else {
            ApkInfo apkInfo = new ApkInfo();
            do {
                this.setApkInfoProperty(apkInfo, tmp);
            } while((tmp = br.readLine()) != null);
            Return apkInfo;
        }
    } catch (Exception e) {}
    finally {
        // 省略收尾操作代码
    }
}
```

图 4.10: APK 服务-解析的实现

4.3.2　设备服务核心类图

设备服务的核心类图如图 4.14所示。AutoTestService 为设备服务的主要调度方。DeviceService 为设备服务对外提供的接口。主要提供了获取当前在线设备、筛选设备、更新/获取设备执行状态、CAS 更新设备占用状态等抽象接口。该接口还是用到了 Device 类，该类为系统中设备的抽象对象。DeviceServiceImpl 为 DeviceService 的实现类，负责实现接口对外暴露的功能。该实现类主要通过 DeviceRunningMap 类实现对设备执行状态的维护；通过 DeviceStatusMap 类实现对设备占用状态的维护；通过 OsUtil 工具类实现执行 adb 命令。最后，DeviceStatus 为设备占用状态的枚举类，其包含空闲、执行中、被占用三个枚举值，代表设备

```
// 下载服务
public String downloadApk(String downloadUrl, String traceId) {
    if (!verifyDownloadUrl(downloadUrl)){
        throw new ApkDownloadException("Download APK failed because this is not an apk");
    }
    String path = getApkPath(downloadUrl, traceId);
    int downloadTimes = 0;
    while (downloadTimes < Consts.DOWNLOAD_MAX_ATTEMPTS) {
        try {
            FileUtils.copyURLToFile(new URL(downloadUrl), new File(path));
            return path;
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 省略下载失败，sleep 10s 的代码
        downloadTimes ++;
    }
    // 抛出下载失败异常
}
```

图 4.11: APK 服务-下载的实现

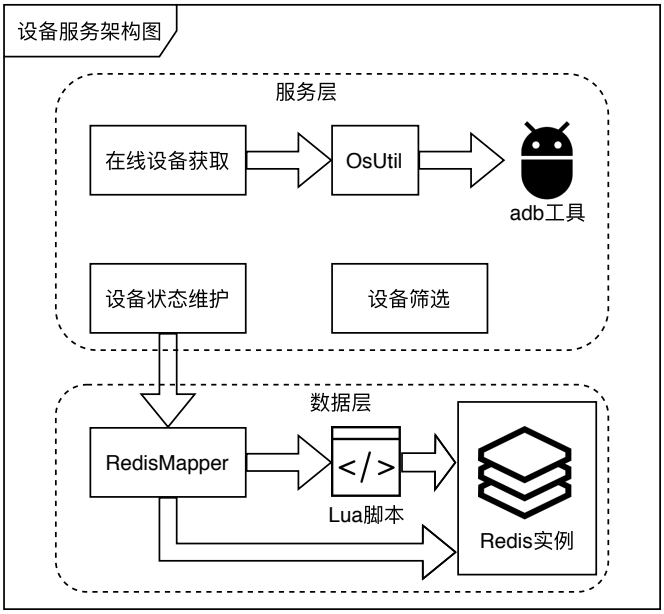

图 4.12: 设备服务-架构图

的三种占用状态。DeviceRunningStatus 为设备执行状态的枚举类，其包含初始化、准备中、执行中、结束中、完成、等待中六个枚举值，分别代表一台设备在执行一次自动化测试任务过程中的各个阶段。

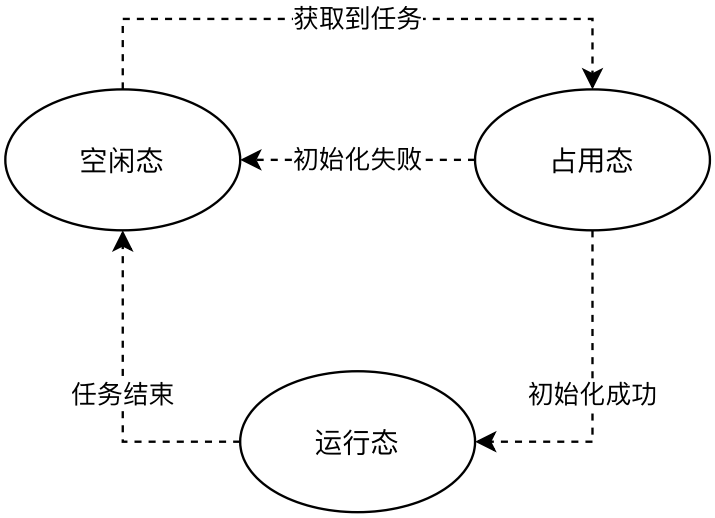

图 4.13: 设备服务-设备有限状态机

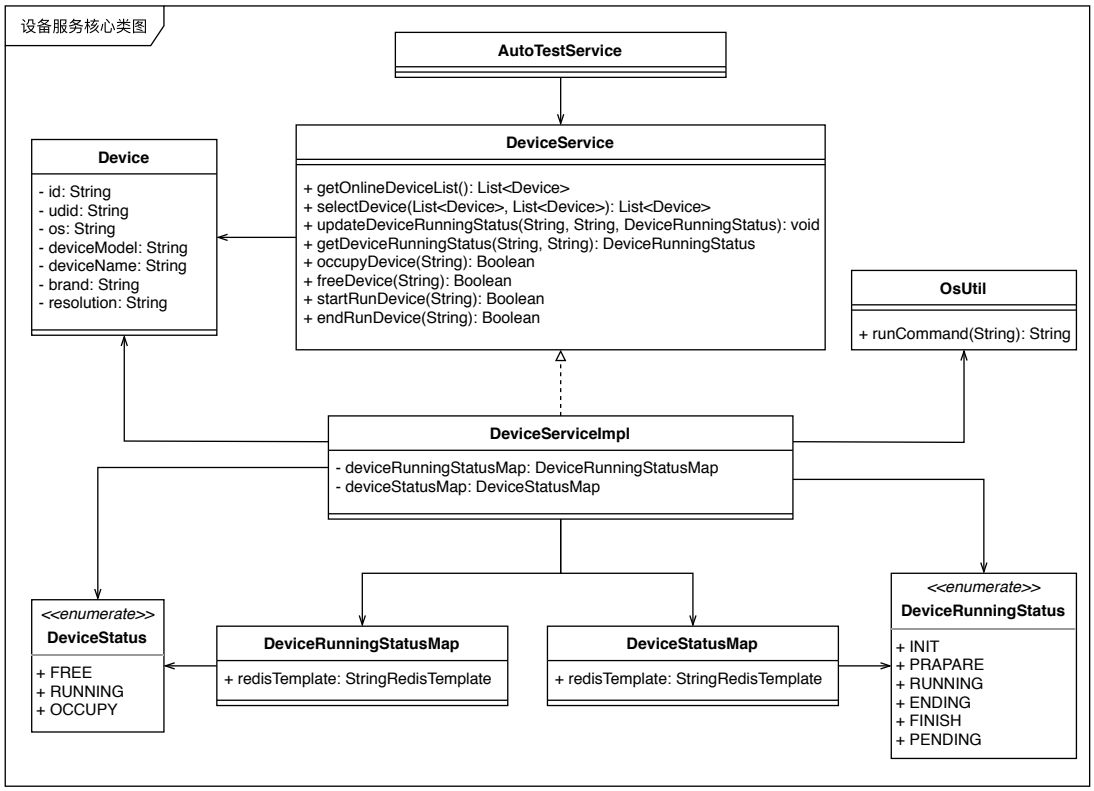

图 4.14: 设备服务-核心类图

4.3.3　设备服务顺序图

设备服务的顺序图如图 4.15所示。在自动化测试任务刚刚创建的时候，自动化测试服务会向设备服务发送获取当前在线设备的请求，然后设备服务通过

OsUtil 工具类执行 “adb devices” 命令，获取当前在线的设备。之后，再通过 selectDevices() 方法筛选出可用的设备用于执行自动化测试任务。

在任务执行的过程中，每台设备的执行会经历若干个阶段，自动化测试服务则会向设备服务发送更新/获取设备执行状态的请求，设备服务则会使用 DeviceRunningStatusMap 类的 get() 方法，通过 RedisTemplate 类与 Redis 实例通信，获取或更新设备执行状态。当自动化测试即将开始或即将结束时，需要占用或释放设备，此时自动化测试服务则会发送 updateDeviceStatus() 请求，更新设备状态。由于此处存在多线程之间同步的问题，所以采用 CAS 乐观锁来更新设备占用状态。DeviceStatusMap 收到设备服务的 CompareAndSet 类请求时，则会使用 RedisTemplate 的 execute 方法执行 Lua 脚本，CAS 更新设备状态。

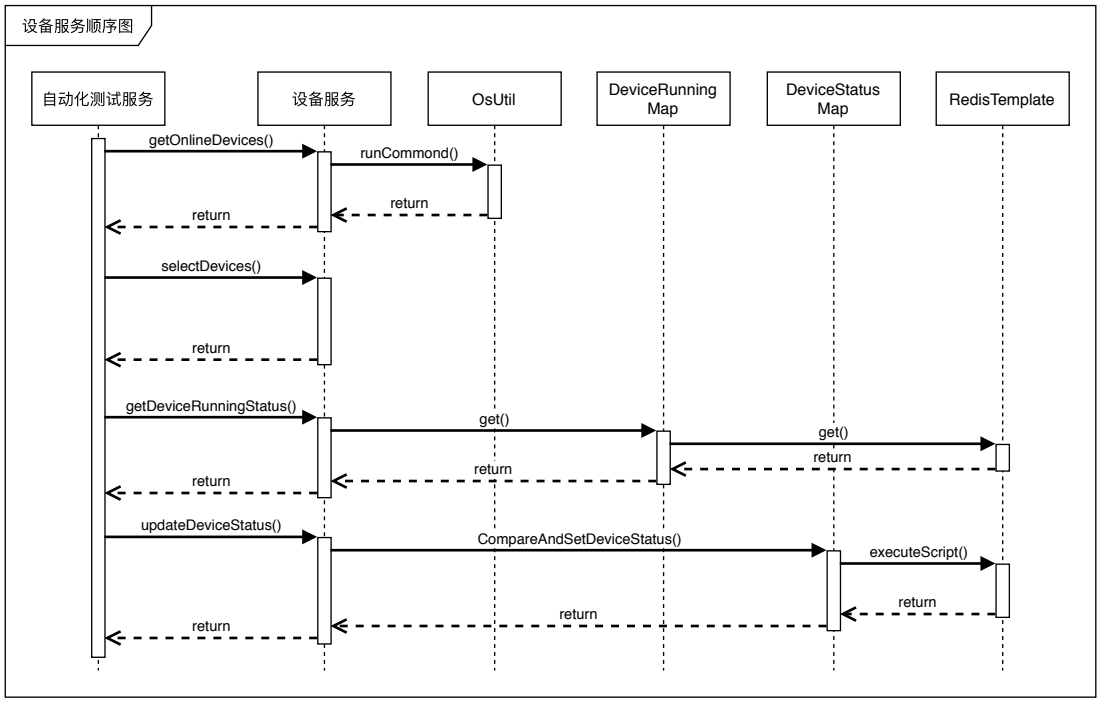

图 4.15: 设备服务-顺序图

4.3.4　设备服务关键代码

图 4.16为设备服务获取在线设备的关键代码。首先，设备服务通过 OsUtil 工具类执行 “adb devices” 命令，获取到当前在线的所有设备 ID，然后逐行通过模式匹配获取并组装 Device 对象，最后返回给调用方。

图 4.17为设备服务 CAS 更新设备占用状态的关键代码。首先，该方法从 resources 资源目录中获取到对应的 LUA 脚本的内容，之后通过 DefaultRedisScript

```
@Overrid
public List<Device> getOnlineDeviceList() {
    // 执行 adb devices 命令并获取结果
    String command = "adb devices";
    String comResult = OsUtil.runCommand(command);
    String[] comResultLine = comResult.split("\n");
    List<Device> devices = new ArrayList<>(comResultLine.length);
    // 逐行处理结果
    for (String line:comResultLine){
        line = line.trim();
        if ("".equals(line) || line.length()==0 || line.startsWith("List")){
            continue;
        }
        // 此处省略模式匹配与组装 Device 对象代码
    }
    return devices;
}
```

图 4.16: 设备服务-获取在线设备的实现

构造方法，构造出 RedisSrcipt 对象，最后通过 redisTemplate 对象的 execute() 方法执行如图 4.18所示的 LUA 脚本，获取到返回值。如果返回值为 1，则说明更新成功，返回 true；否则说明更新失败，返回 false。

```
public boolean compareAndSet(String udid, DeviceStatus oldStatus, DeviceStatus newStatus){
    try {
        // 读取 Lua 脚本
        String scriptContents = IOUtils.toString(
                getClass().getResourceAsStream("/" + Consts.CAS_SCRIPT), "UTF-8");
        // 创建 RedisScript 对象
        RedisScript<Long> script = new DefaultRedisScript<>(scriptContents, Long.class);
        // 执行 LUA 脚本
        Long result = redisTemplate.execute(script,
                Collections.singletonList(DEVICE_STATUS_PREFIX + udid),
                oldStatus.getCode(), newStatus.getCode());
        return result == 1;
    } catch (IOException e){
        return false;
    }
}
```

图 4.17: 设备服务-更新设备占用状态的实现

图 4.18为设备服务 CAS 更新设备占用状态的 LUA 脚本。此脚本主要负责处理更新设备占用状态信息。此脚本的输入参数为 Key 值 KEYS[1], 旧值 ARGV[1] 与新值 ARGV[2]。此处存在三种情况。（1）如果需要 CAS 更新的 Key 值不存在，则直接将 Key 值设置为新值 ARGV[2]，并返回 1；（2）如果 Key 值存在，但是当前的值与旧值不同，这则说明 Key 已经被修改过，则放弃更新，返回 0；（3）如果当前的值与旧值相等，则可以将 Key 值设置为新值 ARGV[2]，并返回 1。

