



南京大學

NANJING UNIVERSITY

研究生畢業論文

(申請工程碩士學位)

論文題目 迭代式安卓應用自動化測試系統的設計與實現

作者姓名 徐悠然

學科、專業名稱 工程碩士（軟件工程領域）

研究方向 軟件工程

指導教師 陳振宇 教授 房春榮 助理研究員

2020年5月23日

学 号 : MF1832202

论文答辩日期 : 2020 年 5 月 23 日

指导教师 :  

The Design and Implementation of Iterative Android Application Automated Testing System

By

Youran Xu

Supervised by

Professor Zhenyu Chen, Assistant Research Fellow Chunrong Fang

A Thesis

Submitted to the Software Institute

and the Graduate School

of Nanjing University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Engineering

Software Institute

May 2020

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：迭代式安卓应用自动化测试系统的设计与实现

工程硕士（软件工程领域）专业 2018 级硕士生姓名：徐悠然

指导教师（姓名、职称）：陈振宇 教授 房春荣 助理研究员

摘 要

随着移动应用开发周期的不断缩短，如何提升移动应用质量保障的效率成为测试界的一大难题。一方面，部分公司将测试任务派发给志愿者进行人工测试。然而，志愿者的报告水平参差不齐，真正有用的“用户操作流程”这一过程本身，通常会被忽视。另一方面，一部分测试人员意图开发自动化测试工具以减少测试上人力资源与时间资源的消耗。但是，由于工具缺少人类拥有的测试知识，它们的实际效果仍然无法赶超人工测试。如果将包含人类测试知识的“用户操作流程”引入自动化测试工具，理论上可以提高自动化测试的效果。

本文设计与实现了一种迭代式安卓应用自动化测试系统，对测试用户的用户操作流程进行自动化的记录、提取与融合，并输入到测试工具中。通过流程上的迭代：用户操作信息引入工具，工具输出结果传递回用户产生下一轮用户操作信息；以及运行中的迭代：开展与一般自动化测试工具的一次性测试流程不同的多轮迭代式测试流程，来增强系统的测试效果。系统通过对 Appium 框架进行修改，从而于后台自动化的获取用户操作流程。前端使用 Angular2 完成 web 页面，令测试人员可以轻松使用本系统，后端则使用 SpringBoot 框架完成各项控制。通过 ADB(Android Debug Bridge) 工具和 UiAutomator 框架与接入系统的待测设备进行联系与控制。最后通过 Echarts 框架与 Dot 工具对覆盖流程进行绘制，让测试人员理解测试工具的覆盖情况。为了验证本系统的可用性，本文设计了一场实验，对待测应用使用 Jacoco 框架进行插桩，从而获取到应用运行时被覆盖的代码行数占总代码数的比例。

本文选取了 10 款知名移动应用与 50 份用户操作流程开展了一场实验，并与知名工具 Monkey 的进行对比。实验结果表明，系统的测试结果相较于未引入用户信息时得到了显著的提高，在测试时间为一小时的情况下，平均代码覆盖率得到了 13.98% 的提升，到达了 37.83%，这一结果同样超过了相同条件下 Monkey 的平均代码覆盖率 28.90%。并且，引入信息后的测试结果完全包含了用户或工具单独测试时的覆盖情况，没有产生覆盖遗漏，证明了本系统的可用性。

关键词：移动应用测试，安卓应用 GUI 测试，安卓自动化测试，自动化测试框架，自动化测试工具

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 Iterative Android Application Automated Testing System

SPECIALIZATION: Software Engineering

POSTGRADUATE: Youran Xu

MENTOR: Professor Zhenyu Chen, Assistant Research Fellow Chunrong Fang

Abstract

With the continuous shortening of mobile application development cycles, how to improve the efficiency of mobile application quality assurance has become a major problem in the testing community. On the one hand, some companies send test tasks to volunteers for manual testing. However, the level of volunteer reporting varies, and the truly useful "User Operation Flow" itself is often overlooked. On the other hand, some testers intend to develop automated testing tools to reduce the consumption of human and time resources on testing. However, due to the lack of human-owned testing knowledge, the actual effect of the tools cannot surpass manual testing. If the "User Operation Flow" containing human testing knowledge is introduced into an automated testing tool, the effect of automated testing can theoretically be improved.

This paper designs and implements an iterative Android application automated testing system, which automatically records, extracts, and fuses the user operation flows of the test users, and enters them into the testing tool. Through iterations in testing process: user operation information is introduced into the tool, and the output of the tool is passed back to the user to generate the next round of user operation information; and iterations in operation: multiple rounds of iterative testing that are different from the one-time testing process of general automated testing tools process, to enhance the test results of the system. By modifying the Appium framework, the system automatically obtains the user operation flows in the background. The front-end uses Angular2 to complete the web page, so that testers can easily use the system, and the back-end uses the SpringBoot framework to complete various controls. The ADB (Android Debug

Bridge) tool and UiAutomator framework are used to contact and control the device under test connected to the system. Finally, the coverage process is drawn through the Echarts framework and the Dot tool, so that testers can understand the coverage of the test tool. In order to verify the usability of this system, an experiment is designed in this paper to use the Jacoco framework for instrumentation of the application under test, so as to obtain the ratio of the number of code lines covered by the application runtime to the total number of codes.

This paper selected 10 well-known mobile applications and 50 user operation flows to conduct an experiment, and compared with the well-known tool Monkey. The experimental results show that the test results of the system have been significantly improved compared to the case without the introduction of user information. When the test time is set to one hour, the average code coverage rate reaches 37.83 %, which exceeds Monkey under the same conditions, which average code coverage is 28.90 %. In addition, the test results after the introduction of the information completely included the coverage of the user or the tool when it was tested alone, and no coverage omissions were generated, which undoubtedly proved the usability of the system.

Keywords: Mobile testing, Android GUI testing, Android automated testing, automated testing framework, automated testing tools

目录

| | |
|----------------------------|-----------|
| 表 目 录 | viii |
| 图 目 录 | x |
| 第一章 引言 | 1 |
| 1.1 选题背景与研究意义 | 1 |
| 1.2 国内外研究现状 | 3 |
| 1.2.1 学术界现状 | 3 |
| 1.2.2 工业界现状 | 4 |
| 1.3 本文主要研究工作 | 5 |
| 1.4 本文的组织结构 | 6 |
| 第二章 相关概念与技术 | 7 |
| 2.1 自动化软件测试与安卓自动化测试 | 7 |
| 2.2 安卓自动化测试相关技术 | 8 |
| 2.2.1 UiAutomator | 8 |
| 2.2.2 ADB | 8 |
| 2.3 安卓自动化测试框架 Appium | 9 |
| 2.4 安卓自动化测试工具 | 11 |
| 2.4.1 基于随机探索策略的工具 | 11 |
| 2.4.2 基于模型探索策略的工具 | 12 |
| 2.4.3 体系化探索策略的工具 | 12 |
| 2.5 代码覆盖率工具 Jacoco | 13 |
| 2.6 本章小结 | 15 |
| 第三章 需求分析与概要设计 | 16 |
| 3.1 系统整体概述 | 16 |
| 3.2 系统需求分析 | 18 |

| | | |
|------------|--------------------------|-----------|
| 3.2.1 | 功能性需求 | 18 |
| 3.2.2 | 非功能性需求 | 20 |
| 3.3 | 系统用例图 | 21 |
| 3.3.1 | 用例图 | 21 |
| 3.3.2 | 用例总表 | 22 |
| 3.3.3 | 用例描述 | 22 |
| 3.4 | 系统设计与模块设计 | 27 |
| 3.4.1 | 系统总体框架设计 | 27 |
| 3.4.2 | Appium 管理模块设计 | 30 |
| 3.4.3 | 自动化脚本流程获取模块设计 | 31 |
| 3.4.4 | 自动化测试模块设计 | 32 |
| 3.4.5 | 用户操作流建模模块设计 | 34 |
| 3.5 | 用户操作流设计 | 35 |
| 3.5.1 | 用户操作流数据化 | 35 |
| 3.5.2 | 用户操作流融合生成应用控件覆盖树 | 36 |
| 3.6 | 数据库设计 | 37 |
| 3.7 | 本章小结 | 39 |
| 第四章 | 详细设计与实现 | 40 |
| 4.1 | Appium 管理模块详细设计与实现 | 40 |
| 4.1.1 | Appium 管理模块概述 | 40 |
| 4.1.2 | Appium 管理模块详细设计实现 | 40 |
| 4.2 | 自动化脚本流程获取模块详细设计与实现 | 44 |
| 4.2.1 | 自动化脚本流程获取模块概述 | 44 |
| 4.2.2 | 自动化脚本流程获取模块实现 | 45 |
| 4.3 | 自动化测试模块详细设计与实现 | 48 |
| 4.3.1 | 自动化测试模块概述 | 48 |
| 4.3.2 | 自动化测试模块实现 | 49 |
| 4.4 | 用户操作流建模模块详细设计与实现 | 55 |
| 4.4.1 | 用户操作流建模模块概述 | 55 |
| 4.4.2 | 用户操作流建模模块实现 | 55 |
| 4.5 | 本章小结 | 60 |

| | |
|----------------------------|-----------|
| 第五章 系统测试与实验分析 | 61 |
| 5.1 测试环境 | 61 |
| 5.1.1 功能测试环境 | 61 |
| 5.1.2 实验待测应用信息 | 62 |
| 5.2 测试过程和实验设计 | 63 |
| 5.2.1 功能测试 | 63 |
| 5.2.2 实验设计 | 65 |
| 5.3 测试结果 | 66 |
| 5.3.1 功能测试结果与系统展示 | 66 |
| 5.3.2 实验结果 | 69 |
| 5.3.3 实验结果分析 | 71 |
| 5.4 本章小结 | 74 |
| 第六章 总结与展望 | 75 |
| 6.1 总结 | 75 |
| 6.2 展望 | 75 |
| 参考文献 | 77 |
| 简历与科研成果 | 83 |
| 致谢 | 84 |