```
-- key 不存在，赋值 key
if redis.call('EXISTS', KEYS[1]) == 0 then
    redis.call('SET', KEYS[1], ARGV[2])
    return 1
end

-- key 存在，但与旧值不相等，不进行赋值
if redis.call('GET', KEYS[1]) ~= ARGV[1] then
    return 0
end

-- key 存在，且与旧值相等，进行赋值
redis.call('SET', KEYS[1], ARGV[2])
return 1
```

图 4.18: 设备服务-LUA 脚本的实现

4.4　测试执行服务的设计与实现

4.4.1　测试执行服务架构图

图 4.19为本系统测试执行服务的架构图。测试执行服务所针对的是每一台设备上的操作执行，主要负责每一台设备上操作流程的控制以及保障整个流程的通畅。在进行自动化遍历测试之前，测试执行服务会通过 ADB 工具先执行一系列基础测试，包括 APK 文件安装、卸载、覆盖安装、冷启动等，以确保 APK 的正常。之后，会分别启动 Logcat 监控服务、设备截屏与设备状态监控服务，然后执行具体的自动化测试脚本。自动化测试脚本的执行，主要是通过 Appium 服务将操作指令传送给安卓手机来完成的。在脚本执行完毕之后，测试执行服务还会执行一些收尾操作以保证安卓设备环境的干净。

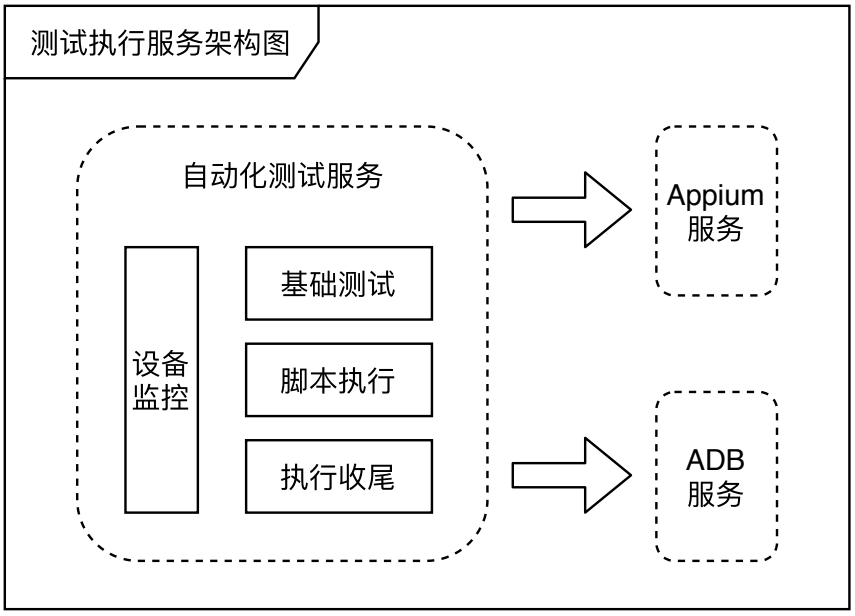

图 4.19: 测试执行服务-架构图

4.4.2　测试执行服务核心类图

图 4.20为测试执行服务的核心类图。CoverageTest 类为测试执行服务的核心类，其控制和掌握着单台设备上自动化测试的流程。DefaultScript 为系统提供的自动化执行的默认脚本，其基于深度优先遍历算法对 APP 进行自动化遍历测试。AbstarctScript 类为默认脚本的抽象类，其抽象了 stopFlag 停止标记位与一系列模板方法，通过 AbstarctScript 抽象类，可以尽可能地使各类脚本统一，方便执行。ScriptExecutor 为自定义脚本的执行类，用于执行用于自定义的测试脚本。Operation 类则抽象了自动化遍历过程中的一些基本操作，如点击、输入、滑动等，用于给默认脚本提供设备操作的抽象。

4.4.3　测试执行服务顺序图

图 4.21为测试执行服务的顺序图。当测试执行服务受到自动化测试服务的 start 指令时，会先启动 Logcat 监控线程，用于监控安卓手机的 Logcat 日志。之后会进行一系列的基础测试工作。基础测试完成后，则会启动截屏以及设备状态监控线程。然后会根据用户是否使用自定义脚本，调用默认脚本的 start() 方法或是脚本执行类 ScriptExecutor 的 executeScript() 方法。此调用过程为同步过程，测试脚本会在出现异常情况或者测试完成后退出。最后，则是进行收尾操作，用于保障安卓设备环境的干净。

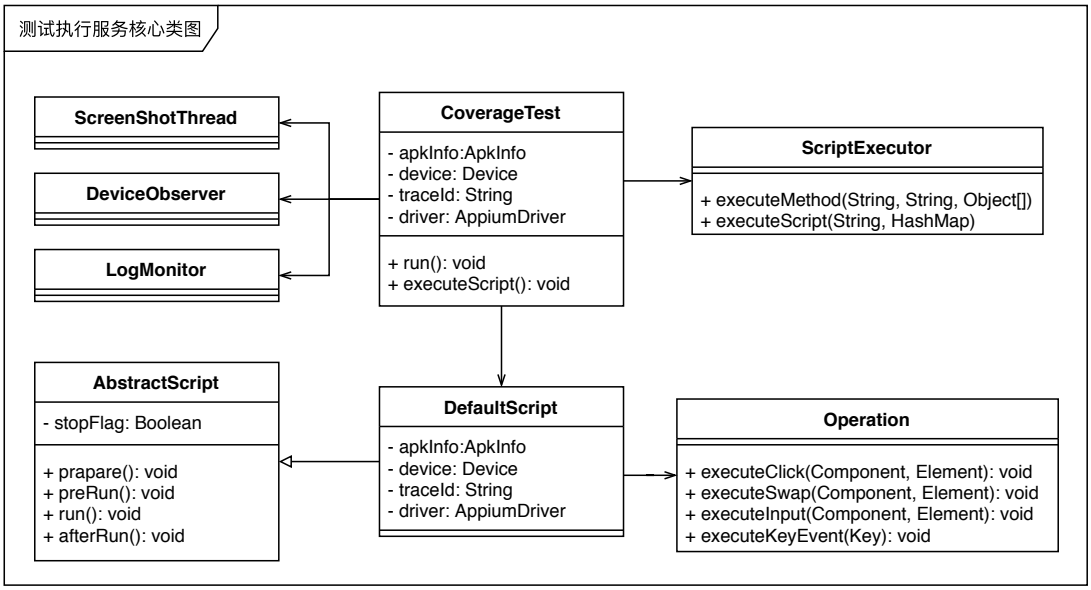

图 4.20: 测试执行服务-核心类图

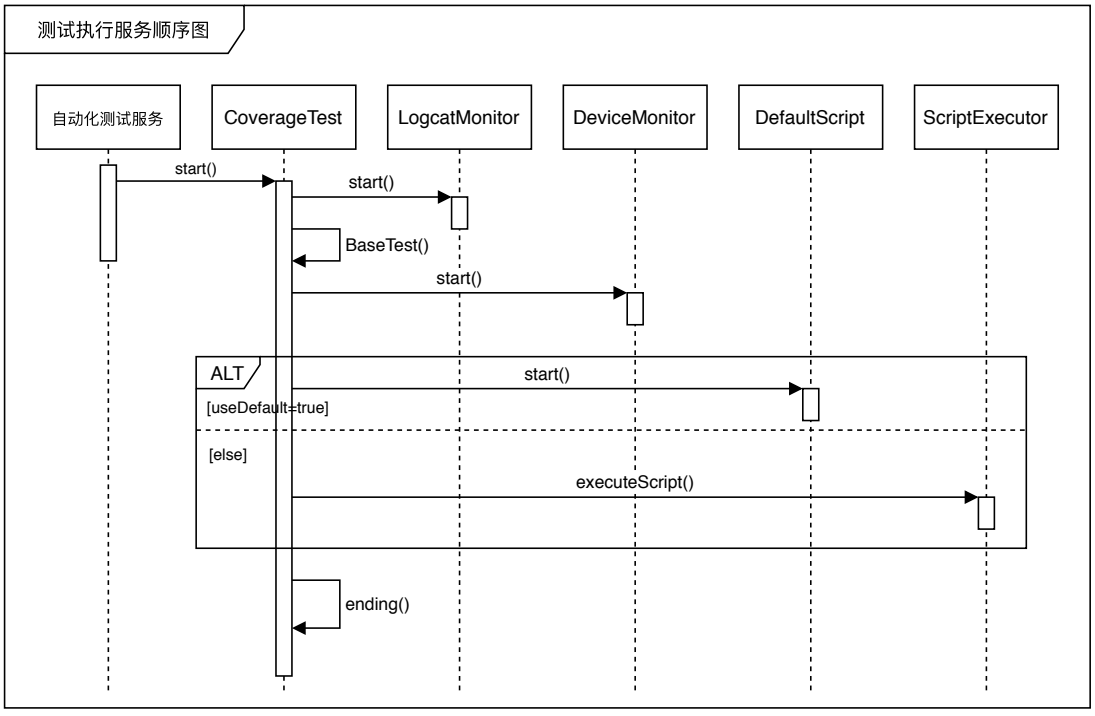

图 4.21: 测试执行服务-顺序图

4.4.4　测试执行服务关键代码

图 4.22为测试执行服务的关键代码实现。测试执行服务的主要功能为控制单台设备上自动化测试的流程。当接收到需要在一台安卓设备上执行自动化测

试的请求时，测试执行服务首先需要启动 AppiumServer 以及 Logcat 的监控线程，之后会先进行一系列的基础测试，包括安装、覆盖安装、冷启动和卸载。然后会初始化 AppiumDriver 用于和安卓设备通信，初始化完成后，在设备端应用就会被启动，所以之后需要紧接着启动截屏线程与设备状态监控线程。上述所有操作正常完成后，便会调用具体的遍历流程算法，进行自动化的遍历操作。在遍历算法结束后，还需要做一些额外的收尾操作，包括恢复输入法、保存设备的执行信息、退出 APP、停止一系列监控线程与 AppiumServer、重置该台设备的 ADB 连接等，最后将设备的状态通过 CAS 更新为 FREE。整个测试调度流程即结束。

```
@Override
public void run() {
    try{
        startAppiumServerAndLogcat();  // 启动 appium server 和 logcat 监控线程
        // 省略基础测试代码：安装、覆盖安装、冷启动、卸载
        initDriver();/   /初始化 Appium driver
        startScreenShotAndDeviceObserverThread();//开启截屏&监控设备状态线程
        executeScript();  // 执行算法逻辑
    } catch (Exception e) {}
    finally {
        // 省略收尾操作代码：恢复输入法、保存执行信息、停止 APP，停止监控线程
        stopAppiumServer(this.port);  // 停止 appium server
        reconnectDeviceAdb();  // 重新连接设备的 adb
        deviceService.endRunDevice(udid);  // 恢复设备占用状态
        finalEnd = true;
    }
}
```

图 4.22: 测试执行服务-外部框架实现

图 4.23为测试执行服务 DFS 算法的核心代码。每次 DFS 算法的调用，都意味着应用进入了一个全新的页面，所以首先需要处理各类弹窗以及安卓的权限申请；之后则会通过 AppiumDriver 获取到当前页面的组件列表 ComponentList，通过组件列表判断当前是否还处于待测 APP 中；然后遍历组件列表中的所有组件并执行相应操作；最后当组件列表遍历结束后，则将该页面的状态设置为 Done，并返回上一级页面。

DFS 内部组件遍历的核心代码如图 4.24所示。首先获取到当前需要遍历的组件对象，之后检查该组件是否已被测试过或者被配置跳过；然后根据组件的类型执行具体的操作，如果组件是 EditText 输入框，则执行输入操作，如果是

```
private void DFS(String fatherActivity, int depth){
    handleAndroidAlertOnLaunch();
    List<Component> componentList = getCurrentClickableComList(depth, true);
    if (isOutOfAppPackage(componentList, depth)){
        this.endScript();
        return;
    } else if (driver.currentActivity().equals(fatherActivity)){
        return;
    }
    // 省略更新页面 Activtity 与判断 Activity 是否结束的代码
    for(Component c: componentList){
        // 省略遍历所有组件并点击的代码
    }
    activityList.get(index).setDone(true);
    // 返回上一页
    returnPrevActivity();
}
```

图 4.23: 测试执行服务-DFS 的实现

其他组件类型，则执行点击操作。在执行点击操作之后，则需要判断点击操作前后的页面状态是否一致，（1）如果页面完全没有变化，则继续遍历下一个组件；（2）如果还处于同一个页面但是组件变化了，则更新组件列表并继续遍历下一个组件；（3）如果页面发生了跳转，则通过调用如图 4.23所示的 DFS 算法进行更深一层的调用，遍历新的页面。当新的页面遍历完成并返回了当前页面后，则需要处理下弹窗以及页面不属于待测 APP 的情况，最后更新一下组件列表，继续下一个组件的遍历。

图 4.25为测试执行服务执行用于自定义测试脚本的核心实现。本系统共提供了两种执行策略，对于拥有完整结构的 Groovy 脚本，首先通过 GroovyScriptEngine 加载并编译脚本文件，最后通过反射调用指定的方法并传递参数。对于没有完整结构之后代码片段的 Groovy 脚本，首先需要通过 Binding 进行参数组装与绑定，然后通过 GroovyScriptEngine 加载并编译脚本文件，最后使用 engine 的 run() 方法执行脚本，获取结果。

```
// 遍历组件并点击的代码
for (int comIndex=0;comIndex<componentList.size();++comIndex){
    Component component = componentList.get(comIndex);
    // 省略检查组件状态的代码，省略定位组件的代码
    if (component.getClassname().contains("EditText")){
        executeInput(component, elementForTest);
    }else {
        //其他控件，则点击
        executeClick(component, elementForTest);
        String activityAfterClick = driver.currentActivity();
        int pageHashAfterClick = driver.getPageSource().hashCode();
        // 页面完全没变化
        if (pageHashAfterClick == currentHash){
            continue;
        }
        // 页面未变，组件变了
        if (activityAfterClick.equals(currentActivityName)){
            // 省略更新组件列表的代码
            continue;
        }
        // 页面 Activity 变化了，遍历新的页面
        DFS(currentActivityName, depth+1);
        // 省略处理弹窗和不在 app 中的代码
        // 省略更新组件列表的代码
    }
}
```

图 4.24: 测试执行服务-DFS 组件遍历的实现

4.5　监控服务的设计与实现

4.5.1　监控服务架构设计

本系统监控服务的架构图如图 4.26所示。监控服务共分为两部分，一是任务监控，其负责对任务整体进度的把控；二是设备监控，其负责在自动化测试执行期间实时获取每台安卓设备的各项数据、截屏及日志，然后将数据生成格式化日志储存在文件系统中。

在系统收到自动化测试任务后，会将测试任务分发到安卓设备上。同时，将任务的执行信息发送给任务监控服务，并启动任务监控。任务监控启动后，会每 5 秒检查所有安卓设备的执行状态，如果所有设备均已执行完成或是任务已

```
public static void executeMethod(String scriptName, String methodName, Object[] objects){
    try {
        // 构造 GroovyObject 对象
        GroovyScriptEngine scriptEngine = new GroovyScriptEngine("");
        Class c = scriptEngine.loadScriptByName("scripts/" + scriptName);
        GroovyObject groovyObj = (GroovyObject) c.newInstance();
        //调用成员方法
        groovyObj.invokeMethod(methodName, objects);
    }catch (Exception e){ }
}

public static void executeScript(String scriptName, HashMap<String, Object> params){
    // 组装参数
    Binding binding = new Binding();
    for (String key : params.keySet()) {
        binding.setVariable(key, params.get(key));
    }
    try {
        // 调用脚本
        GroovyScriptEngine engine = new GroovyScriptEngine("");
        Object result = engine.run("scripts/" + scriptName + ".groovy", binding);
        System.out.println(result);
    } catch (Exception e) {}
}
```

图 4.25: 测试执行服务-执行自定义脚本的实现

经超时，那么任务监控则会终止所有任务，并根据创建时的参数调用相应报告生成器，生成并上传相应的测试报告。同时，在每台设备执行自动化测试前，也会分别启动设备监控服务，用于收集该设备的 Logcat 日志、截屏、CPU、内存、屏幕刷新率、网络、电池温度等信息，并用这些信息生成结构化的日志，供各类报告生成器使用。

4.5.2　监控服务核心类图

监控服务的核心类图如图 4.27所示。其中，TraceDaemon 为任务监控服务类，其主要负责对所有设备执行的线程 CoverageTest 进行监控，同时在所有设备执行线程结束后更新任务状态，并根据相应参数生成并上传报告。OssService 所提供的是报告上传服务，TraceService 所提供的是更新任务状态的服务。CoverageTest 为设备执行的线程，在其中主要通过 ScreenShotThread 类实现对设备的截屏操作；通过 LogMonitor 类实现对设备 Logcat 日志的收集与解析；通过 De-

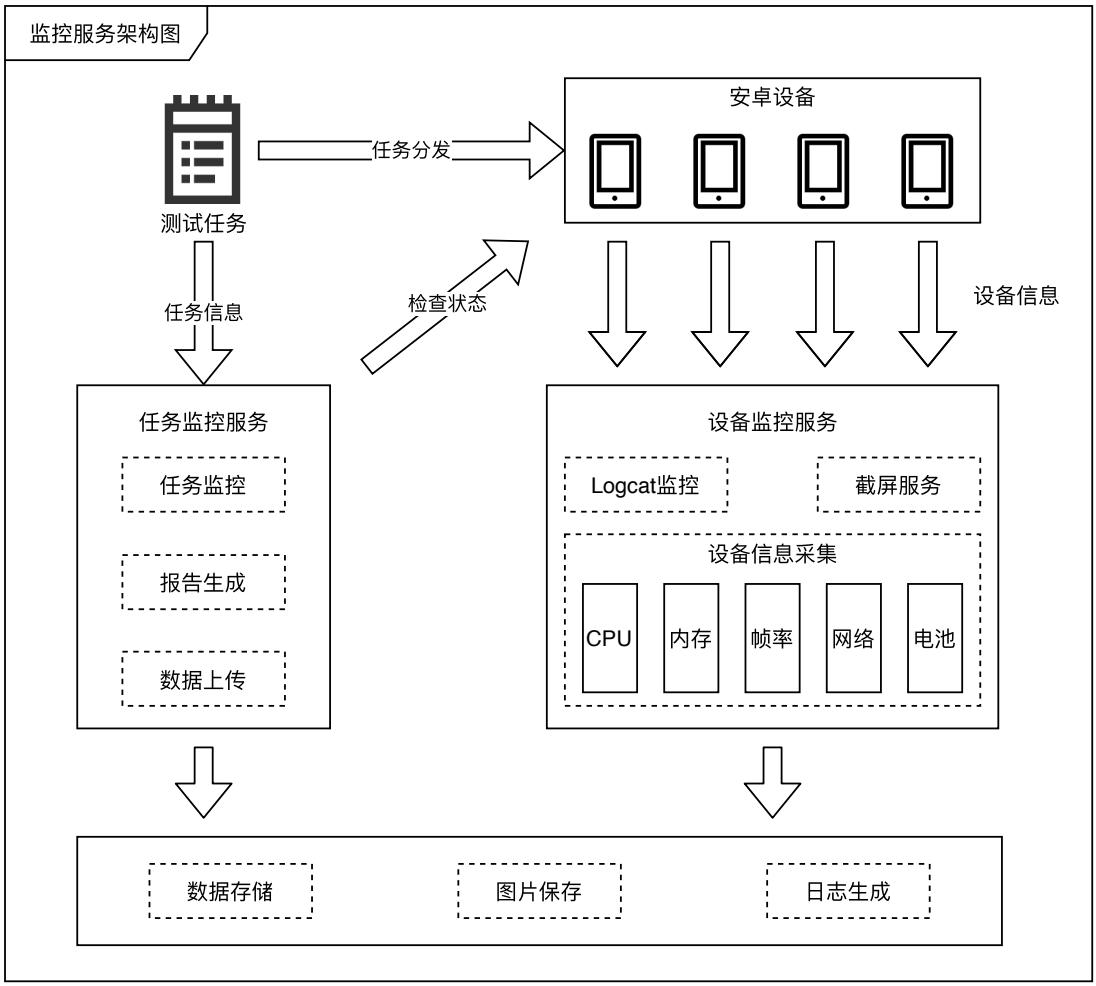

图 4.26: 监控服务-架构图

viceObserver 类实现对设备各种资源状态的收集；通过 SMMonitor 实现对设备流畅度的收集与计算。最后需要说明的是，TraceDaemon、CoverageTest、ScreenShotTrhread、DeviceObserver、LogMonitor 均继承于 Thread 类，为使类图简单清晰，本图并未画出相关继承关系。

4.5.3　监控服务顺序图

设备监控服务的顺序图如图 4.28所示，上下分别为任务监控的顺序图与设备监控的顺序图。

对于任务监控来说，当任务监控线程被创建之后，则会每五秒检查该任务是否超时，或者是否所有的设备执行线程已结束。如任务超时或所有设备执行线程已结束，则任务监控线程会给所有设备执行线程发送结束与 join 指令，此操作是为了等待设备执行线程完成各自收尾操作。之后监控线程会更新任务状

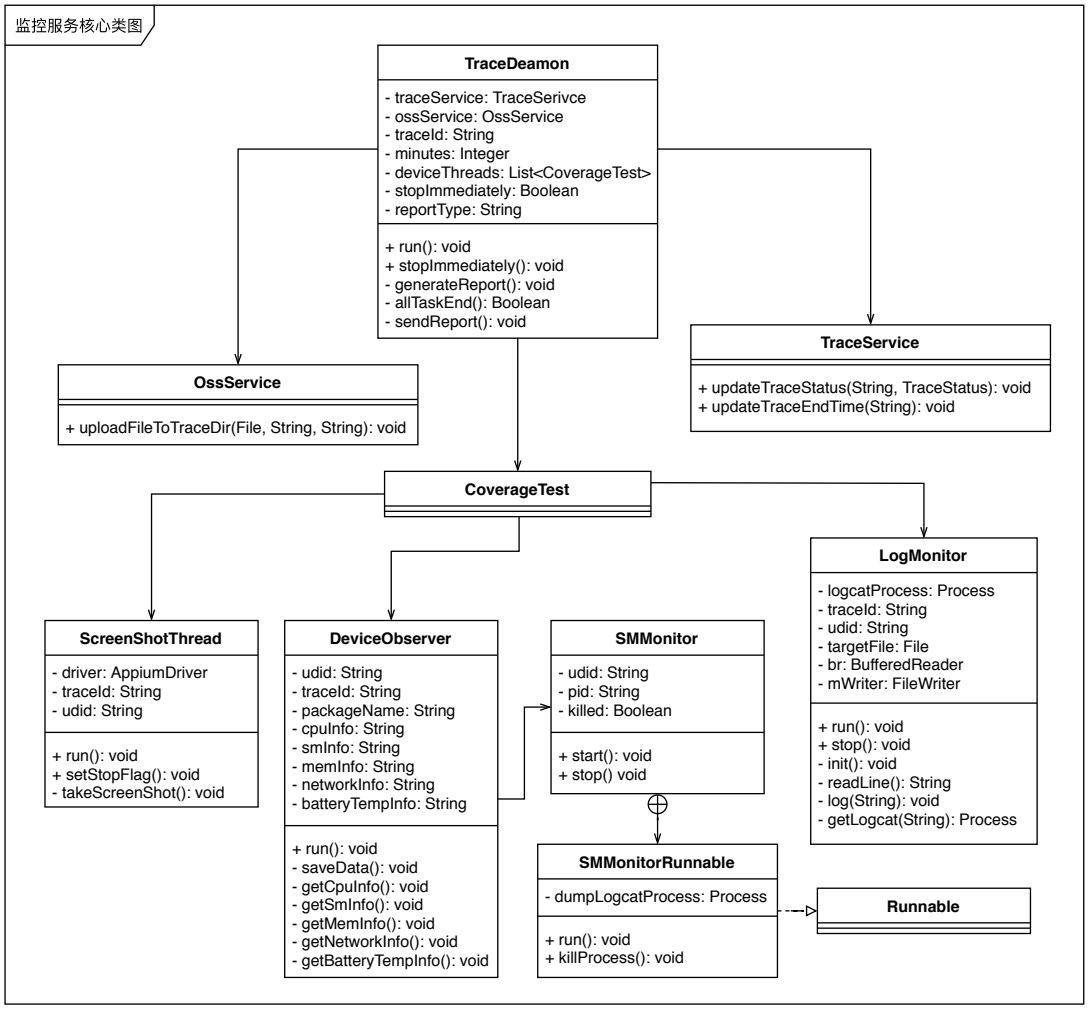

图 4.27: 监控服务-核心类图

态为报告生成中，然后根据配置生成对应的测试报告，最后通过 OssService 将生成的测试报告上传至阿里云 OSS 服务。

对于设备执行监控来说，在一个测试任务即将执行前，会分别启动 Logcat 收集线程、截屏收集线程、设备信息收集线程。这三个线程均在各自线程中通过死循环定时获取各类信息，并将其结构化的存入相应的日志文件，直至收到 end() 请求，才会结束死循环。

4.5.4　监控服务关键代码

图 4.29为任务监控服务的关键代码。在异步启动任务监控线程之后，首先会进行元数据初始化，之后每隔 TRACE_CHECK_TERMINAL（默认为 5）秒执行一次循环，如果超时或者收到了停止请求，则终止循环。在循环体内，检查是

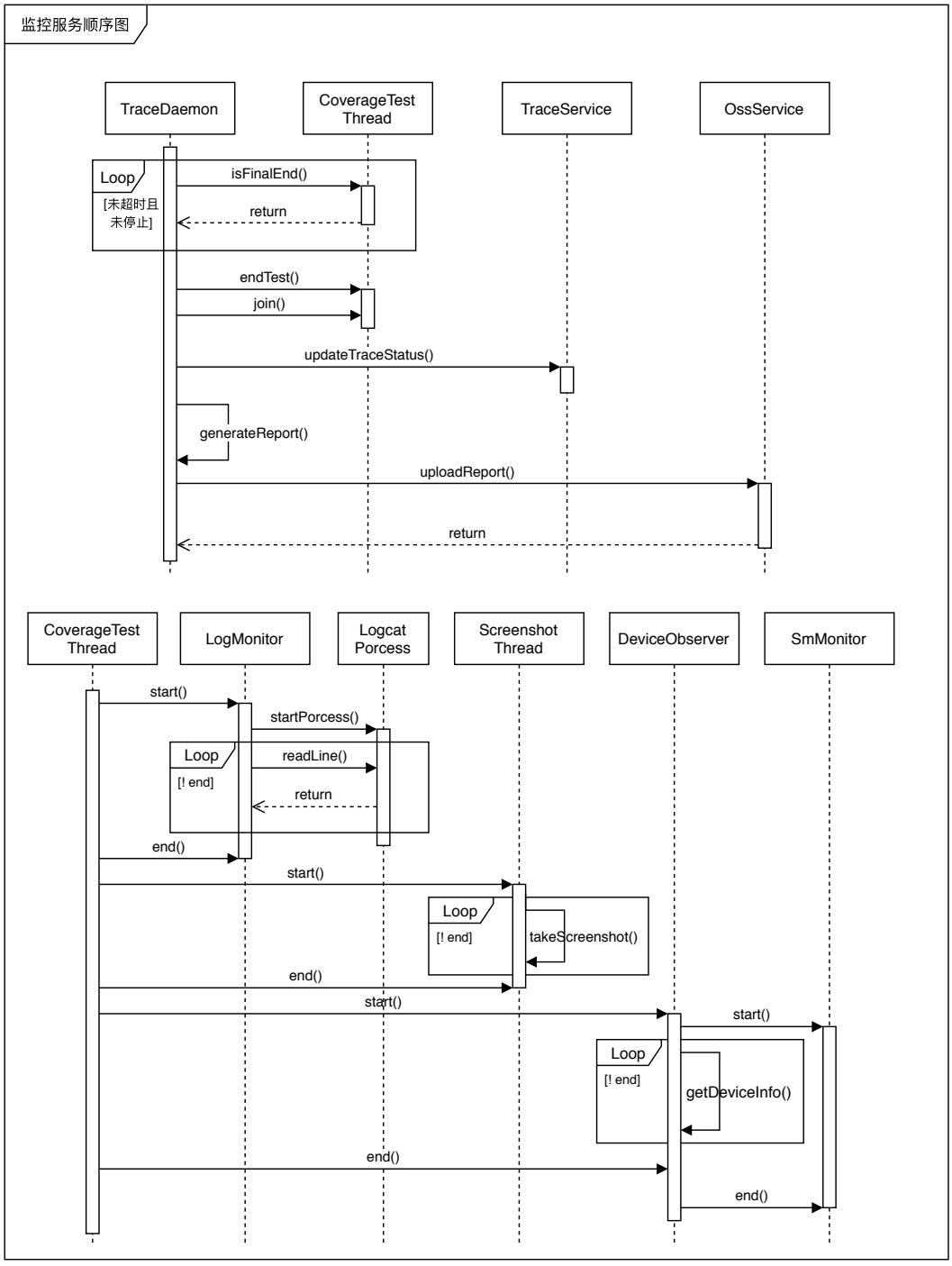

图 4.28: 监控服务-顺序图

否所有设备执行线程已经结束，如果均已结束，则跳出循环体。循环结束后，分别调用所有设备执行线程的 endTest() 方法和 join() 方法，等待所有设备执行线程结束。之后再进行状态更新、报告生成与报告上传的工作。