表 目 录

| | | |
|------|------------------------|----|
| 3.1 | web 交互服务功能性需求列表 | 18 |
| 3.2 | Appium 管理模块需求列表 | 19 |
| 3.3 | 自动化脚本流程获取模块需求列表 | 19 |
| 3.4 | 自动化测试模块需求列表 | 20 |
| 3.5 | 操作流建模模块需求列表 | 20 |
| 3.6 | 非功能性需求列表 | 20 |
| 3.7 | 迭代式安卓自动化测试系统用例总表 | 22 |
| 3.8 | 新建待测应用用例描述 | 23 |
| 3.9 | 申请自动化测试任务用例描述 | 23 |
| 3.10 | 审核自动化测试任务用例描述 | 24 |
| 3.11 | 上传用户操作流用例描述 | 24 |
| 3.12 | 执行自动化脚本用例描述 | 25 |
| 3.13 | 执行自动化测试任务用例描述 | 25 |
| 3.14 | 用户操作流记录用例描述 | 26 |
| 3.15 | 用户操作流融合用例描述 | 26 |
| 3.16 | 查看测试结果用例描述 | 27 |
| 3.17 | 待测应用表数据结构 | 38 |
| 3.18 | 测试任务表数据结构 | 39 |
| 3.19 | 测试结果表数据结构 | 39 |
| 4.1 | Appium 管理模块主要类 | 42 |
| 4.2 | 控件相关元素表 | 45 |
| 4.3 | 自动化脚本流程获取模块主要类 | 47 |
| 4.4 | 自动化测试模块主要类 | 51 |
| 4.5 | 用户操作流建模模块主要类 | 56 |
| 5.1 | 测试环境 | 62 |
| 5.2 | 待测应用信息 | 62 |

| | | |
|-----|---------------------|----|
| 5.3 | 申请自动化测试任务测试用例 | 63 |
| 5.4 | 审核自动化测试任务测试用例 | 63 |
| 5.5 | 执行自动化脚本测试用例 | 64 |
| 5.6 | 自动化测试任务执行测试用例 | 64 |
| 5.7 | 查看自动化测试结果测试用例 | 65 |
| 5.8 | 测试用例执行结果 | 66 |
| 5.9 | 实验结果 | 70 |

图 目 录

| | | |
|------|--|----|
| 2.1 | adb 框架关系逻辑 | 9 |
| 2.2 | Appium 运行原理 | 10 |
| 2.3 | Jacoco 详细代码覆盖结果样例 | 13 |
| 2.4 | Jacoco 部分代码覆盖率结果 | 14 |
| 2.5 | Jacoco 插桩原理图示 | 15 |
| 3.1 | 迭代式安卓自动化测试系统整体流程概述 | 17 |
| 3.2 | 迭代式安卓自动化测试系统用例图 | 21 |
| 3.3 | 迭代式安卓自动化测试系统部署图 | 28 |
| 3.4 | 迭代式安卓自动化测试系统框架图 | 29 |
| 3.5 | Appium 管理模块流程图 | 30 |
| 3.6 | 自动化脚本流程获取模块流程图 | 32 |
| 3.7 | 自动化测试模块流程图 | 33 |
| 3.8 | 用户操作流建模模块流程图 | 35 |
| 3.9 | 系统数据库关键实体关系图 | 38 |
| 4.1 | Appium 管理模块时序图 | 41 |
| 4.2 | Appium 管理模块主要类图及部分方法 | 43 |
| 4.3 | AppiumManager 类的启动 Appium 方法核心代码部分 | 44 |
| 4.4 | 自动化脚本流程获取模块时序图 | 46 |
| 4.5 | 自动化脚本流程获取模块类图及部分方法 | 47 |
| 4.6 | RemoteWebElement 类的 click 方法内容修改及注释说明 | 48 |
| 4.7 | 自动化测试模块时序图 | 49 |
| 4.8 | 自动化测试模块类图及部分方法 | 51 |
| 4.9 | DFSRunner 类的 DFS 测试算法以及核心代码 | 53 |
| 4.10 | OperationProcessRecurrenter 类的复现用户操作方法以及核心代码 | 54 |
| 4.11 | 用户操作流建模模块时序图 | 55 |
| 4.12 | 用户操作流建模模块类图及部分方法 | 57 |

| | |
|--|----|
| 4.13 widgetMergeTools 类的将操作流进行融合 getResList 方法 | 59 |
| 4.14 DOTDrawer 类的生成控件覆盖树图片 resList2Pic 方法 | 60 |
| 5.1 上传任务页面 | 66 |
| 5.2 审核任务页面 | 67 |
| 5.3 上传用户操作流页面 | 67 |
| 5.4 测试结果页面 | 68 |
| 5.5 页面信息页面 | 68 |
| 5.6 控件信息页面 | 69 |

第一章 引言

1.1 选题背景与研究意义

近十年以来，随着互联网的普及，移动设备也随之得到了大量的推广，它们甚至已经逐渐取代了传统计算机在人们生活中的地位。根据 2019 年中国互联网发展报告发布的内容，截至 2018 年底，我国的网民数量已经超过了 8.29 亿，而网民中使用智能手机的人数高达 8.17 亿，这代表着使用智能手机上网的网民占总网民数量的比例达到了 98.6% 之高 [1]。移动用户数量的爆发式增长，使得人们对移动应用的需求也得到了海量的提升。在大量的移动设备系统中，安卓系统由于开源、应用发布简单等因素独占鳌头，根据市场研究机构 IDC 发布的智能手机市场预测报告，2019 年，安卓系统的市场份额已经高达 87% [2]。

正因如此，如何保障安卓应用的质量成为了一个巨大的难题。随着安卓移动应用之间的不断比拼，应用质量的需求也跟着水涨船高，换言之，移动用户对于应用的眼界也随之不断增长。人们不仅仅关注于应用的各项基本功能，对于性能、用户体验等其他各方面也变得更加重视。一旦应用出错，用户对应用的评价也会随之降低 [3]，而如果评价过低，应用甚至可能会被谷歌商店移除 [4]。因此，安卓应用从开发到发布这一系列过程中的质量保障，就变得尤其重要。

目前的主流软件测试分类主要分为三个部分：单元测试、接口测试和 UI (User Interface) 测试，对于安卓应用而言，单元测试和接口测试的方法已经较为稳定，现有的 PC 端、WEB 端框架在经过修改后已经能够很好的满足安卓应用对于这两种测试的需求，但是 UI 测试则有所不同。安卓应用测试的 UI 测试被称为 GUI (Graphic User Interface, 图形化用户界面) 测试，这部分的测试主要是为了解决安卓框架的碎片化所导致的各方面问题。虽然所需投入的测试占比低于前两种测试方式，但是重要性很高，可以说必不可缺 [5]。随着应用迭代的速度越来越快，测试人员在进行手动 GUI 测试时，需要进行大量的重复性工作来达到回归测试的目的 [6] [7]。但是由于测试人员的主观性较强，每一次人工测试中重复性的工作很难做到完美的复现。同时，测试的准确性也极度依赖于测试人员本身的知识水平，如果测试人员的水平不足，很有可能产生提交的缺陷在尝试复现时难以成功的情景。为了优化 GUI 测试的成本以及执行效率，开发者们更希望使用安卓自动化测试框架（目前安卓自动化测试主要指代安卓 GUI 自动化测试，因此本文之后的部分如若提到安卓自动化测试，均指代安卓 GUI 自动化测试）和安卓自动化测试工具来完成测试工作 [8]。

其中，安卓自动化测试框架，是指由专业的技术人员编写一份能够借由框架进行重复性执行的脚本 [9]。一旦应用出现改动需要进行回归测试时，只需要对脚本进行少量的修改就能完美的对测试流程进行复现。目前比较流行的自动化框架有 Appium [10] [11]、Robotium [12] [13] 等。这样的脚本虽然编写起来需要一定的知识与时间，而且会根据编写者水平的高低达到不同的效果，但是一经完成，就能够保证编写内容的覆盖。工业界的许多测试平台，也偏向于使用自动化测试框架来进行安卓自动化测试。目前的大型平台有华为云测试¹、百度 MTC²、腾讯 WeTest³、阿里云 MQC⁴等等。这些测试网站的测试选项中，几乎都包含了添加自动化测试脚本的功能，可以由用户自行添加，也可以聘请企业的专家来帮忙编写。然而，这些网站在自动化测试工具的选择上却较为匮乏，部分网站甚至不提供自动化测试工具遍历的选项，其中有提供该选项的网站也只是选用了谷歌官方提供的一款，主要用于压力测试而非功能性测试的随机点击型测试工具 Monkey [14]。本系统所选用的安卓自动化框架为 Appium 框架，理由是由于该框架目前在业界排名靠前，口碑良好，且使用简单。