```
@Override
public void run() {
    // 此处省略初始化各项数据代码
    // 每 checkTerminal 秒检查一次所有线程状态
    while (sleptTime <= timeoutTime && !stopImmediately){
        try {
            Thread.sleep(Consts.TRACE_CHECK_TERMINAL * 1000);
            if (allTaskEnd = allTaskEnd()){
                break;
            }
        } catch (InterruptedException e) {}
        sleptTime += Consts.TRACE_CHECK_TERMINAL * 1000;
    }
    try {
        for (CoverageTest coverageTest: deviceThreads){
            coverageTest.endTest();
        }
        // 等待设备执行线程结束
        for (CoverageTest coverageTest: deviceThreads){
            coverageTest.join();
        }
    } catch (InterruptedException e) {}
    finally {
        // 省略生成、上传报告，更新状态的代码
    }
}
```

图 4.29: 监控服务-任务监控的实现

图 4.30为设备截屏服务的关键代码。在循环体内，每经过 2 秒，或通过 AppiumDriver 的 getScreenshotAs() 方法截屏并获取截屏文件，之后通过 FileUtils 将截屏文件保存到文件系统的指定目录中，供报告生成使用。

图 4.31为获取设备 Logcat 日志的关键代码。首先执行 “adb logcat -b main -b system -b crash -b events -b radio -v time” 命令，构造出 logcat 的进程对象并执行，然后获取到子进程输出流。之后读取输出流的每一行日志，通过正则匹配将每一行日志格式化，最后存入指定路径的 Logcat 日志文件中。

图 4.32为获取设备当前流畅度的关键代码。首先，构造子进程并执行 “adb shell logcat -v time Choreographer:I *:S” 命令，之后获取子进程的输出流，并对输出流进行模式匹配，最后获取到当前设备的流畅度信息。

图 4.33为获取设备当前执行状态信息的关键代码。首先，获取设备执行信

```
private void takeScreenShot(AppiumDriver driver,String udid) {
    while( !isEnd ){
        try {
            File screenShotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
            String path = getScreenShotFilePath(traceId, udid);
            FileUtils.copyFile(screenShotFile, new File(path));
        } catch (WebDriverException e){
            // 异常处理
        } finally {
            // sleep 2 秒钟
        }
    }
}
```

图 4.30: 监控服务-设备截屏的实现

```
@Override
public void run() {
    // 构造并执行 adb logcat 命令，获取进程的输入流
    List<String> args = new ArrayList<>(Arrays.asList("adb", "-s", udid, "logcat", "-b", "main", "-b",
        "system", "-b", "crash", "-b", "events", "-b", "radio", "-v", "time"));
    ProcessBuilder pb = new ProcessBuilder(args);
    pb.start();
    BufferedReader br = new BufferedReader(new InputStreamReader(pb.getInputStream()));
    // 读取并格式化写入每一行 Log
    while (!stoped)
        try {
            String line = br.readLine();
            if (line != null) {
                formatAndWriteLog(line);
            }
        } catch (IOException e) {}
    }
}
```

图 4.31: 监控服务-获取 Logcat 日志的实现

息线程需要先启动获取设备流畅度的线程，之后在循环体中每五秒钟分别去获取设备当前的 CPU 占用、内存占用、网络消耗、屏幕刷新率与电池温度等相关信息，最后进行格式化并保存在相关日志文件中。

获取相关信息的实现逻辑与图 4.32中所示的获取设备流畅度的实现逻辑基本一致，即：通过执行相关 adb 命令获取到命令的输出流，再对输出流进行模式匹配，最后获取到相关数值信息。故此处对获取其余信息的实现不再赘述。

```
@Override
public void run() {
    String[] args="adb", "-s", udid, "shell", "logcat", "-v", "time", "Choreographer:I", "*:S"));
    try {
        // 忽略执行子进程的代码
        // 读取子进程返回值，解析获得结果
        while((line = reader.readLine()) != null && !killed) {
            // 此处省略异常处理代码
            line = line.substring(50, line.length() - 71);
            Integer value = Integer.parseInt(line.trim());
            count.addAndGet(value);
        }
    } catch (IOException e) {} finally {
        killProcess();
    }
}
```

图 4.32: 监控服务-获取设备流畅度的实现

```
@Override
public void run() {
    // 启动帧率获取线程
    mSmMonitor.startSMMonitor(udid, pid);
    while (!isEnd) {
        try {
            getCpuInfo(udid, packageName); // 获取 CPU 信息
            getMemInfo(udid, packageName); // 获取内存信息
            getNetworkInfo(udid, packageName); // 获取网络信息
            getSmInfo();// 获取流畅度信息
            getBatteryTempInfo(udid); // 获取电池温度信息
            saveData(); //保存数据
        } catch (InterruptedException e) {}
    }
}
```

图 4.33: 监控服务-获取设备状态信息的实现

4.6　二次分析服务的设计与实现

4.6.1　二次分析服务架构设计

图 4.34为二次分析服务的架构图。该服务从上游服务接收 HTTP 请求，会对指定任务的中间数据使用指定的工具进行二次分析，生成二次分析报告。之后，

将二次分析报告上传至阿里云 OSS 中。最后，将报告通过 HTTP 请求回传至指定的报告接受服务。同时，上游服务还可以随时通过查询状态请求，获取二次分析的执行进度。

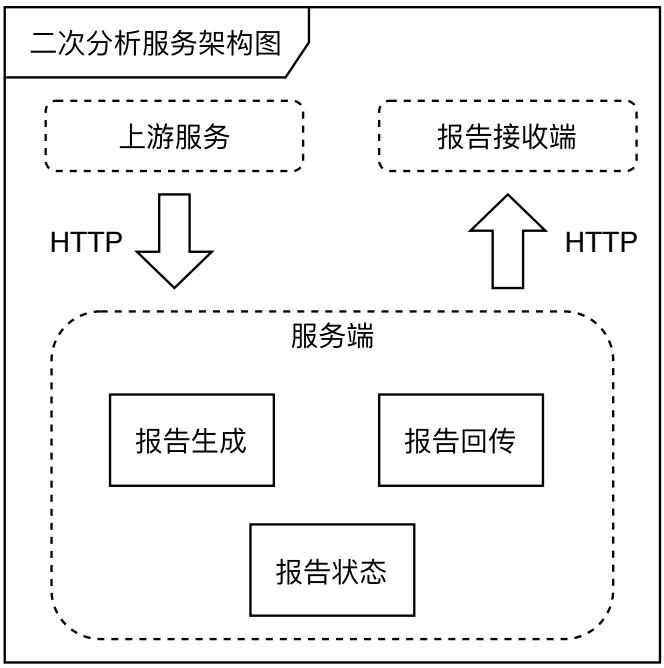

图 4.34: 二次分析服务-架构图

4.6.2　二次分析服务核心类图

图 4.35为二次分析服务的核心类图。SecondaryController 为对外提供路由分发的控制类。其通过调用 SecondaryService 接口提供的抽象功能来完成分析的开始与状态查询操作。SecondaryServiceImpl 为 SecondaryService 的实现类，因为二次分析为耗时操作，所以其构造了一个内部线程类 RunTestThread 负责执行具体的二次分析功能。在分析的过程中，通过 SecondaryStatusMap 实时更新二次分析的状态。分析结束后，通过 OssService 将分析结果上传至阿里云 OSS 中。最后，SecondaryStatus 为二次分析状态的枚举类，包括执行中、上传中、发送中、成功与失败。SecondaryResult 为二次分析结果的包装类，主要用于与上游服务进行数据传输。

4.6.3　二次分析服务顺序图

图 4.36为二次分析服务的顺序图。当 SecondaryController 收到二次分析请求时，会调用 SecondaryService 的 generateReport() 方法，在此方法中，会先检查参

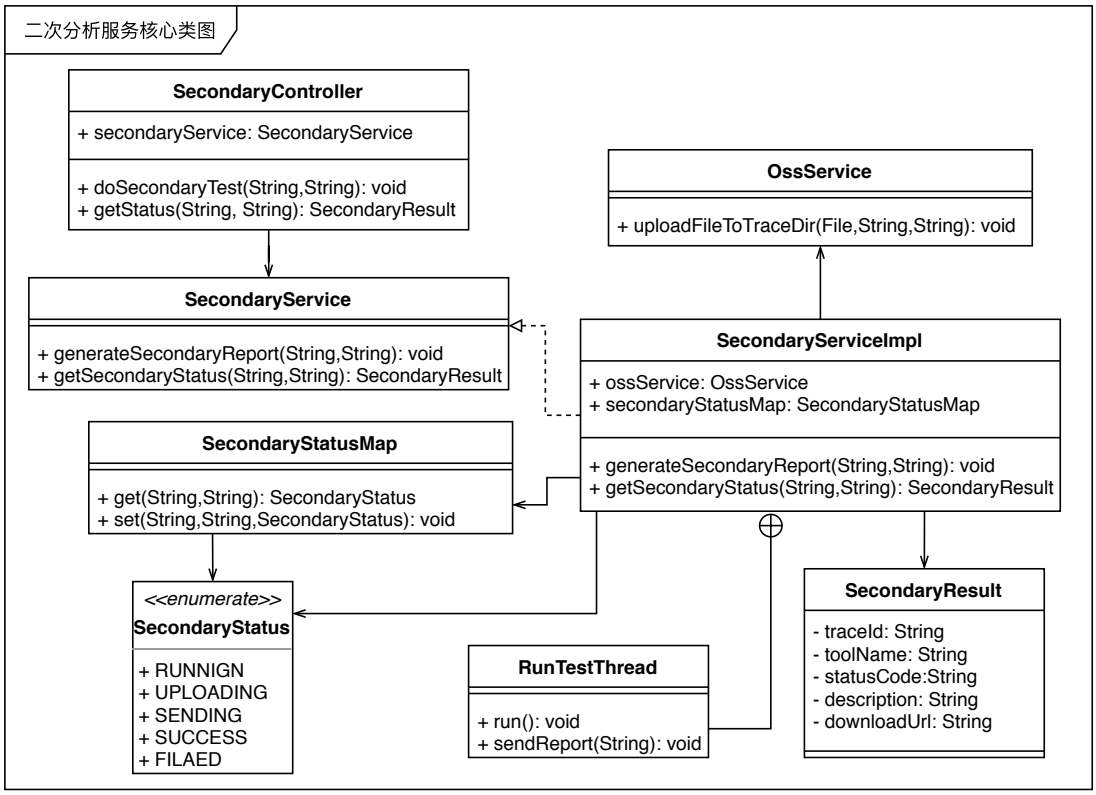

图 4.35: 二次分析服务-核心类图

数的有效性，之后生成一个新的执行线程并开始分析。线程开始后，Secondary-Service 则立刻返回。在执行线程内部，会一次执行报告生成、报告上传与报告回传操作，报告上传是通过 OssService 实现的。同时，在执行的过程中，执行线程会实时通过 SecondaryStatusMap 进行状态更新。

当 SecondaryController 收到查询状态请求时，SecondaryService 会调用 SecondaryStatusMap 进行状态查询，如果分析已完成，还会调用 OssService 获取到报告的下载链接。最后，服务会将所有信息拼装成 SecondaryResult 对象返回给上游服务。

4.6.4　二次分析服务关键代码

图 4.37为二次分析服务开始的核心代码实现。当二次分析服务收到开始的请求时, 会首先更新分析的状态为 RUNNING, 之后通过 OsUtil 工具类执行” java -jar toolName traceId “命令生成对应的分析报告，报告名默认与工具名一致。之后，通过 uploadReport() 方法将报告上传至阿里云 OSS 进行持久化储存。

图 4.38为二次分析服务结果上传的核心代码实现。首先需要将执行状态更新为 UPLOADING，之后判断报告文件是否存在，如不存在，则说明报告文件生

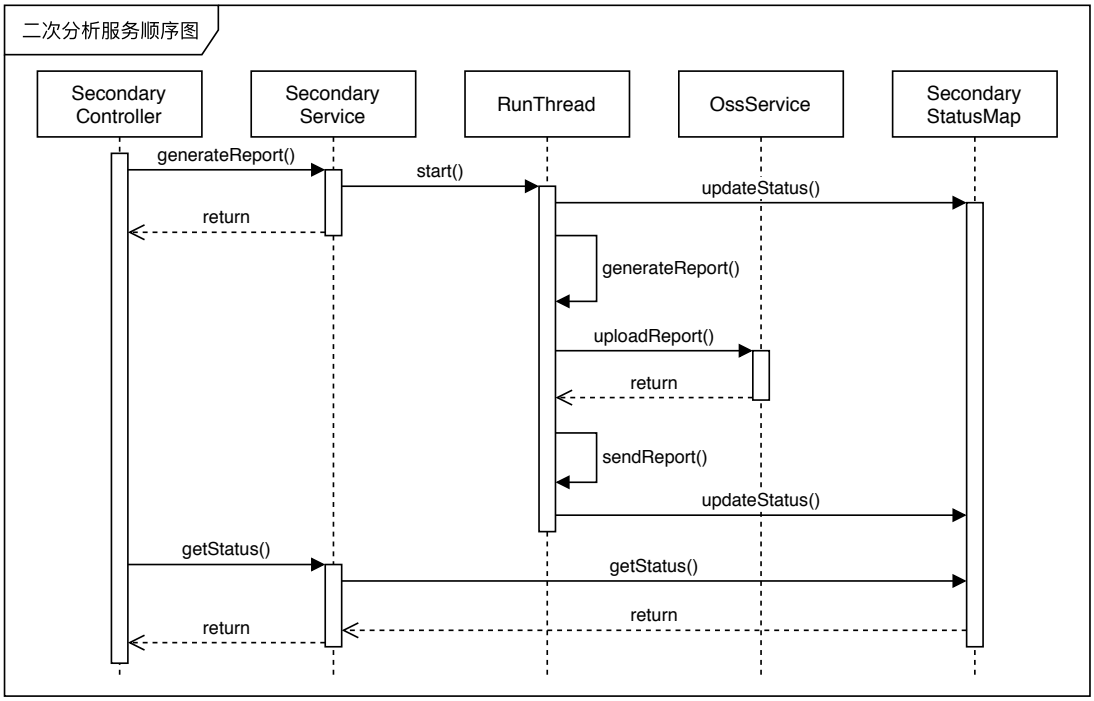

图 4.36: 二次分析服务-顺序图

```
@Override
public void run(){
    try {
        secondaryStatusMap.put(traceId, toolName, SecondaryStatus.RUNNING);
        // 执行生成命令
        String command = String.format("java -jar tasks/%s.jar %s", toolName, traceId);
        OsUtil.runCommand(command);
        String reportName = toolName + ".json";
        // 上传 & 回传报告
        uploadReport(reportName);
        secondaryStatusMap.put(traceId, toolName, SecondaryStatus.SUCCESS);
    } catch (Exception e) {// 异常处理}
}
```

图 4.37: 二次分析服务-执行的实现

成失败，抛出异常。如报告存在，则通过 OssService 的上传接口进行结果上传。上传成功则正常返回，失败则更新状态为 UPLOAD_FAILED。

图 4.39为二次分析服务状态查询的核心代码实现。首先，通过 SecondaryStatusMap 查询任务状态，状态不存在，则说明任务已过期或者从未开始过，继而抛出异常。之后，将获取到任务状态组装成 SecondaryResult 对象，如果任务状

```
private void uploadReport(String reportName){
    // 更新状态
    secondaryStatusMap.put(traceId, toolName, SecondaryStatus.UPLOADING);
    File file = new File(AddressUtil.getReportJsonPath(traceId, reportName));
    if (!file.exists()) {
        secondaryStatusMap.put(traceId, toolName, SecondaryStatus.RUN_FAILED);
        throw new RuntimeException("报告生成失败");
    }
    try {
        // 上传报告
        ossService.uploadFileToTraceDir(file, traceId, reportName);
    } catch (Exception e){
        secondaryStatusMap.put(traceId, toolName, SecondaryStatus.UPLOAD_FAILED);
        throw e;
    }
}
```

图 4.38: 二次分析服务-结果上传的实现

态为成功，则还会将任务报告的下载地址进行赋值。最后返回 SecondaryResult 对象。