而另一种方式的安卓自动化测试工具的思路是开发出一款自动化的工具，无需任何人工操作就可以全自动的对一款应用进行遍历。这种应用遍历的方式通过达到较高的应用覆盖率，即通过对应用内容的大面积覆盖，来触发应用中可能出现的 BUG（系统漏洞），最终记录下结果，交由测试人员复现与修改。从 2011 年开始，学术界就出现了许多关于安卓自动化测试工具的研究，如 Dynodroid [15]，Sapienz [16],PUMA [17],Evodroid [18],Stoat [19],A3E [20] 等等。这些自动化测试工具确实可以对应用控件做到事无巨细的覆盖，只要到达了新的页面，该页面上的控件一定能够完全被记录下来。但是，由于弹窗问题、回退问题、控件组合型问题等特殊问题的产生，自动化测试工具即使能够发现大部分的控件，却很难做到将它们全部实际的覆盖 [21] [22] [23] [24]。近年来已有的相关研究，大多只是将目标放在致力于解决众多自动化工具问题中的一种，只是提出对以往工具的一些小的改进。这些工具的论文中所做出的对照性实验，也通常只是关注自己工具的覆盖率相较于其他的工具是否有所提升。但是从宏观上观察这些工具的覆盖率，相较于人工测试所能达到的数值而言，依旧相差甚远。同时，大部分工具的测试流程往往都是一次性的，对于一次测试流程中，被发现却由于各种问题无法覆盖到的控件以及分支路径，这些工具并没有将它们进行重复式利用，来增强自己的覆盖率。

¹<https://auth.huaweicloud.com/>

²<http://mtc.baidu.com/>

³<https://wetest.qq.com/>

⁴<https://www.aliyun.com/product/mqc>

对比自动化测试框架和自动化测试工具的差异，我们不难发现，使用自动化测试框架编写自动化脚本，虽然需要专业知识以及一定的时间，但是完成的脚本一定能够覆盖到编写内容所涉及所有功能。不过，自动化脚本终究是对人类手工测试的模拟，一旦编写者自身漏过了一些细节，脚本自然也无法复现并覆盖这部分应用内容。而自动化测试工具在理论上可以遍历应用的绝大多数控件。对于来到新的页面后发现并保存下来的控件，一定是该页面存在的所有控件，只不过现今的技术仍不完善，并不能做到完美的覆盖。如果我们学习人机结合的思想 [25] [26]，将自动化脚本，乃至人工手动操作中的用户操作流程信息进行提出，并将这些信息结合导致自动化测试工具中，并以一种迭代式的方式来进行自动化测试，让自动化测试工具在了解到待测应用的基本框架后，再去进行多轮迭代式的测试，势必能够使得自动化测试工具的覆盖率得到一个较大的提升。同时，如果让自动化工具能够输出自己的覆盖情况，交由测试人员去进行新一轮的人工测试流程，并进行记录，就可以将这些信息再次输入到工具中，使得测试覆盖率得到一种迭代式的、螺旋形的上升。

本文致力于实现一种基于用户操作流，令工具本身对应用控件框架产生一定了解，从而优化自身测试路径的迭代式安卓应用自动化测试方法，并将这种测试方法扩展为一个测试系统，供测试人员使用。其中，用户操作流指代一次完整的测试路径信息。无论是自动化测试脚本、人工手动测试、抑或是自动化测试工具本身，他们的测试流程都可以被抽象为一种用户操作流，这种用户操作流通过格式化任意来源的测试流程进行获取。它可以为自动化测试工具提供应用控件的可达性路径分析，使得原本自动化测试工具无法到达的场景，在经由路径复现后成功到达。同时，新一轮的测试流程本身也可以抽象为用户操作流，作为迭代信息进行保存，对整个测试过程进行优化。通过引入用户操作流，系统可以保证自身覆盖待测应用的大部分功能模块，同时在到达大部分模块的基础上进行自动化测试，从而覆盖模块的细节，使得测试结果得到大幅度的提升。而由于每一次的路径都会被记录并迭代式的用于下一轮测试，系统可以保证测试路径不会产生丢失。

1.2 国内外研究现状

1.2.1 学术界现状

学术界对于安卓自动化测试的研究，主要着眼于自动化测试工具的发明：

TesterHome 的黄延胜 (思寒) 提出了一种可以通过接受特殊配置文件，引导工具在特定页面对特定控件如何进行操作的自动化测试工具 AppCrawler [27]。这款工具正如其名，会像网页爬虫一样对待测应用的内容进行爬取。本系统的

DFS 测试逻辑思路就受到了该工具的启发。但是配置文件的内容有着强指定性，一旦待测应用中有另一处页面无法靠工具本身的逻辑跨越，则必须对配置文件进行针对性的修改。目前学术界的大部分自动化测试工具也都采用了这种配置文件的方式，通过编写文件内容引导自身跨越一些如输入账号、输入密码等必须拥有相关知识才能通过的页面。

南京大学的 Xiujiang Li, Yanyan Jiang 等人提出了一款可以融合人工操作的自动化测试工具 [28]。这款工具要求测试人员对待测应用进行一个简单的遍历，不需要测试到应用的详细功能，但是需要通过点击到达应用的大部分界面，即覆盖应用的基础框架。接着工具会通过 Reran [29]，一款通过安卓底层事件日志来记录和复现用户的操作的工具，对测试人员的操作进行复现，并在测试人员停顿较长时间的点、或是页面 Activity 发生改变的点，重新启动自动化逻辑，从而增加应用覆盖率。本系统对于将用户操作流与工具进行融合的思路就受到了该工具的启发。但是这款工具由于使用的是底层日志，即一种记录坐标的方式对操作进行复现，只能在测试人员进行操作的手机上开启后续自动化测试流程。同时，文章内没有解释如果出现了回退问题，即待测应用的上一个页面无法通过系统自带的返回键、或是页面内的返回键来进行返回时，工具该如何处理。

Facebook 的 Ke Mao, Mark Harman 等人同样提出了一款可以融合人工操作的自动化测试工具 [30]。这款工具同样使用 Reran 记录下了测试人员们的操作流程，接着，这款工具会将操作流程通过进化算法进行处理，转化为可以输入到另一款工具 Sapinz [16] 中的配置文件。通过这种方式获得配置文件，工具能够覆盖到更多的应用部分。但是，使用 Reran 带来的问题同样存在，那就是获得的操作流，因为从像素层面获取，如果在不同分辨率的手机上进行复现，很大概率会产生错误。因此，这些人工的操作流程，必须要在相同款式的手机上进行复现才能够大概率得到成功。

1.2.2 工业界现状

相较于学术界的研究，工业界更倾向于使用现有的自动化测试框架。目前国内的移动测试平台包括华为旗下的华为云测试、百度旗下的 MTC、腾讯旗下的 WeTest、阿里旗下的 MQC、云测 Testin 等，国外的测试平台则包括 Test Cloud⁵，Testdroid Cloud⁶等。这些自动化测试平台虽然包括很多测试服务，但是很大一部分是提供自己持有的机型供测试人员使用，和自动化测试工具相接轨的服务却很少。与自动化测试相关的服务大多是要求用户上传一份针对应用的自动化测

⁵<https://www.test-cloud.com/>

⁶<https://bitbar.com/>

试脚本，或者支付费用，由平台专家进行编写，其中，阿里使用 Appium 框架，腾讯使用自研框架。在实际对这些网站的相关功能进行了解后，作者发现，有使用到自动化测试工具的平台，只有百度 MTC 与腾讯 WeTest，并且，这两个测试网站也只是将 Monkey，一款主要用于压力测试的工具加入深度遍历逻辑中。

腾讯的 Dong Zhen 等人提出了一种记录应用关键状态的工具 [31]，它可以在自动化测试陷入一种循环时，主动返回到之前被记录下的最活跃的页面，从而打破循环，继续进行测试。但是该工具也并没有被实际使用在 WeTest 服务中，只是作为一篇学术研究进行了发表。

1.3 本文主要研究工作

虽然目前学术界提出了大量移动应用自动化测试工具，但是根据一些综合性的研究 [32] [33] [34]，这些工具几乎都存在着两种问题：其一，现存的工具仍然无法完全解决大部分人类可以轻易解决的问题。即使一些工具提出了改进方式，也只是针对这些问题中的其中一点，而其余的部分仍然难以解决，并且，由于工作量过大的原因，现今也没有人提出一种将所有工具的改进全部集合起来的工具。其二，现存的工具大部分都只是进行一轮测试流程就得出测试结果，这种情况下，流程中因为控件优先级而被忽略的低优先级的页面会被永远的遗弃。只有进行迭代式的多轮测试，才能够将这些低优先级的页面与控件完全覆盖。

本文所设计以及实现的迭代式安卓自动化测试系统，针对上述所提到的两个问题提出以下解决方案：一方面，通过对自动化脚本、手工测试、甚至自动化测试工具本身的测试流程进行记录、格式化以及融合，建立用户操作流模型来帮助工具了解应用控件框架，同时根据操作流引导工具到达大部分应用页面，解决工具由于缺少相关知识而无法跨越部分页面的问题。同时，本系统可以将自身的测试流进行输出，交由测试人员与应用进行比对，令测试用户生产的全新测试流程更加具有针对性，从而达到迭代式运行的目的。另一方面，通过迭代式运行，在未覆盖完被发现的控件，但第一轮测试已经结束的时间点，开启第二轮测试，优先覆盖之前未被覆盖的控件，重复运行直至所有被发现的控件都被覆盖，从而解决控件遗漏的问题。通过两方面的增强，本文所提出的系统可以保证自身的覆盖率不会低于输入的用户操作流与原本工具测试逻辑的覆盖率之和，并且还有很大的机会进入新的页面，从而覆盖更多的控件，达到更高的应用覆盖率，令测试结果得到大量提升。

对于增强的结果，我们采用两种指标对系统进行评估，分别是 AC (Activity Coverage) 与 CC (Code Coverage)。AC 是系统在一轮测试中，所覆盖的页面数量占页面总数的比例。CC 则是系统在一轮测试中，所覆盖的代码数量占总代码

数的比例，本系统通过使用 Jacoco [35] 对应用进行插桩来获取应用的覆盖情况。CC 可以分为六种覆盖率，包含行覆盖率、方法覆盖率等等，在本文开展的实验中，我们选择粒度最小的桩覆盖率作为代码覆盖率基准，由于粒度最小，这种覆盖率的数值在六种覆盖率中是最低的，但也是最精准的。本文会收集一部分人工操作流程并格式化为系统所需的用户操作流，将未输入操作流时工具的覆盖率、输入操作流后应用的覆盖率、以及 Monkey，一款主流测试工具的应用覆盖率进行比较，来证明本系统的确实可用。

1.4 本文的组织结构

本文描述了迭代式安卓自动化测试系统的需求分析与设计、具体实现与实验验证，本文的组织结构共分为六个章节，如下所示：

第一章，引言。本章首先介绍了本文的研究背景以及研究意义，接着介绍了国内外的移动自动化测试相关研究，并将本系统与其进行了比较，最后对本系统的主要研究内容进行了简单的介绍，并给出验证本系统可用性的方案。

第二章，相关概念以及技术。本章首先介绍了安卓自动化测试的概念，接着介绍了一系列本系统相关的技术，包含 Appium、ADB、UiAutomator 等，然后讲解了安卓自动化测试工具的三种分类，最后介绍了本系统的验证部分所使用的测试框架 Jacoco。

第三章，迭代式安卓自动化测试系统的需求分析与概要设计。本章首先通过一张概述流程图介绍了系统的使用流程，接着以模块作为分割，介绍了系统的功能性需求以及非功能性需求，并配合系统用例图与系统用例描述讲解了系统的使用方法。最后，本章对本系统的主要模块设计以配合模块流程图的方式进行了一一讲解，并说明了数据库的设计方式。

第四章，迭代式安卓自动化测试系统的详细设计与实现。本章与第三章描述的模块设计相呼应，从模块的详细设计与实现方式入手，配合时序图、类表、类图以及关键方法的代码等内容，对 Appium 管理模块、自动化测试脚本流程获取模块、自动化测试模块以及应用建模模块进行了详细的说明与讲解。

第五章，迭代式安卓自动化测试系统的测试结果与实验设计。本章首先抛出三个问题作为系统验证的切入点，要求测试与实验内容要能够回答出这三个问题。接着介绍了测试的硬件环境、待测应用信息以及系统的功能测试设计和实验设计方案。最后，本章给出了包含系统演示的功能测试的结果以及包含覆盖率信息的实验结果。

第六章，总结与展望。本章主要总结了本文的整体概述与总结，并为本系统的未来修改与扩展提出了进一步的展望。

第二章 相关概念与技术

2.1 自动化软件测试与安卓自动化测试

自动化软件测试是指，将原本需要以人类为主体进行的测试行为，转化为由机器执行的一种过程。换言之，就是通过编写一份能够模拟人类操作的脚本来代替人工执行，从而实现对软件的自动化测试 [36]。由于人工测试的准确性完全依附于测试者的主观意识，即使规定好了相同的测试路径，测试者依旧有可能会根据不同的理解进行不同的测试方案。而对自动化测试而言，每一次的执行必然和以往是一摸一样，丝毫不差的。同时，由于软件开发的周期随着市场的激烈竞争逐渐缩短，人工测试的速度和开销都无法跟上快速的开发，因此，自动化测试能够节约更多的成本以及时间，并且保证质量不会下降 [37]。

使用自动化测试的方案进行软件测试，可以节约大量人力，在减少人工测试的开销的同时节约人工测试的时间，同时能够保证每一次的回归测试流程都可以保持一致。由于自动化测试脚本拥有如上所述的一致性，待测软件的可信度也会因此而随之逐渐提升，可以说是当今测试界的一剂良药。对于安卓自动化测试而言，现今使用的自动化测试技术主要分为自动化测试框架以及自动化测试工具两种。

自动化测试框架是指，拥有相关经验的专业测试人员，根据自动化测试框架暴露给测试人员的接口，针对某一款特定的应用编写特定的自动化测试脚本。每一份脚本的内容实际都是模仿人工测试的操作，一旦出现了软件本身进行修改的情况，需要进行回归测试，则只需要修改少量代码即可立刻复用。但是，这也需要脚本编写者有着深厚的代码功底。由于安卓系统的碎片化，以及安卓应用的不稳定性，脚本必须要有优秀的鲁棒性才能保证能够在每一次复用是都可以成功运行 [38] [39] [40]。反之，劣质的脚本即使在同一环境，同一版本下的待测应用上运行多次，也有可能无法每次都运行成功。

自动化测试工具是指，在不知道具体是什么应用的情况下，编写一款万能的工具，它对任何应用都能够根据自身的测试逻辑进行一定的覆盖。大部分的自动化测试工具都需要依附于某一种框架来进行与待测应用的交互，并做出如点击、滑动等相应的 UI 操作，也因此，我们可以把自动化测试工具看作是一种特殊的自动化测试脚本。但是由于每一款应用都有其特殊性，目前的自动化测试工具，还无法做到对任意一款应用都能直接做到完美覆盖。也正因如此，该领域有着大量的研究正在不断地进行着。一种常见的提升自动化测试工具覆盖率

的方法是，为工具输入一份由测试人员专门编写的针对某一款特定应用的配置文件，这也可以看作是一种简单的将测试脚本与测试工具结合的方式。

2.2 安卓自动化测试相关技术

2.2.1 UiAutomator

UiAutomator 是安卓官方开发的一款用于进行 UI 自动化测试的框架。它为开发者提供了一系列接口，可以用来对安卓应用进行一系列的自动化测试操作，比如点击、滑动、键盘输入、触摸、长按，甚至断言方法等等。由于它是在安卓 4 之后才被推出的，因此想要使用它，要求安卓手机版本至少大于安卓 4.3，最好是安卓 5 以上。

UiAutomator 被存放在安卓 SDK (Software Development Kit) 的工具文件夹中，包含两个工具来支持 UI 自动化测试，其一是 `uiautomatorviewer`，他是一个图形工具界面，用来扫描和分析应用的 UI 控件。通过对手机设备进行特殊的截图指令，它可以将设备当前界面上的所有布局信息和控件元素数据以快照的方式存储下来。比如控件 id (`resource-id`)、控件文本 (`text`)、控件所属类 (`class-name`)、控件所属包 (`package-name`)、控件描述信息 (`desc`) 等等。通过这些属性，我们可以利用 UiAutomator 提供的 API 对特定的拥有相关属性的控件或一系列控件进行定位，并进行操作。

另一个工具则是 UiAutomator 库，这是一个用于测试的 java 库，包含了创建 UI 测试的各种接口和执行自动化测试的引擎。通过调用这些接口，就可以对移动设备进行各种各样的操作，如点击、滑动、触摸、长按、输入等等。

UiAutomator 的优点在于：首先，他是官方推出的工具包，拥有优秀的稳定性和更新支持，并且能够获得更多的安卓权限。其次，它提供了丰富的定位和操作方法，包括通过控件属性定位元素、通过坐标定位元素以及点击、触摸等大量操作方法。最后，它可以跨应用进行操作，而其他的一些自动化框架并不能直接做到这一点。也正因如此，它被 Appium 选为进行安卓操作的工具，关于 Appium，在下一节中，我们会详细提到。

2.2.2 ADB

ADB 的全称为 Android Debug Bridge，也就是安卓调试桥，是安卓 SDK 自带的一款调试工具，顾名思义，是安卓设备和电脑进行连接与调试的桥梁。

ADB 采用的架构为常见的 C/S (Client/Server) 架构，它包含三个部分：client 客户端、server 服务端以及 daemon 守护程序。他们的相互关系如图 2.1 所示。

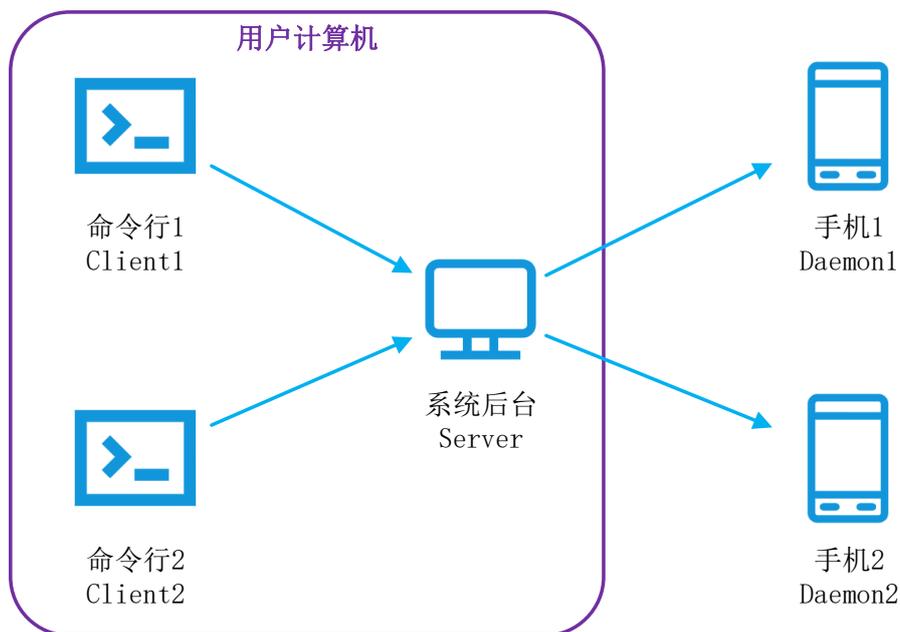


图 2.1: adb 框架关系逻辑

客户端运行于电脑端上，用于从 shell 或者脚本中执行 adb 命令。首先，客户端的 adb 程序会尝试定位是否存在一个 adb 服务端，如果没有，则会自动启动一个 adb 服务端，以等待与设备链接。

服务端则是运行在电脑端上的一个后台进程，它主要用于检测 USB 端口是否连接或者拔除了设备，或者安卓模拟器的启动与否。服务端会从客户端接受 adb 命令，并通过 usb 或者 tcp 的方式发送到对应的 adb 守护程序上。