```
@Override
public SecondaryResult getSecondaryStatus(String traceId, String toolName) {
    SecondaryStatus ss = secondaryStatusMap.get(traceId, toolName);
    if (ss == null){
        throw new HttpNotFoundException("任务不存在或信息已被清理");
    }
    // 此处忽略组装 SecondaryResult 的代码
    if (ss == SecondaryStatus.SUCCESS){
        sr.setDownloadUrl(ossService.getDownloadPath(traceId, toolName + ".json"));
    }
    return secondaryResult;
}
```

图 4.39: 二次分析服务-状态查询的实现

4.7　本章小结

本章节着重介绍了自动化测试服务、APK 服务、设备服务、测试执行服务、监控服务与二次分析服务的详细设计与实现。通过架构图展示了各个模块的架

构设计，通过核心类图与顺序图，从静态和动态两个角度展示了模块的设计思路，并在最后给出了各个模块关键代码的实现。

第五章　系统测试与系统实例展示

5.1　系统测试

前一章节对系统的各个模块服务进行了详细的设计说明与代码展示。本章节将从以下三个角度，对系统的测试工作进行说明。

（1）单元测试。单元测试也称模块测试，是对系统设计中的最小单位（在本系统中即为函数方法）进行测试的方法 [41]，通过对其依赖的其他模块方法进行 Mock 操作，以实现测试的独立性。单元测试为所有测试过程中的最小测试单位，其应由开发者自行编写，保证了项目代码最基础的质量。

（2）接口测试。接口测试是针对各类服务对外提供的 API 进行的测试，主要是为了保障系统对外提供的功能是否可靠，是否符合交付需求，是否符合性能需求 [42]，是否能按照预期返回相应的状态码以及内容。

（3）集成验收测试。此项测试为慕测科技产品成熟度第三级别的要求。其要求系统需要对《慕测产品服务验收测试集》中的所有测试数据集进行全覆盖，且不得出现死锁、无响应、崩溃、资源异常等情况。此项测试为慕测平台产品集成的最低要求。

5.1.1　测试环境

本系统的测试环境如表 5.1所示。本系统部署的操作系统为 Ubuntu16.04 系统，所使用的 Appium Server 版本为 1.7.2 版本，所使用的 JDK 版本为 1.8.0_191，Redis 缓存数据库的版本为 4.0.11，使用的 ADB 安卓调试桥的版本为 1.0.41，部署所使用的 Docker 版本为 18.09.2。

表 5.1: 系统测试环境

环境项	说明
操作系统	Ubuntu16.04
Appium 服务	Appium Server 1.7.2
JDK 版本	JAVA 1.8.0_191
Redis 版本	Redis 4.0.11
ADB 版本	ADB 1.0.41
Docker 版本	Docker 18.09.2, build 6247962

5.1.2　单元测试

单元测试为系统开发过程中非常重要的一环。本系统的开发语言为 Java 语言，故选用 Junit 测试框架作为单元测试的框架 [43]，同时使用 Mockito 和 PowerMockito 作为单元测试的 Mock 框架。

本系统所采用的系统架构为分层结构，共分为 Controller 层，Service 层与 Dao 层。由于不同层所负责处理功能侧重点有所不同，所以不同功能层单元测试的侧重点也应有所不同 [44]。Controller 层负责处理 HTTP 请求的接与并分发及参数校验，故其测试侧重点应该为异常参数的校验；Service 层负责处理具体业务逻辑，故其侧重点应该为正常业务流程的正确性以及异常业务流程是否被准确识别并捕获；Dao 层负责封装与底层数据库交互细节，故其侧重点应该为执行数据库操作命令语法的正确性。同时，由于本系统划分为多个功能模块，不同功能模块之间存在复杂的调用关系，为了使单元测试能更加关注于待测试函数本身，并使其独立于其他模块执行测试用例，故本系统使用 Mockito 和 PowerMockito 框架，用于模拟其他服务的调用或者返回。

表 5.2所展示的为本系统不同分层的单元测试用例统计表。

表 5.2: 单元测试用例

单元测试项	类型	说明	用例数量
AutoTestControllerTest	接口层	自动化测试服务接口	6
SecondaryControllerTest	接口层	二次分析服务接口	4
OssServiceTest	Service 层	Oss 服务	3
TraceServiceTest	Service 层	任务服务	5
AutoTestServiceTest	Service 层	自动化测试服务	6
SecondaryServiceTest	Service 层	二次分析服务	13
DeviceServiceTest	Service 层	设备服务	6
ApkServiceTest	Service 层	Apk 服务	3
DeviceRunningStatusMapTest	Dao 层	设备执行状态 Dao	3
DeviceStatusMapTest	Dao 层	设备占用状态 Dao	4
TraceStatusMapTest	Dao 层	任务执行状态 Dao	3
TraceDeviceMapTest	Dao 层	任务设备映射关系 Dao	3
SecondaryStatusMapTest	Dao 层	二次分析状态 Dao	3
总计			62

共计 13 个测试项，Controller 层 2 项，分别针对自动化测试服务与二次分析服务对外提供接口进行测试，测试重点为异常参数的校验，如参数类型错误，缺少必要参数等。Service 层 6 项，其主要测试的是各类 Service 所提供的服务是否能准确捕获与处理正常与异常流程。Dao 层 5 项，其主要测试的是在于底层数据

库引擎交互的过程中命令的正确定。所有测试用例共计 62 个。

单元测试的结果如图 5.1所示。所有用例的执行时间均低于预期时间，通过率为 100%（62/62）。在系统的后续开发中，每次系统需要打包部署前，都会至少执行上述测试用例，如果全部通过，则可以正常执行打包部署；如果出现失败的情况，则说明系统在改动过程中改变了某些业务的流程或逻辑，这种情况下，就需要针对特定的情况对代码或者相关测试用例进行修改。

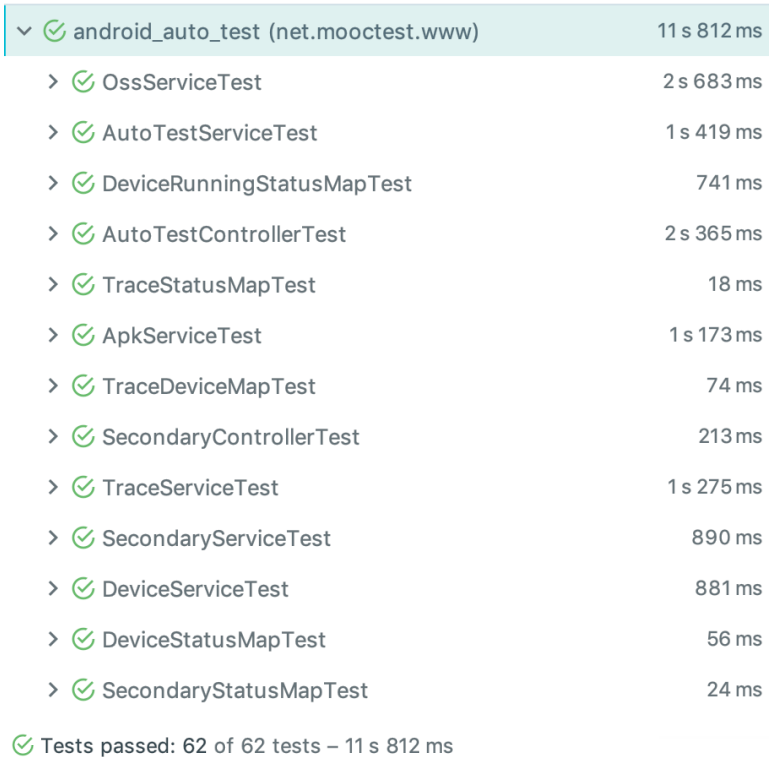

图 5.1: 单元测试结果

5.1.3　接口测试

接口测试是一种针对服务与服务之间交互的点的测试，狭义上可以理解为针对系统 API 的测试，是一种发生在信息层的测试。由于现阶段敏捷开发较为流程，不论是 UI 界面还是系统内部实现都会有很快的迭代速率，但是系统对外提供服务的接口大部分情况下是稳定不变的，所以针对接口进行测试有着极高的重要性，同时接口测试也是自动化测试过程中非常重要的一环 [45]。

由于本系统为安卓自动化测试中台系统，对外提供了泛化的安卓自动化测试能力，对上游系统的实现无依赖，且上游系统可以任意切换。所以需要针对本系统对外提供的接口进行测试。接口测试所测试的重点在于不同服务之间交互

数据的一致性以及服务返回状态码是否符合预期 [46]，本系统的接口测试也主要针对上述两个测试点进行测试。

本系统所采用的接口测试的工具为 Postman。Postman 为后端开发常见的接口测试工具，其提供了强大的接口调用与测试能力。本系统所执行的接口测试用例如表 5.3所示。

表 5.3: 接口测试用例

测试 ID	说明	输入	预期输出	结果
IT-1	测试启动测试正常流程	正确任务元信息对象	正确任务状态对象	通过
IT-2	测试启动测试异常流程	缺少参数的任务元信息对象	抛出 HTTP400 异常	通过
IT-3	测试获取任务状态正常流程	正确的任务 ID	任务状态对象	通过
IT-4	测试获取任务状态任务 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-5	测试停止任务正常流程	正确的任务 ID	true	通过
IT-6	测试停止任务 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-7	测试正常执行二次分析流程	正确的任务 ID	true	通过
IT-8	测试执行二次分析 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过
IT-9	测试正常查询二次分析状态流程	正确的任务 ID	true	通过
IT-10	测试查询二次分析状态 ID 不存在流程	错误的任务 ID	抛出 HTTP404 异常	通过

接口测试的结果如图 5.2所示。共计 10 个请求，5 个 POST 请求、4 个 GET 请求、1 个 DELETE 请求；共计 24 个测试单元，这些测试单元分别从请求返回的状态码、错误信息以及执行耗时三个方面进行了验证；共执行了 10 轮 240 次测试。测试结果为全部通过。

5.1.4　集成验收测试

根据慕测科技产品成熟度第三级别的要求，任何向慕测平台集成的系统必须对《慕测产品服务验收测试集》中的所有测试数据集进行全覆盖，且不得出现死锁、无响应、崩溃、资源异常等情况。因此，本系统针对其中《安卓应用自动化测试验收测试数据集》，在共计 12 台安卓设备上进行了集成验收测试。

本次集成验收测试选取了市面上较为常见的 12 台安卓手机，作为测试机型。共覆盖了 6 个品牌，4 个安卓版本。其详细信息如表 5.4所示。

图 5.2: 接口测试结果

表 5.4: 集成测试机型

设备 ID	设备识别码	品牌	型号	安卓版本
D1	8DF6R16729019298	荣耀	Honer Note8	8.0.0
D2	36b7a00	小米	MI 6X	9.0.0
D3	63fa9ed5	小米	MI 8	9.0.0
D4	66J5T19401004951	华为	Mate20	10.0.0
D5	ca8684d9	红米	Note 7 Pro	9.0.0
D6	ce0717179034e124027e	三星	Galaxy Note 8	9.0.0
D7	CLB0218724002507	华为	P20	9.0.0
D8	FJH5T18830019181	华为	Nova 3	9.0.0
D9	Q5S5T19411008221	华为	P30	10.0.0
D10	RKKDU18323002730	荣耀	Honer V10	9.0.0
D11	WBUBB18923510113	荣耀	Honer 8X MAX	9.0.0
D12	2003161a	OPPO	OPPO R17	8.1.0

本次集成验收测试，共针对表 5.5、表 5.6、表 5.7中所示的 206 个闭源安卓应用进行了测试。这些闭源应用随机选择自国内各大安卓应用市场下载量排行榜. 其类型分布如图 5.3所示。这 206 个应用共涉及工具、生活、视频、金融、阅读、社交、购物、拍照、导航、学习等多种类型，除了较为特殊的手机游戏之外，已经覆盖了市面上绝大多数的应用类型。

表 5.5: 慕测安卓自动化集成验收测试集 1

ID	应用名	版本	类型	ID	应用名	版本	类型
1	12306 官方版	4.2.36	生活	2	12306 铁友火车票	8.0.1	生活
3	139 邮箱	8.9.9	工具	4	2345 天气王	7.9	生活
5	360 借条	1.6.1	金融	6	360 手机卫士	8.3.0	工具
7	365 淘房	8.0.70	生活	8	58 同城	9.1.2	生活
9	BeautyCam 美颜相机	9.0.20	相机	10	Boss 直聘	7.120	生活
11	CCtalk	7.6.3	社交	12	KingRoot	5.4.0	工具
13	MOMO 约	1.3.2	社交	14	PP 体育	5.16	视频
15	PicACG	2.0.0	阅读	16	QQ	8.1.5	社交
17	QQ 浏览器	9.7.1	浏览	18	QQ 邮箱	5.6.9	工具
19	QQ 音乐	9.5.0.6	影音	20	RAR for Android	5.71	工具
21	Speedtest	4.4.19	工具	22	Steam	2.3.11	工具
23	UC 浏览器	12.6.6	浏览	24	UC 浏览器极速版	12.0.3	浏览
25	U 校园	2.0.0.0	工具	26	WPS Office	12.0.4	工具
27	WiFi 万能钥匙极速版	6.0.63	工具	28	WiFi 钥匙	5.7.7	工具
29	YY	7.21.21	社交	30	七猫免费小说	3.7	阅读
31	丝瓜视频	4.3.1	视频	32	中国建设银行	4.1.9	金融
33	中国移动	5.7.0	生活	34	中国银行	6.0.9	金融
35	中通快递	5.2.9	生活	36	中青看点	1.6.0	阅读
37	乐教乐学	1.0.187	学习	38	九游	5.2.7.1	工具
39	书旗小说	10.9.5.93	阅读	40	二手手机找靓机	7.5.01	生活
41	云闪付	6.5.1	金融	42	交管 12123	2.3.2	生活
43	京东	8.3.0	购物	44	京东金融	5.2.80	金融
45	今日头条	7.4.5	阅读	46	优酷视频	8.1.4	视频
47	作业帮	12.1.4	学习	48	先锋影音	5.6.6	影音
49	免费小说听书	6.4.9	阅读	50	全本免费小说阅读器	1.7.6	阅读
51	全民 K 歌	6.10.8	影音	52	农行掌上银行	4.1.0	金融
53	凤凰新闻	6.7.0	阅读	54	刷宝	1.800	视频
55	千千静听百度音乐版	2.1.0	影音	56	同城约会	4.7.5	社交
57	向日葵远程控制	9.8.5	工具	58	呆萝卜	3.9.7	工具
59	和包支付	8.5.46	金融	60	咪咕视频	5.6.6.10	视频
61	哈🅴出行	5.25.1	生活	62	哔哩哔哩	5.48.3	视频
63	喜马拉雅	6.6.18.3	影音	64	嗨来电秀	1.7	工具
65	墨迹天气	7.0928.02	生活	66	壁纸多多	4.3.8.0	工具
67	多鹿	1.7.300	工具	68	大众点评	10.19.12	生活
69	天天快报	6.0.80	阅读	70	央视影音	6.7.6	影音
71	夸克	3.5.1.118	浏览	72	学习强国	2.6.1	阅读
73	安全教育平台	1.5.3	工具	74	安居客	12.21.2	生活
75	宜人贷借款	5.9.8	金融	76	宝宝树孕育	8.11.0	生活
77	小猿搜题	9.9.0	学习	78	小米路由器	5.2.1	工具
79	小红书返利	1.7.2	生活	80	屏幕自动点击助手	3.1.1	工具

表 5.6: 慕测安卓自动化集成验收测试集 2

ID	应用名	版本	类型	ID	应用名	版本	类型
81	工行手机银行	4.1.0.5.3	金融	82	平安金管家	5.06.11	金融
83	度小满金融	4.0.9	金融	84	录音宝	3.0.1421	工具
85	影视大全	3.2.0	影音	86	微信	7.0.7	社交
87	微博	9.9.3	社交	88	微鲤	1.6.2	生活
89	志愿汇	3.4.2	工具	90	快去水印	3.5	工具
91	快对作业	2.28.0	学习	92	快手	6.8.2	视频
93	快手极速版	1.5.0.67	视频	94	快点	2.36.11	影音
95	快猫	1.4.0	视频	96	快看漫画	5.54.0	阅读
97	快递 100	5.8.0	生活	98	悦跑圈	5.3.1	工具