守护程序又名 adbd (adb daemon)，他是一个在安卓设备后台运行的后台进程。他会连接运行在主机上的服务端，并提供相应的服务。

2.3 安卓自动化测试框架 Appium

随着安卓自动化技术的不断发展，一个又一个的自动化测试框架也被不断地提出。而 Appium 凭借着其入门简单，功能强大，以及跨平台性在各大自动化测试框架中独占鳌头。

Appium 是一款基于 Node.js 的，开源的移动应用自动化测试框架 [10]。它不仅仅支持安卓，同样支持 IOS 以及 FirefoxOS 的使用。正是这种跨平台性，使得 Appium 在众多自动化测试框架中，能够获得众人的认可。

由于继承了 Selendroid, Appium 支持大量的语言，如 Java、Python、Object-C、

JavaScript、PHP、Ruby、C#、Perl，都可以用来编写测试脚本。同样也归功于此，在实现了对 WebDriver 协议的扩展后，它可以同时用于原生、混合和移动 web 应用程序的测试。

Appium 的架构是常见的 C/S 架构，他的服务端是一个 Web 服务器，并对外提供了一套用于操纵手机的接口。客户端调用这些接口，从而能够在移动设备上点击、触摸、滑动等操作。在操作被手机执行完毕后，服务端会将执行的结果返回给客户端。

Appium 的运行原理如图 2.2 所示。简而言之，Appium 客户端发送申请给 Appium 服务端，Appium 服务端将申请解析为安卓设备可以理解的命令并进行执行，执行结束后，Appium 服务端将执行结果返回给 Appium 客户端，如此往复。

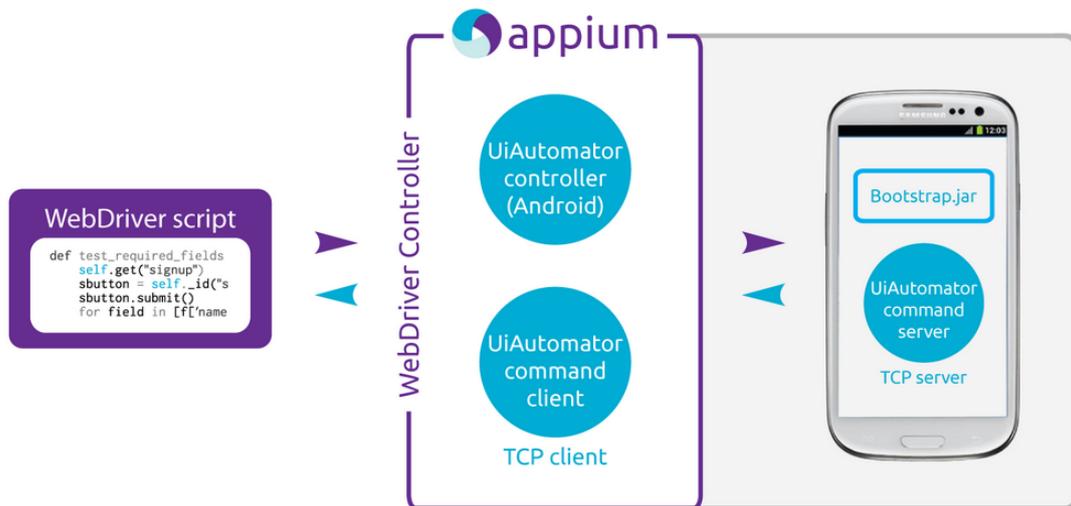


图 2.2: Appium 运行原理

首先，Appium 会在安卓手机上使用 adb 命令部署 BootStrap.jar，这是一个使用 java 书写的中间件，其本身是一个 UiAutomator 测试脚本，用来解析和处理 Appium 服务端和手机之间的交互。

接着，Appium 服务端会在电脑上打开两个特定的端口，一般为 4723 与 4724 端口。4723 端口用来接受客户端的请求，4724 端口用来与安卓设备通讯。BootStrap 会一直监听 4724 端口，当 Appium 客户端发出操作请求时，Appium 服务端接收到该请求，并把该请求通过端口传递给 BootStrap.jar。BootStrap 会调用 UiAutomator 的命令来执行，并将执行结果返回为 Appium 服务端。最后，Appium 服务端将执行结果返回给 Appium 客户端。

对于用户而言，他们只需要提交一份完整的测试脚本，而 Appium 客户端会运行这份包含了大量的安卓操作申请的脚本。在 Appium 客户端启动时，会同步初始化一个 Session，用来与服务端进行交互。之后，所有的自动化测试命令就都会在这一个 Session 中运行了。

本系统所使用的自动化测试算法，正是基于 Appium 所编写。Appium 可以轻松的获取到用户操作流中的控件信息，帮助系统轻易定位到每一个控件，从而在后续自动化流程中复现出相应的操作。这也是我们选择该框架的原因之一。

2.4 安卓自动化测试工具

常见的安卓自动化测试工具可以分为三类，他们根据自己的探索策略进行划分，分别是：随机探索策略、基于模型的探索策略以及体系化的探索策略 [41]。

2.4.1 基于随机探索策略的工具

随机探索策略，顾名思义，就是指所有的探索操作都是随机生成的。这种探索策略的自动化工具，会根据一种预先设定好的逻辑随机生成安卓应用输入。由于所有的输入都是随机生成的，所以该策略的最大优点就是测试速度十分快捷，在对安卓应用进行大量输入的情况下，安卓应用很有可能产生崩溃、卡顿等问题，这也是测试人员希望能够发现的问题。但是，由于安卓应用和网页类似，许多页面的深度较深，需要进行一系列有顺序的操作才能够进入，这就使得随机化策略难以进入这些页面。

比较著名的应用随机探索策略的自动化工具有 Monkey [14] 和 Dynodroid [15]。Monkey 是一款由安卓官方团队提供的命令行工具，该工具的名称来源于“无限猴子定理”，即让上帝创造无限只猴子，放在电脑面前随机进行打字，那么其中一定会有一只能够成功的打出莎士比亚的著作《哈姆雷特》。研究者们认为，对于测试而言该定理也能够成立：让一款工具一直长时间的随机进行点击，那么一定能够发现大部分的应用问题。Dynodroid 则是一款在相较于早年的 Monkey 具有更多功能的随机探索策略工具，它可以通过检查安卓框架中，哪些事件与应用程序有关来生成系统事件，从而让随机策略更有针对性。由于其发表的时间较早，也可以说是安卓自动化工具的开山鼻祖之一。

现如今，由于 Monkey 在不断的更新，其自身也吸收了近年来各种随机探索策略工具的优秀逻辑，随机探索策略中产生的问题逐渐被解决完毕。因此，研究者们已经不再着眼于研发随机探索策略的自动化工具，相关的工具发表在近年来也几乎消失，可以认为，该策略的工具之中，只有 Monkey 得到了留存，成为了唯一且高效的随机策略自动化工具。

2.4.2 基于模型探索策略的工具

基于模型的探索策略，主要是令工具在测试过程中为应用建模，在测试的过程中对模型不断完善，直至测试完毕。这些模型通常是将页面作为状态，操作作为过渡的有限状态机。比较常见的例子是从网页爬虫类工具得到灵感 [42]，并使用这种方式进行探索的工具。通常，这些工具会获取每一个页面中能够被点击的控件，将他们存入栈或队列中，接着根据自身逻辑中的顺序一个个进行处理。最简单的方式就是每到达一个新的页面，就将所有的控件信息进行存储，然后一个个的遍历下去，如果进入了新的页面，则将之前的数据保存，并在新的页面遍历完成后，回到上一个页面时，将存储的数据进行恢复，并接着继续遍历，直到所有收集到的控件全部被覆盖后终止。

这种方法的优点，其一在于每一次的测试路径，是一致的，除非遇到了会根据时间改变控件内容的应用，否则每一次一定会经过同一条测试路径，这给问题的复现带来的极大的便利。其二是在理论上，无论控件的深度有多深，该策略都可以将应用所有的控件都覆盖到。但是实际上，会有很多页面由于进入顺序不同的原因无法到达。本文所涉及的迭代式安卓自动化测试系统，就是基于该策略进行扩展的。

比较著名的该策略的自动化测试工具有 AppCrawler [27]、Puma [17]、EvoDroid [18] 等，该策略的自动化测试工具非常之多，2018 年的一篇综述论文对该类型的工具进行了整理，发现近几年至少有 65 款，甚至更多的该类型工具被发表，而相比之下，随机策略的工具只有 11 款 [33]。正因为该方法为目前自动化测试工具的主流方法，其思想概念十分成熟，所以本文系统所选用的自动化测试逻辑也使用了基于模型探索的策略。

2.4.3 体系化探索策略的工具

体系化探索策略的工具运用了一些更加复杂的技术，如使用符号执行或者进化算法来对尚未被覆盖到的应用部分进行探索。

这一方法的研究者较少，通常只能直接给出实验算法以及实验数据，而具体算法参数的选择方式则难以进行说明。如 Polaris [30] 就是一款体系化探索策略的工具，它将人们的操作记录下来，接着使用遗传算法对这些操作进行交叉互换，最终得到一组新的操作序列，并将这些操作序列引入到自研的自动化测试工具中开展实验。虽然这些方法起作用的原因较为难以解释，但是它的实验结果确实证明了该方法能够提高工具的应用覆盖率。

2.5 代码覆盖率工具 Jacoco

Jacoco 的全名是 Java Code Coverage，是一款 Java 代码的覆盖率计数工具 [35]。在我们对代码进行测试时，一个公认的测试结果评估标准就是代码覆盖率。测试脚本覆盖了越多的代码，就说明这次测试的效果越好，就越有可能找到更多的 BUG。但是，即使是如 Eclipse、IDEA 等功能强大的代码开发工具，也并不具备直接测量出在运行测试代码时，源码的覆盖率数值的功能，而让测试人员自己一行行的数被覆盖的代码显然也是不现实的，因此，一些代码覆盖率工具就诞生了。

Jacoco 正是这些 Java 代码覆盖率工具的其中之一。由于其使用起来十分便利，目前已经被集成到了如 Sonar、Jenkins 等许多第三方工具中。同时，Jacoco 也是目前最为成熟的安卓 GUI 测试代码覆盖率获取工具。Jacoco 的代码覆盖结果样例如图 2.3 所示，其最终呈现的结果十分容易理解，绿色代表被覆盖，红色则代表没有被覆盖，黄色多见于判断语句中，代表该分支的部分可能性被覆盖，而剩余的部分没有被覆盖。

```
390.     public void reloadFromBackStack() {
391.         pageFragmentManager.setTab(getCurrentTab());
392.         if (!pageFragmentManager.backStackEmpty()) {
393.             pageFragmentManager.loadFromBackStack();
394.         } else {
395.             loadMainPageInForegroundTab();
396.         }
397.     }
```

图 2.3: Jacoco 详细代码覆盖结果样例

测试结果也可以从整体覆盖率数值上进行观察，如图 2.4 所示，总共包含了六种尺度的覆盖率，分别是：

- 指令覆盖率 (Instructions): Java 字节码指令，也是 Jacoco 计算覆盖率的最小单位，同样是 Jacoco 插桩时插入的桩数量的总和。指令覆盖率表明了所有的指令中，哪些指令被执行过以及哪些指令没有被执行。
- 分支覆盖率 (Branches): 源码中 if, switch 等语句中的分支覆盖情况，他会计算代码中的总分支数量，以及没有覆盖到的分支数量，但是异常处理不被考虑在内。

- 圈复杂度 (Cxty, Cyclomatic Complexity): Jacoco 会为每一个非抽象的方法计算圈复杂度。根据官方定义, 圈复杂度指覆盖所有可能情况最少使用的测试用例数。
- 行覆盖率 (Lines): Java 源码中的每一行一般都会对应一条或者多条字节码指令。为了让测试人员能够从源码层面了解覆盖情况, Jacoco 给出了代码源码中的每一行被覆盖的情况, 即每一行代码中至少有一条指令被执行到的情况。
- 方法覆盖率 (Methods): 和行覆盖率类似, Jacoco 给出了代码源码中, 每一个方法中至少有一个指令被执行的覆盖情况, 其中包括构造方法和编译器自动生成的方法。
- 类覆盖率 (Classes): 和行覆盖率类似, Jacoco 给出了代码源码中的每一个类里, 至少有一个方法被覆盖到的情况。

org.wikipedia.staticdata

| Element | Missed Instructions | Cov. | Missed Branches | Cov. |
|----------------------------------|---|------|--|------|
| FileAliasData |  | 99% |  | 50% |
| MainPageNameData |  | 99% |  | 50% |
| SpecialAliasData |  | 99% |  | 50% |
| Total | 24 of 4,575 | 99% | 3 of 6 | 50% |

图 2.4: Jacoco 部分代码覆盖率结果

Jacoco 指令插桩方式有一套专门的逻辑, 其会将桩 (prob) 插入到 java 应用的字节码中。如图2.5所示, 为 Jacoco 为字节码中的 SEQUENCE (源码为直线流程) 和 Jump (源码中出现分支) 逻辑进行插桩的示意图, P 是插入的桩, 具体为一个 boolean 变量。在执行到 P 时, P 的值会进行改变。当测试结束后, Jacoco 会根据 P 的值与位置和源码位置对应, 最终输出相应的红绿黄色覆盖率信息。

对于正常的 Java 代码来说, Jacoco 的覆盖率统计已经做到了尽善尽美, 但是对于安卓源码而言, Jacoco 除了要对安卓字节码进行插桩, 还必须要在拥有源码, 并对源码进行一部分配置的情况下才能够获取到安卓代码覆盖率 (如在 onDestroy 中触发覆盖率文件的生成)。通过对安卓源码进行修改后, 再对源码进行打包, 当运行打包完成的 apk, 触发 onDestroy 函数时, Jacoco 就会自动生

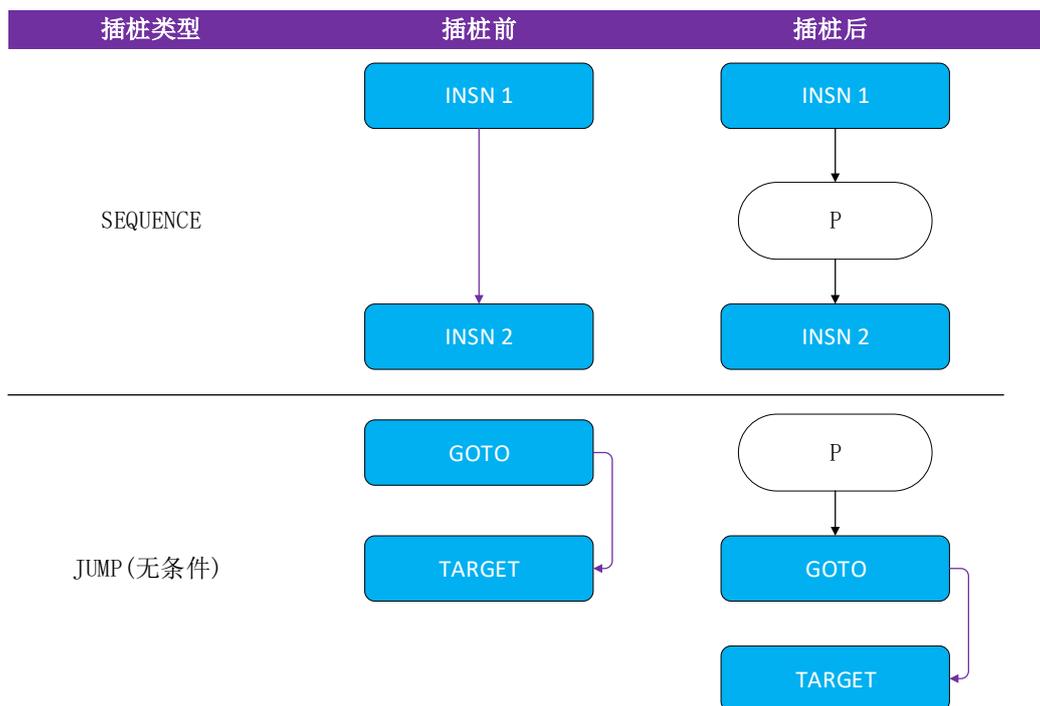


图 2.5: Jacoco 插桩原理图示

成一份 coverage.ec 文件在安卓系统中。当测试结束后，导出这个文件后就可以通过解包的方式将该文件解析成上文所述的覆盖率文件了。

2.6 本章小结

本章节介绍了本论文的系统中所涉及到的相关概念，以及系统所使用到的相关技术。首先，本章介绍了自动化测试的相关概念以及自动化测试在安卓领域的应用；接着，本章介绍了与本系统相关的自动化测试框架的信息，包含了 Appium 框架所使用的 UiAutomator 框架与 ADB 工具的相关原理，以及 Appium 框架本身的工作原理；然后，本文介绍了自动化测试工具的相关知识，在小节中对自动化测试工具的种类进行了分类与介绍，并阐述了本系统使用基于模型的策略的原因；最后，介绍了本系统验证过程中所使用的代码覆盖率获取工具 Jacoco，该工具可以通过插桩的方式对安卓字节码进行修改，从而在运行的过程中获取到应用的被覆盖信息。

第三章 需求分析与概要设计

3.1 系统整体概述

虽然智能手机得到了大量普及，但是，由于 UI 测试流程远长于开发流程的原因，移动应用的质量难以迅速跟上安卓应用的快速发展。使用者们经常会在移动应用上触发一系列功能、界面以及性能上的问题，这也自然使得人们对移动应用的质量产生了质疑。为了修正 BUG，安卓应用的迭代速度大大加快，但是每次迭代的更改幅度却可能非常小。但是无论多小的修改都需要进行一整轮测试，使得测试的耗时远远超出了开发的耗时，需要花费巨大的人力与时间来进行回归测试。随着移动应用开发速度的加快，应用测试的频率也不得不随之得到提升，因此诞生了自动化测试。

虽然研究者们提出了许多自动化测试的方法，但是根据一部分研究，现今的安卓自动化测试工具仍不能满足工业界的需求 [41]。目前常见的工业界移动应用测试方式中，极少数是利用自动化测试工具，一小部分为专家编写自动化测试脚本，而大部分仍然是测试者通过手工测试来验证软件质量。有一些商家会将测试任务分发给众包工人或者志愿者来进行¹，并以文字描述的方式进行问题的反馈，并借由开发者进行修正。而众包工人人们的测试操作，往往在最终的反馈上只会缩减成几句用于复现错误的操作，并不会得到完整的记录与复用 [6]。

本系统通过将人工操作信息与自动化测试工具进行融合的方式，既能够复用测试人员的操作流，又通过这些操作流帮助自动化测试工具探索到更多的应用范围，使得工具的覆盖率得到提升。本系统的整体流程概述如图3.1所示，系统面向用户为测试人员。用户操作分为两个入口：一方面，测试用户可以通过各种能够获取到用户操作信息，并加以保存的方式（如图所示的方式为使用修改后的 Appium 依赖包，在运行自动化测试脚本时自动记录测试流程。本章中均以该方法作为获取用户操作信息的具体实现进行介绍，系统同样包含其他获取信息的方式，后续会进行讲解）将前置数据上传至云端 OSS 中。另一方面，系统会提供一个 web 端交互界面，让测试人员进行前置数据上传、进行自动化测试服务的选择与执行。当用户在 web 端创建测试任务之后，前端的各项功能申请会被统一传输到移动测试平台后端上，再由移动平台后端调用自动化测试的各项功能，执行后续的用户操作流合并，自动化测试逻辑，以及测试结果整理。