99	手机公积金查询	3.9.2	生活	100	手机淘宝	9.1.0	购物
101	手机电视	8.1.1	工具	102	抖音极速版	2.0.1	视频
103	抖音短视频	8.3.0	视频	104	护眼宝防蓝光	9.5	工具
105	招商银行	7.5.0	金融	106	拼多多	4.77.0	购物
107	拼车顺风车	5.19.0	生活	108	探探	3.6.8.1	社交
109	搜书大师	16.12	阅读	110	搜狐视频	7.6.5	视频
111	搜狗地图	10.5.0	地图	112	支付宝	10.1.75	金融
113	斗鱼直播	6.0.0	视频	114	时光相册	2.6.3	工具
115	智慧人社	3.9.6	工具	116	智能公交	3.8.0	工具
117	智行火车票 12306 高铁抢票	8.0.1	生活	118	有妖气漫画	4.8.0	阅读
119	步多多	1.0.5	工具	120	水印相机	3.1.3.473	相机
121	永安行	4.16	工具	122	淘集集	2.28.0	购物
123	湘行一卡通	2.0.9	金融	124	滴滴出行	5.3.8	生活
125	澳客	1.9.1	金融	126	火币 PRO	9.0.0	工具
127	火萤视频壁纸	7.0.0	工具	128	火锅视频	2.5.1	视频
129	爱奇艺	10.12.5	视频	130	爱奇艺极速版	9.9.1	视频
131	猎豹清理大师	6.11.9	工具	132	猫语翻译器	2.4.3	工具
133	球球视频	2.4.0.3	视频	134	生肖宝典	1.0	工具
135	电信营业厅	7.5.0	生活	136	男人帮	1.2	社交
137	番茄社区	3.0.5	社交	138	百姓网	9.7.3	阅读
139	百度	11.13.8.10	工具	140	百度网盘	10.0.73	工具
141	百度贴吧	10.3.8.12	工具	142	皮皮蟹	4.19.5	社交
143	稿定设计	3.9.3	工具	144	精读圣经	2.7.9	阅读
145	网商银行	3.2.1	金融	146	网易 UU 加速器	2.3.7	工具
147	网易云音乐	6.4.2	影音	148	网易邮箱大师	6.17.2	工具
149	美团骑手	5.2.3.992	生活	150	美图秀秀	8.6.6.1	相机
151	美拍	8.3.28	相机	152	翼支付	6.5.2	金融
153	考拉海购	4.17.0	购物	154	联通手机营业厅	6.2.1	生活
155	腾讯地图	8.9.3	地图	156	腾讯手机管家	8.0.1	工具
157	腾讯视频	7.6.0	视频	158	芒果 TV	6.5.0	视频
159	花间	7.5.10	视频	160	苏宁易购	8.0.2	购物

表 5.7: 慕测安卓自动化集成验收测试集 3

ID	应用名	版本	类型	ID	应用名	版本	类型
161	菜鸟裹裹	5.5.0	生活	162	菱菱邦	7.5.4	生活
163	萌推	2.5.0	购物	164	虎牙直播	7.4.6	视频
165	西瓜视频	3.9.7	视频	166	要出发周边游	5.9.98	生活
167	触宝电话	6.7.8.8	工具	168	贝贝	9.25.01	购物
169	转转	7.1.2	购物	170	迅雷	6.09.2	工具
171	邮储银行	4.1.5	金融	172	酷我音乐	9.2.4.2	影音
173	酷狗音乐	9.3.5	影音	174	钉钉	4.7.10	社交
175	铃声多多	8.7.70.0	工具	176	闪送	5.2.20	生活
177	闲鱼	6.5.51	购物	178	阿里星球	10.0.8	影音
179	音悦台	4.7.7	影音	180	饿了么	8.24.4	生活
181	饿了么商家版	7.23.2	生活	182	首汽约车	7.0.10	生活
183	香蕉视频	2.5.0	视频	184	驾考宝典	7.4.8	工具
185	高德地图	10.10.0	地图	186	高速 ETC	v3.9.2	生活
187	麦芽贷	3.5.2	金融	188	QQ 空间	8.5.1	社交
189	最美天气	6.04.001	工具	190	搜狗浏览器	5.27.8	浏览
191	趣头条	3.9.59	阅读	192	美团外卖	7.27.2	生活
193	摩拜单车	8.25.2	生活	194	车轮	8.2.0	生活
195	天气通	6.28	工具	196	汽车之家	10.4.5	生活
197	赶集网	8.23.8	生活	198	安居客	12.24.2	生活
199	美丽说	10.6.2	生活	200	前程无忧	9.0.1	生活
201	下厨房	7.2.3	生活	202	猫眼	9.0.0	生活
203	口碑	7.1.82	生活	204	百度糯米	8.6.12	购物
205	当当	2.10.1	购物	206	个人所得税	1.2.2	工具

本次集成验收实验，通过 Postman 与慕测企业版云平台，向本系统发送开始执行安卓自动化测试任务的请求。以是否可以正常启动 AppiumDriver 为标准，对应用是否可以正常进行自动化测试进行判断。实验结果如表 5.8所示。只有 19 个应用无法在本系统中正常进行自动化测试，测试通过率为 90.78%。虽然这 19 个应用无法进行自动化测试，但是这些应用并未引起系统的死锁、无响应、崩溃、资源异常等问题，说明本系统有很强的鲁棒性，完全符合慕测科技集成验收测试标准。

表 5.8: 集成验收实验结果

执行结果	应用数量
正常执行	187
非正常执行	19

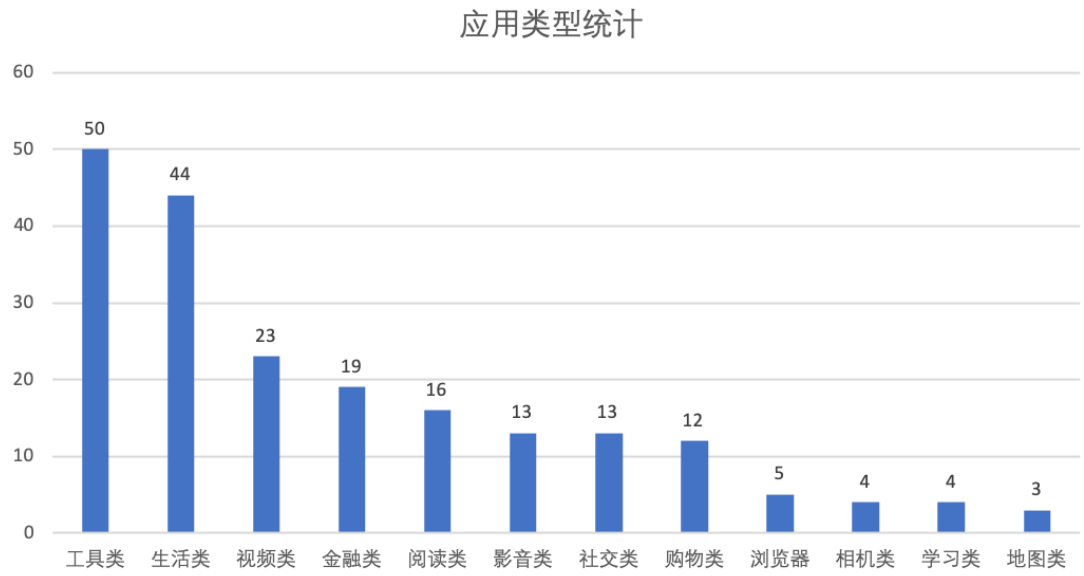

图 5.3: 应用类型数量统计

同时，对于无法进行测试的 19 个应用，我们又进行了进一步的分析。通过分析，我们发现应用无法通过测试主要包含两大原因。这 19 个应用的详细信息如表 5.9所示。总结来说，共有两种原因：

（1）应用配置不规范。共 14 款应用出现该问题。该问题主要体现在应用未规范使用第三方库，导致存在多个可启动 Activity，又或是应用未配置可启动 Activity，导致 AppiumDriver 无法识别，从而无法正常进行自动化测试。详细来说，即 Appium 进行自动化测试需要通过 APK 的 launchable-activity 属性来启动 APP，如果应用没有配置或者配置了超过 1 个 launchable-activity，则 Appium 就无法正确启动 APP，从而导致无法正常进行测试。

（2）应用自身不可用。共 5 款应用出现该问题。主要体现在应用因年久失修、无人维护，启动应用会出现闪退、黑屏等兼容性问题。

上述两个问题均是应用自身原因导致无法正常进行自动化测试。因此集成验收测试的结果可以充分说明本系统已对不同的安卓设备以及安卓应用有了较高的支持度。完全符合慕测科技产品成熟度第三级别的要求。

5.2　系统实例展示

图 5.4所示的为慕测科技企业版创建安卓自动化测试任务的界面，在该界面中，通过上传待测 APK，配置任务基础信息以及设置任务执行时间，便可以创建一个安卓自动化测试任务。

表 5.9: 失败应用详情

ID	应用名称	版本号	失败原因
1	千千静听百度音乐版	2.1.0	应用黑屏
2	嗨来电秀	1.7	应用闪退
3	屏幕自动点击助手	3.1.1	应用闪退
4	翼支付	6.5.2	应用闪退
5	阿里星球	10.0.8	应用闪退
6	二手手机找靓机	7.5.01	未配置可启动 Activity
7	KingRoot	5.4.0	未配置可启动 Activity
8	时光相册	2.6.3	未配置可启动 Activity
9	抖音短视频	8.3.0	未配置可启动 Activity
10	搜狐视频	7.6.5	未配置可启动 Activity
11	爱奇艺	10.12.5	未配置可启动 Activity
12	爱奇艺极速版	9.9.1	未配置可启动 Activity
13	花间	7.5.10	未配置可启动 Activity
14	饿了么	8.24.4	未配置可启动 Activity
15	PP 体育	5.16	存在多个可启动 Activity
16	安居客	12.21.2	存在多个可启动 Activity
17	安居客	12.24.2	存在多个可启动 Activity
18	平安金管家	5.06.11	存在多个可启动 Activity
19	高德地图	10.10.0	存在多个可启动 Activity

图 5.4: 发布自动化测试任务截图

图 5.5所示的为自动化测试任务执行完成后，所生成的报告概述界面。在报告概述界面中，可以看到所提交 APK 的基本信息，以及所发现 Bug 的统计信息，通过饼图直观的像用户展示所发现 Bug 的分类以及严重等级的分布情况。

图 5.5: 自动化测试报告-任务概述

图 5.6所示的为自动化测试报告的 Bug 列表部分。该部分列出了本次测试任务中发现的所有潜在 Bug，展现了每一个 Bug 的类别、描述、严重等级、所出现的机型等。该列表可以通过 Bug 类型或者 Bug 严重等级进行筛选，同时可以通过 Bug 详情按钮查看 Bug 的具体信息。

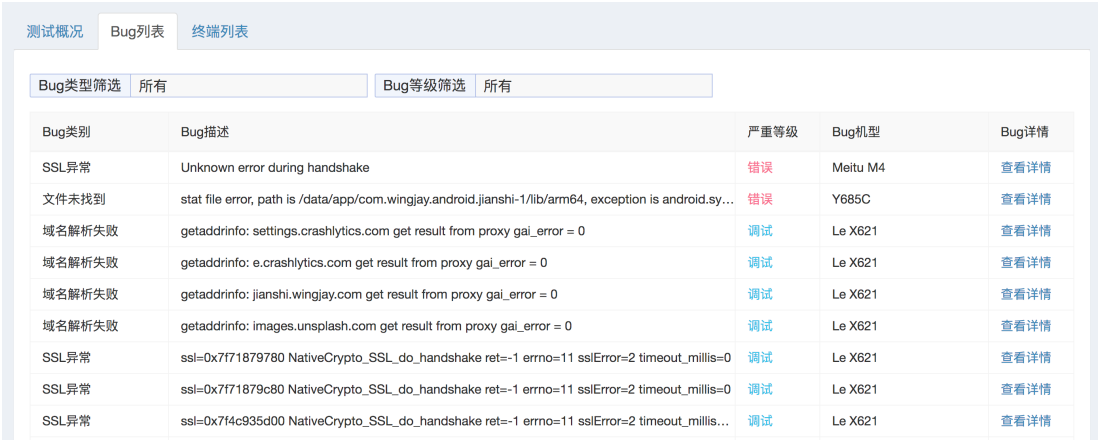

图 5.6: 自动化测试报告-Bug 列表

图 5.7所示的为自动化测试报告的 Bug 详情部分。在该部分，可以看到每一个 Bug 的详细信息，包括严重等级、发生的页面、上下文信息、不一致性标签等。同时，报告还会给出该 Bug 可能发生的原因以及解决方案。为用户提供了较为良好的使用体验。

图 5.8所示的为自动化测试报告的设备执行详情页面。在该页面中，用户可以了解到，在本次任务中每一台安卓设备的具体信息。包括设备的基础信息和

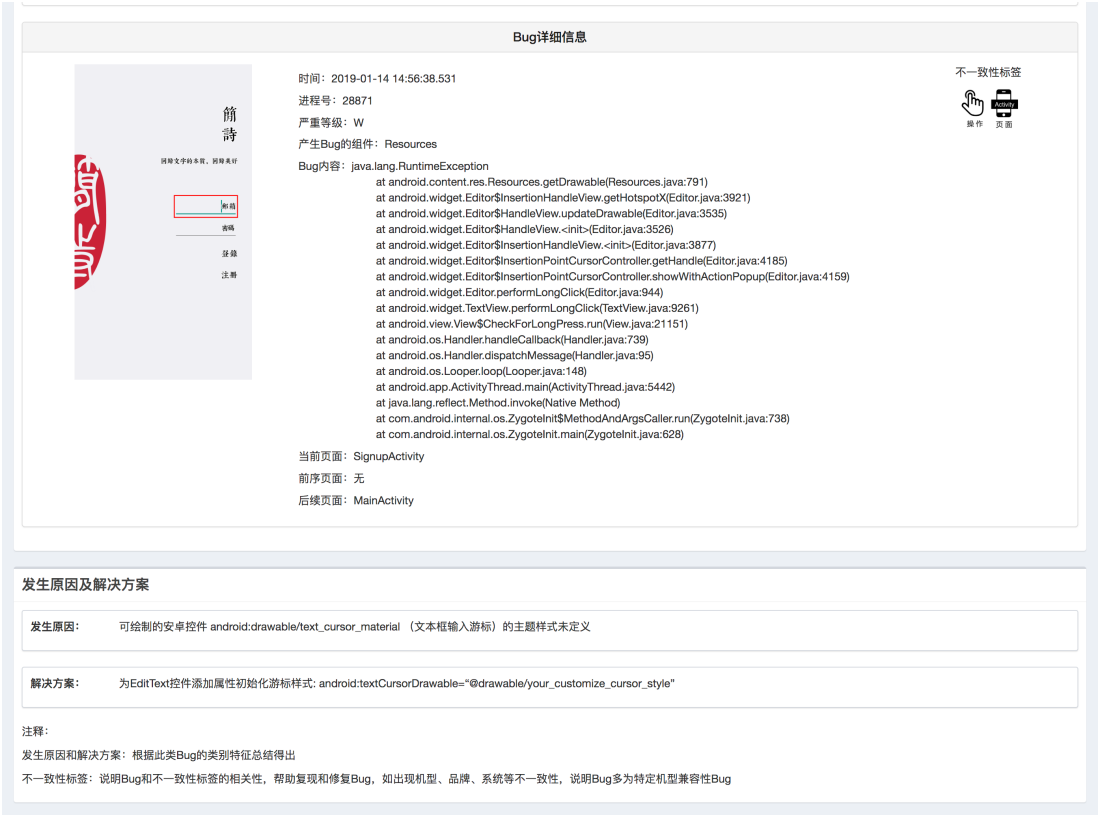

图 5.7: 自动化测试报告-Bug 详情

图 5.8: 自动化测试报告-终端详情

设备执行过程中的性能表现。同时，报告还会给出设备的执行日志以及执行过程中的所有截图，以帮助用户更好的复现测试过程、快速定位问题。

5.3　本章小结

本章节从单元测试、接口测试以及集成验收测试三个方面对系统的测试工作进行了详细的介绍。通过上述三种测试，可以极大程度上保证系统的各方面质量，可以保障系统持续稳定的提供可靠服务。最后，展示了部分系统的实际运行截图并给出了相关说明。

第六章　总结与展望

6.1　总结

随着移动互联网的快速发展，智能手机在我们生活中扮演着越来越重要的角色。与此同时，手机用户已经不仅仅满足于智能应用的功能，而更多的在追求应用的使用体验与感受。所以，针对移动应用的测试工作就显得越来越重要。对于占据着 87% 市场份额的安卓系统来说尤为重要。因为安卓系统的开源性以及国内手机市场的蓬勃发展，造成了安卓系统碎片化的问题。同时，谷歌公司还在快速对安卓系统进行迭代升级。碎片化问题和安卓系统快速迭代的现象，导致了安卓应用的测试工作会消耗非常多的测试成本。许多企业转向使用第三方的测试服务，以降低自身的成本。如何为企业团队提供通用的安卓自动化测试解决方案，成为了当前比较热门的话题之一。另一方面，国内外各大高校也纷纷开设安卓开发课程，以应对安卓人才紧缺的问题。对于这些课程，以往都是用人工评审的方式进行考核，但是这样主观性太强，无法做到公平公正，同时又十分耗费时间。如何使用自动化的方式对学生的课程作业进行评价，也是各大高教比较关注的问题之一。

在此背景下，为了应对不同场景下的需求差异，适应各类需求变化的风险，依靠于慕测科技丰富的高校资源与企业资源，本文设计并实现了一套安卓应用自动化测试中台系统，以同时支撑上述多种复杂场景对于安卓应用自动化测试不同的需求。本系统通过抽象良好的接口、可配置的测试脚本与分析工具，提供了移动应用泛化的测试能力，具有对上游服务无依赖、上游服务可以随意切换等特点。所以本系统可以适应各类业务需求与场景，为企业、高校乃至任意第三方提供可靠稳定的安卓自动化测试服务。

本文首先介绍了系统所依赖的一些科学理论、技术框架以及开源工具，并详细阐述了选择相应工具的背景及理由。其次，结合本人所学的软件需求分析、软件架构设计、高级人机交互等软件工程领域专业课程，对系统进行了详细的涉众分析、需求分析与用例分析。之后根据分析结果，从不同的角度对系统进行了更为详细的设计与分析，并通过系统架构图、系统 4+1 视图和其他图表的形式加以展现。在第四章，本文阐述了系统各个模块的详细设计与实现，通过架构图、类图、顺序图，分别从静态与动态的角度对每个模块进行了详细的分析与说明。同时展示了模块的关键代码，以辅助阅读与理解。最后，为了保障系统的质量，本系统进行了全面的测试工作，包括单元测试与接口测试。同时为了能够

和慕测科技企业版与教育版顺利集成，本系统还进行了大规模的集成验收测试，共在 12 台安卓设备上，针对 206 个常见商业应用进行了测试与实验，共产出超过 80G 的实验数据供实验室后续研究使用。同时，在测试期间，系统并未出现死锁、无响应、崩溃、资源异常等情况，这足以说明本系统可以提供可靠稳定的服务。

6.2　展望

目前，本系统已部署于慕测科技，为慕测科技企业版与教育版提供着稳定可靠的安卓自动化测试服务。但是还有许多需要改进的地方，主要包括以下几个方面。

（1）本系统作为测试中台系统，通用性与稳定应已经可以较好的保证。但是由于 Appium 框架与 ADB 工具的架构特性，在测试过程中，会偶发手机失联或者指令卡住的情况，这些情况会在一定程度上影响自动化测试的效率。如何应对手机失联与指令卡住的情况，也是我们后续较为关注的改进方向之一。

（2）由于本系统需要同时操作多台安卓设备进行自动化遍历测试，不同设备之间在组件定位、设备通讯等方面存在些许差异。虽说目前类似的现象极少出现，但是随着后续安卓设备的增多，如何对于设备之间的差异进行管理，是接下来要考虑的重要问题。可以考虑通过大规模的设备实验，针对不同机型，提出定制化的解决方案，构成一套方案库。

（3）目前，本系统仅提供了一套默认的自动化遍历测试脚本。后续可以考虑对脚本进行拆分，形成多个基础测试项，供用户进行选择与组装，从而提供更加灵活多变的服务，以适应用户不同的测试需求。

（4）随着 AI 技术的进一步发展，本系统在后续的开发中会考虑到加入图像识别、知识图谱等理论方法作为辅助，以更好的进行自动化测试。

参考文献

[1] IDC, Smartphone challenges continue in 2019, but 5g and emerging markets will bring growth back to the market in 2020, according to idc, https://www.idc.com/getdoc.jsp?containerId=prUS45487719.

[2] 人民网, 2019 年国内手机出货量 3.89 亿部 5g 手机 1376.9 万部, http://finance.people.com.cn/n1/2020/0109/c1004-31542002.html.

[3] L. Wei, Y. Liu, S.-C. Cheung, Taming android fragmentation: Characterizing and detecting compatibility issues for android apps, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 226–237.

[4] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, E. Stroulia, Understanding android fragmentation with topic analysis of vendor-specific bugs, in: 2012 19th Working Conference on Reverse Engineering, IEEE, 2012, pp. 83–92.