¹club.huawei.com/thread-21332310-1-1.html

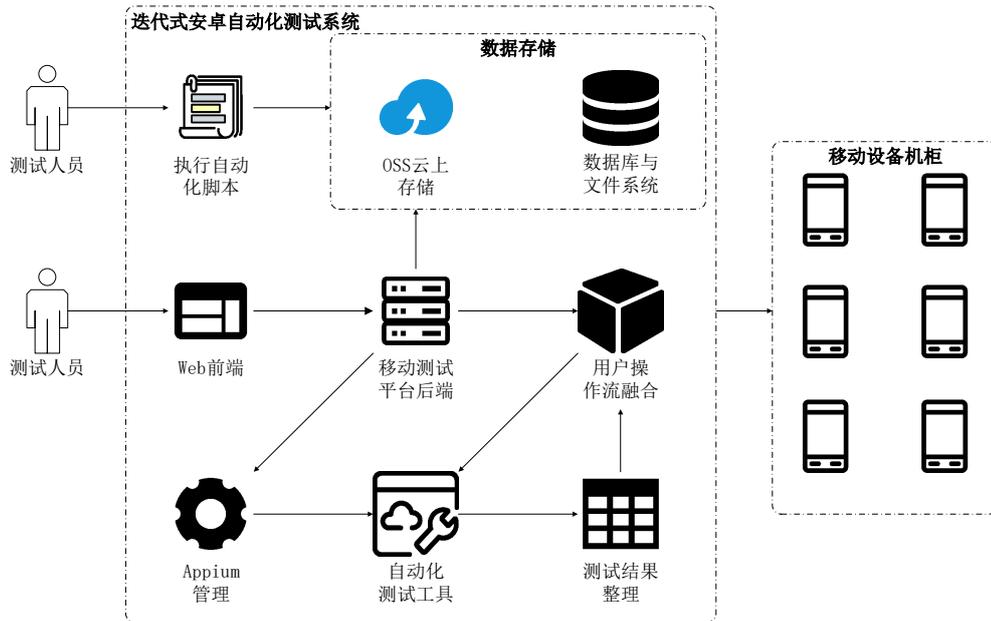


图 3.1: 迭代式安卓自动化测试系统整体流程概述

从功能上分析，本系统的功能模块可以具体划分为移动测试平台前后端、自动化脚本流程获取模块、Appium 管理模块、自动化测试模块以及操作流建模模块。这几个模块负责完成从获取用户操作流、用户操作流建模、Appium 自动建立与删除、进行自动化测试、返回测试结果的整个业务流程。

从使用流程上分析，本系统的主要使用路径分为三条：其一为用户使用接入本系统接口的工具上传与应用相关的用户操作流，如通过使用修改后的 Appium 依赖包进行自动化测试，当测试成功完成后，依赖包中的函数会自动将脚本的测试流程进行用户操作流建模并上传到云端 OSS 上。

其二为用户在 web 页面中进行自动化测试任务的申请与审核、上传测试相关文件，以及查看测试结果。考虑到普通用户可能会上传一些不合理或者错误的文档，系统限定了只允许管理员上传相关任务的用户操作流。这些操作流的来源不限，只要格式正确即可。在测试完成后，除了测试结果外，还会输出本次测试流程在引入用户操作流后，对待测应用覆盖情况的覆盖树以供研究。

其三为用户在 web 页面中选择执行自动化测试任务，系统会对已有的用户操作流进行融合，生成一颗控件覆盖树，用来向工具描述待测应用已经被覆盖的控件框架，并开始执行第一轮自动化测试。执行完毕后，系统会将本次执行过程的操作流和覆盖树进行比对，找出覆盖树中仍未被覆盖的控件结点，在下一

轮测试中优先进行覆盖，从而迭代式的进行下一轮测试，直到覆盖树被遍历完为止。最后，将自动化测试工具在本次测试中自身的测试流程生成一颗覆盖树，并交还给测试人员研究，令测试人员与源码比对，了解到仍未被覆盖到的页面，以便于进行下一轮手工测试。如此往复，可以使得应用覆盖率不断得到提高。

3.2 系统需求分析

3.2.1 功能性需求

本节将对迭代式安卓自动化测试系统的功能性需求通过列举形式进行详细的描述。对测试用户或者管理员而言，他们所进行的操作主要是和 web 前端进行交互或者执行自动化脚本，这会产生一部分需求。而系统由于实现了高度自动化，内部流程也需要由不同的模块之间进行分工合作才能够完成，这就产生了另一部分需求。在本节中，本文会将本系统的功能性需求，根据不同的模块功能进行分类，并一一进行阐明。

web 页面负责与用户进行交互，收集用户上传至系统内的测试任务信息，包含移动应用、用户格式化操作流等信息，也接受用户的自动化测试申请，并向后端发送测试请求。其涉及的具体功能性需求如表3.1所示。

表 3.1: web 交互服务功能性需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|--------------|---|-----|
| R1 | 上传应用 | 测试人员新建待测应用，并将应用的有关信息输入到系统中。 | 低 |
| R2 | 请求/审核自动化测试任务 | 测试人员建立待测应用完毕后，申请对该应用进行自动化测试。管理员可以对该测试请求进行审核，并在后台执行自动化测试。 | 低 |
| R3 | 上传用户操作流文件 | 管理员在审核自动化测试任务的过程中，可以为待测应用上传用户操作流文件，这部分文件可以用于本次以及后续的自动化测试流程。 | 低 |

在执行自动化测试时，系统可能与多台设备相连，这就涉及到 Appium 的管理。Appium 管理模块的作用是对每一款移动设备进行 Appium 的建立，维护以及删除。由于 Appium 是通过接口号进行交互的，因此同一个 Appium 客户端不能够同时接入多款设备。所以，我们在进行多设备执行时，必须要为每一款设备都在一个空闲的接口上启动一个 Appium 客户端连接。这就需要专门设立一个模块来进行 Appium 的建立、管理以及关闭。该模块和测试人员并不会直接联系，而是完全处于系统内部，在执行自动化测试逻辑时自动执行的模块，其具体功能性需求如表3.2所示。

表 3.2: Appium 管理模块需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|-----------|---|-----|
| R4 | Appium 启动 | 系统内部自动为设备对应的 Appium 寻找到空闲接口，在该接口上启动 Appium 客户端，并记录下设备与端口号的对应。 | 高 |
| R5 | Appium 关闭 | 系统内部根据设备与端口号的对应，关闭建立在相应端口号上的 Appium 客户端。 | 高 |
| R6 | 端口号管理 | 系统内部对设备以及端口号的对应进行统一管理。 | 高 |

自动化脚本流程获取模块是获取用户操作信息的一种具体实现，在用户执行自动化脚本的同时，获取到被操作控件的信息，并将这些信息整理成系统可用的用户操作流，最后进行上传。获取具体信息的方式为修改 Appium 依赖包的源码，通过这些修改，在运行脚本时，后台会自动记录下测试过程中，脚本中每一步操作所涉及的控件信息以及页面信息。具体来说，包含被操作的控件本身的信息、操作本身的信息、执行操作前的页面信息以及执行操作后的页面信息。脚本执行完成后，后台会自动将这些信息整合为用户操作流文件，并最终根据待测应用的包名上传至 OSS 中相应的位置。该模块的功能性需求如表3.3所示。

表 3.3: 自动化脚本流程获取模块需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|-----------|--|-----|
| R7 | 脚本流程记录 | 在执行自动化脚本时，后台会自动生成每一步操作的相关信息，并生成相关的记录文件。 | 高 |
| R8 | 用户操作流文件上传 | 当脚本执行完毕后，后台会将收集到的数据整理为相应的用户操作流，并上传至 OSS 中。 | 中 |

自动化测试模块是本系统的核心模块，其具体的算法逻辑会在第四章中进行详细阐述。该模块的作用在于，无论系统内部是否存在可用的用户操作流，本模块都应当能够成功执行对待测应用的自动化测试。同时，测试结束后，要将测试过程中记录的控件信息和控件流程整合为用户操作流文件并进行记录，本模块的功能性需求如表3.4所示。

操作流建模模块同样为本系统的核心模块，主要负责与用户操作流相关的所有操作。本模块可以将自动化测试过程中记录下的信息整合为用户操作流文件，也可以将已有的用户操作流进行融合，生成应用控件覆盖树，帮助自动化工具了解应用的部分结构。本模块在自动化测试结束后给出测试结果的基础上，还需要能够为用户输出可阅读的应用当前控件覆盖树信息。其具体功能性需求如表3.5所示。

表 3.4: 自动化测试模块需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|------------|---|-----|
| R9 | 迭代式自动化测试运行 | 测试用户选择对应用进行自动化测试，系统的自动化测试逻辑根据用户操作流文件获知应用已覆盖的控件模型，并在此基础上展开多轮自动化测试。测试完成后，生成用户操作流文件以及测试结果。 | 高 |
| R10 | 查看测试结果 | 测试用户在申请的自动化测试任务完成后，查看本次测试的结果。测试结果包含每一台设备中，应用的覆盖情况、运行过程中的 BUG 记录、设备本身的耗能情况等。 | 高 |