[5] S. R. Choudhary, A. Gorla, A. Orso, Automated test input generation for android: Are we there yet?(e), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 429–440.

[6] Google, The monkey ui android testing tool, http://developer.android.com/tools/help/monkey.html.

[7] A. Machiry, R. Tahiliani, M. Naik, Dynodroid: An input generation system for android apps, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 224–234.

[8] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, A. M. Memon, Using gui ripping for automated testing of android applications, in: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, IEEE, 2012, pp. 258–261.

[9] T. Azim, I. Neamtiu, Targeted and depth-first exploration for systematic testing of android apps, in: Proceedings of the 2013 ACM SIGPLAN international confer-

ence on Object oriented programming systems languages & applications, 2013, pp. 641–660.

[10] W. Yang, M. R. Prasad, T. Xie, A grey-box approach for automated gui-model generation of mobile applications, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2013, pp. 250–265.

[11] S. Yu, C. Fang, Y. Feng, W. Zhao, Z. Chen, Lirat: Layout and image recognition driving automated mobile testing of cross-platform, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2019, pp. 1066–1069.

[12] X. Zhang, Y. Feng, D. Liu, Z. Chen, B. Xu, Research progress of crowdsourced software testing, Journal of Software 29 (1) (2018) 69–88.

[13] R. Hao, Y. Feng, J. A. Jones, Y. Li, Z. Chen, Ctras: Crowdsourced test report aggregation and summarization, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 900–911.

[14] Y. Li, R. Hao, Y. Feng, J. A. Jones, X. Zhang, Z. Chen, Ctras: a tool for aggregating and summarizing crowdsourced test reports, in: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2019, pp. 406–409.

[15] H. Li, C. Fang, Z. Wei, Z. Chen, Cocotest: collaborative crowdsourced testing for android applications, in: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2019, pp. 390–393.

[16] X. Chen, H. Jiang, Z. Chen, T. He, L. Nie, Automatic test report augmentation to assist crowdsourced testing, Frontiers of Computer Science 13 (5) (2019) 943–959.

[17] 高国伟, 阿里巴巴技术中台的“云启示”, 项目管理评论 2 (2018) 26–27.

[18] 钟华, 阿里巴巴中台战略思想与架构实战, 机械工业出版社, 2017.

[19] 郭忆, 网易数据中台建设实践, https://www.infoq.cn/article/K29hNd0osXPjCNLpJ2Wv.

[20] S. Amatya, A. Kurti, Cross-platform mobile development: challenges and opportunities, in: International Conference on ICT Innovations, Springer, 2013, pp. 219–229.

[21] M. Hans, Appium essentials, Packt Publishing Ltd, 2015.

[22] G. Shah, P. Shah, R. Muchhala, Software testing automation using appium, International Journal of Current Engineering and Technology 4 (5) (2014) 3528–3531.

[23] 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.

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

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

[26] Google, Android debug bridge (adb), https://developer.android.google.cn/studio/command-line/adb.

[27] S. Hwang, S. Lee, Y. Kim, S. Ryu, Bittersweet adb: Attacks and defenses, in: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, 2015, pp. 579–584.

[28] R. Regupathy, Android debug bridge (adb), in: Unboxing Android USB, Springer, 2014, pp. 125–138.

[29] J. Amarante, J. P. Barros, Exploring usb connection vulnerabilities on android devices breaches using the android debug bridge, in: 14th International Joint Conference on e-Business and Telecommunications, ICETE 2017, SciTePress, 2017, pp. 572–577.

[30] J. L. Carlson, Redis in action, Manning Shelter Island, 2013.

[31] S. Heule, M. Nunkesser, A. Hall, Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm, in: Proceedings of the 16th International Conference on Extending Database Technology, 2013, pp. 683–692.

[32] R. P. Goldberg, Survey of virtual machine research, Computer 7 (6) (1974) 34–45.

[33] D. Bernstein, Containers and cloud: From lxc to docker to kubernetes, IEEE Cloud Computing 1 (3) (2014) 81–84.

[34] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux journal 2014 (239) (2014) 2.

[35] C. Boettiger, An introduction to docker for reproducible research, ACM SIGOPS Operating Systems Review 49 (1) (2015) 71–79.

[36] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, Groovy in action, Manning Publications Co., 2007.

[37] K. A. Kousen, Making Java Groovy, Manning, 2014.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, MIT press, 2009.

[39] R. Tarjan, Depth-first search and linear graph algorithms, SIAM journal on computing 1 (2) (1972) 146–160.

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

[41] H. Zhu, P. A. Hall, J. H. May, Software unit test coverage and adequacy, Acm computing surveys (csur) 29 (4) (1997) 366–427.

[42] A. Reichert, Testing apis protects applications and reputations, Dostopno na: https://searchsoftwarequality. techtarget. com/tip/Testing-APIs-protects-applicationsand-reputations [29.7. 2019].

[43] V. Massol, T. Husted, JUnit in action, Manning Publications Co., 2003.

[44] M. Shaw, D. Garlan, et al., Software architecture, Vol. 101, prentice Hall Englewood Cliffs, 1996.

[45] B. Mehta, RESTful Java Patterns and Best Practices, Packt Publishing Ltd, 2014.

[46] 侯可佳, 白晓颖, 陆皓, 李树芳, 周立柱, 基于接口语义契约的 web 服务测试数据生成, 软件学报.

简历与科研成果

基本情况 李灏宇，男，汉族，1994 年 10 月出生，江苏省徐州市人。

教育背景

2017.09 ~ 2020.06 南京大学软件学院 硕士

2013.09 ~ 2017.06 南京大学软件学院 本科

以下是读研期间的成果

1. **Haoyu Li**，Chunrong Fang，Zhibin Wei, Zhenyu Chen，“CoCoTest: collaborative crowdsourced testing for Android applications”，ISSTA，Pages 390–393，2019。
2. 国家重点研发计划：信息产品及科技服务集成化众测服务平台研发与应用 (2018YFB1403400)，2019-2021
3. 国家自然科学基金（重大项目）：软件开发中海量信息的融合反馈机制与支撑平台（61690201），2017-2021
4. 国家自然科学基金项目（面上项目）：协作式众包测试报告分析与融合技术研究（61772014）, 2018-2021
5. 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
6. 中央高校基本科研业务费专项资金资助项目：基于群智协同的众包测试技术（14380021），2020-2020
7. 南京大学技术创新基金项目：基于 AI 中台的移动应用测试技术研究与实现（14913413），2020-2020

致　　谢

在论文完成之际，我要向在系统开发与论文编写过程中，对我进行指导和帮助的人，表达最诚挚的感谢。

首先，在当前新冠肺炎疫情肆虐的特殊时期，我想感谢奋斗在抗疫一线的医务工作者与社区基层干部。是他们夜以继日、舍生忘死的大无畏精神，才让疫情得到了快速的控制，才让我们有了安全、安心环境进行工作。在这里我想对所有奋斗在抗疫一线的医务工作者与社区基层干部表达我最诚挚的感谢，愿你们平安归来。

其次要感谢我的导师陈振宇老师，感谢他在研究生这三年以来对我的指导与照顾。我从大四年级就进入 iSE 实验室进行慕测平台相关的开发工作。这三年多对我来说是十分宝贵的经历。通过在 iSE 实验室与慕测科技的开发经历，让我对于软件工程有了更加深入的体验与理解，让我对软件开发这件事情有了自己的体会，更让我对于今后的发展方向有了深入的思考。很高兴能在 iSE 实验室度过了自己人生关键的三年时光。在毕业论文编写阶段，由于新冠肺炎疫情的影响，我们都只能在家里进行论文的写作工作。虽然如此，陈老师还是通过多种途径督促我们的工作，对我们的论文进行多方面的指导与帮助，提出了许多宝贵的意见。正因如此，我才能按时完成论文。我还想感谢慕测公司的黄勇先生。正是他在项目研发上对我们毫无保留的指导与帮助，才能让我如此快速的成长。让我了解到了软件工程领域的多姿多彩。

同时还要感谢我在研究生阶段的三位室友，赵斯蒙、田元汉和王智成。他们陪伴了我研究生阶段的大部分时间，感谢他们在学习、生活、求职上对我的无私帮助，他们使我的研究生生涯更加的完满。我还要感谢 iSE 实验室 2017 级的朋友们。他们大部分已经在去年毕业，但是他们还是通过各种方式对我提供着帮助。特别是陈圣超与唐珊珊，他们在研究生阶段对我提供了大量的帮助与鼓励，这些帮助与鼓励都是我拼搏奋斗的不懈动力。

这里需要特别感谢我的父母和我女朋友张可欣。他们是我强大的后盾与精神支柱，他们始终是我努力工作、积极生活的动力源泉，他们所给予我所有的爱与关怀，是任何人和事情都无法替代的。

最后，对能在百忙之中抽出时间审阅我论文的各位老师、专家表示由衷的感谢!