表 3.5: 操作流建模模块需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|-----------|---|-----|
| R11 | 用户操作流文件生成 | 当自动化脚本执行或自动化测试任务完成后，系统需要对运行过程中生成的中间文件进行整合与格式化，生成与测试过程相对应的用户操作流文件。 | 高 |
| R12 | 用户操作流文件融合 | 当测试用户选择进行自动化测试时，系统会将 OSS 上待测应用相关的全部用户操作流文件进行融合，生成一颗控件覆盖树，辅助自动化算法进行测试。 | 高 |
| R13 | 控件覆盖模型可视化 | 在测试结果中，系统会将本次测试的测试流程进行融合，生成一颗控件覆盖树，并以可阅读图形的形式将这颗控件覆盖树进行输出，让测试用户能够理解目前已经被覆盖到的应用情况。 | 中 |

3.2.2 非功能性需求

迭代式安卓自动化测试系统的非功能性需求如表3.6所示，本系统要求系统界面简约，操作流程易懂，用户可以在经过少量学习后，了解本系统所涉及的用户操作流的生成过程与原理，以及使用本系统完成自动化测试的启动。同时，如果用户输入了格式错误的的数据，系统要能够成功检查出来，不会因为这些数据导致流程出错。并且，系统要能够在长时间运行的情况下，保证自身的恢复时间，如果发生故障，需要在 5 分钟内通过重启等方式的介入回归正常运行状态。

表 3.6: 非功能性需求列表

| 需求编号 | 需求名称 | 需求内容 | 优先级 |
|------|------|---|-----|
| R14 | 易用性 | 系统应当简单易用，用户能够在短时间内掌握其使用方法。 | 中 |
| R15 | 可扩展性 | 系统各功能应当便于拓展，为未来会增加的功能预留好接口。 | 中 |
| R16 | 可靠性 | 系统不会因为错误的输入而导致异常或者崩溃，如果出现崩溃，在恢复后应保证没有数据丢失。 | 高 |
| R17 | 可用性 | 系统应保证自己可以在较长时间内能够稳定运行。 | 高 |
| R18 | 可维护性 | 系统出现故障后可以通过重启系统、输入脚本等方式，在 5 分钟内重新恢复到可运行的正常状态。 | 高 |

3.3 系统用例图

3.3.1 用例图

迭代式安卓自动化测试系统的用例图如图3.2所示。系统角色包含测试人员与管理员，管理员拥有测试人员的所有权限。系统角色涉及到的相关操作主要为前端交互以及运行脚本，更多的系统功能部分由系统后台在自动化测试执行过程中实现。

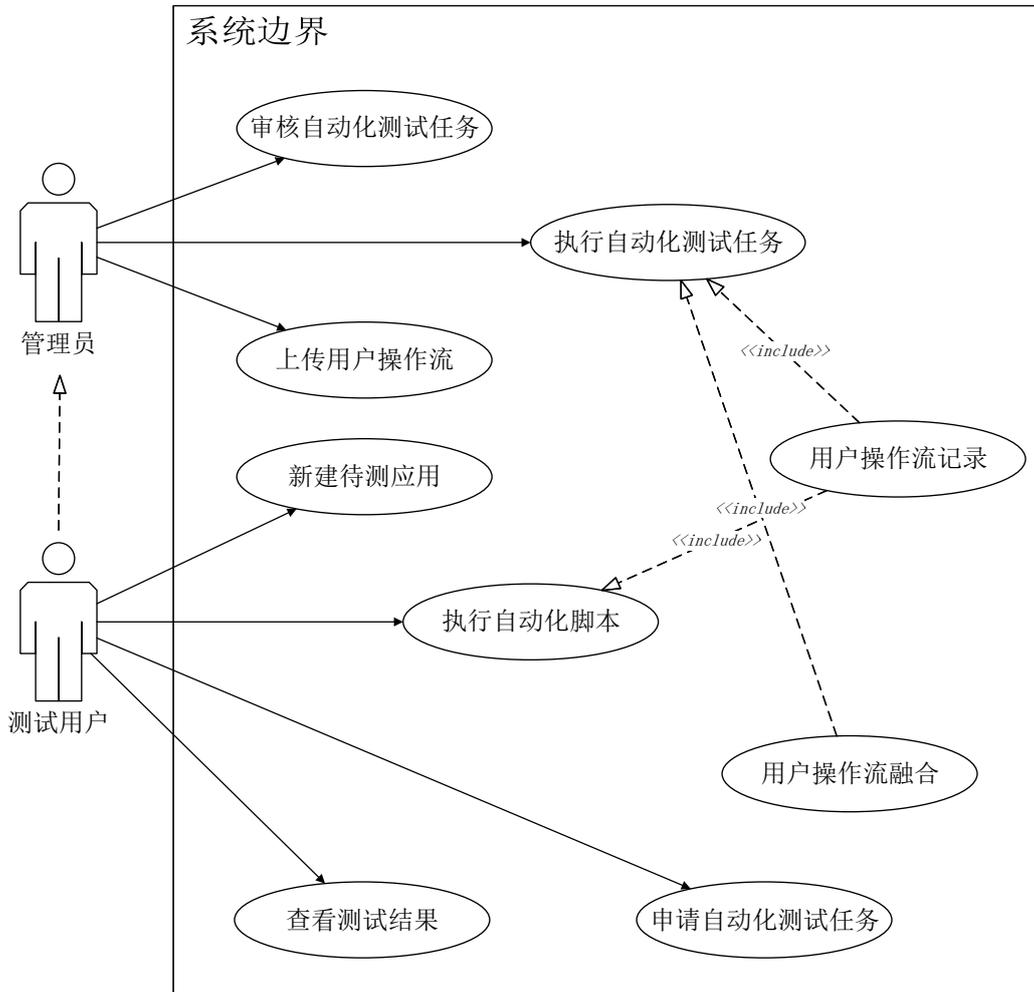


图 3.2: 迭代式安卓自动化测试系统用例图

其中，新建待测应用，申请自动化测试与上传用户操作流，可以对一次自动化测试任务进行初始化，并完善待测应用的相关测试信息。测试人员执行自动

化脚本则会通过后台函数进行记录，并自动上传相关应用的用户操作流。接着管理员可以对测试用户申请的自动化测试任务进行审核，并开始自动化测试执行。在执行自动化脚本或者自动化算法的执行过程中，系统都会自动记录下测试过程中的用户操作流，并进行融合与上传。最后，当测试完成后，测试人员可以查看脚本运行结果以及包含在其中的控件覆盖树信息。

3.3.2 用例总表

如表3.7所示，迭代式安卓自动化测试系统拥有 9 个主要的系统用例，分别为：新建待测应用、申请自动化测试任务、审核自动化测试任务、上传用户操作流、执行自动化脚本、执行自动化测试任务、用户操作流记录、用户操作流融合以及查看测试结果。表中给出了各个用例和需求之间对应的关系，并为每个用例进行了编号，方便对应查看。具体的用例描述在下一节中给出。

表 3.7: 迭代式安卓自动化测试系统用例总表

| 用例编号 | 用例名称 | 需求编号 |
|------|-----------|-------------|
| UC1 | 新建待测应用 | R1 |
| UC2 | 申请自动化测试任务 | R2 |
| UC3 | 审核自动化测试任务 | R2 |
| UC4 | 上传用户操作流 | R3 |
| UC5 | 执行自动化脚本 | R7、R8 |
| UC6 | 执行自动化测试任务 | R4、R5、R6、R9 |
| UC7 | 用户操作流记录 | R11 |
| UC8 | 用户操作流融合 | R12 |
| UC9 | 查看测试结果 | R10、R13 |

3.3.3 用例描述

本节中，会为上一节中描述的系统用例一一进行详细描述：

新建待测应用是使用本系统功能的第一步。测试用户需要先进行应用的上传，将应用上传至云端 OSS 系统上，才能开展之后的进一步测试。该用例的用例描述如表3.8所示。

考虑到部分测试用户上传测试申请会产生问题，系统增加了测试任务审核这一流程，因此，测试任务的开展需要由测试人员编写好测试任务相关信息后进行申请，待管理员审核完毕后，用户才能够继续进行执行操作，令连接至服务器的设备开始进行自动化测试。该用例的用例描述如表3.9所示。

系统管理员可以为测试用户的自动化测试申请进行审核。当管理员认为测试申请中的信息无误时，可以将该任务的状态由未审核修改为审核完毕，当前

表 3.8: 新建待测应用用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC1 |
| 用例名称 | 新建待测应用 |
| 参与者 | 测试人员 |
| 优先级 | 低 |
| 用例描述 | 测试人员在系统中新建一个待测应用。 |
| 前置条件 | 测试人员拥有使用本系统的使用权限。 |
| 主事件流 | 1. 用户点击上传应用按钮。 2. 用户选择待测应用，开始上传。 3. 系统显示上传成功。 |
| 后置条件 | 系统提示应用上传成功，用户可以在应用列表中查看到上传成功的应用，OSS 中对应位置出现该应用。 |
| 扩展事件流 | 3a. 系统检测后台已经存在相同版本的应用，显示上传失败。 |

表 3.9: 申请自动化测试任务用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC2 |
| 用例名称 | 申请自动化测试任务 |
| 参与者 | 测试用户 |
| 优先级 | 低 |
| 用例描述 | 用户向系统提出自动化测试的申请。 |
| 前置条件 | 系统中存在对应的待测应用。 |
| 主事件流 | 1. 测试用户选择待测应用，并输入相关测试信息。 2. 测试用户向系统提出测试申请。 3. 系统显示申请成功。 |
| 后置条件 | 管理员在任务列表中可以看见本次测试申请。 |
| 扩展事件流 | 无 |

任务的状态会被更改为已审核，就可以由测试人员选择开始执行。该用例的用例描述如表3.10所示。

上传用户操作流的目的是为了在自动化测试的过程中，测试工具能够理解到待测应用的控件结构。每一款应用都可以看作页面和控件的组合，而用户操作流可以将应用已经被覆盖的控件和页面展示出来，同时还能附带上到达这些控件与页面的路径信息，这些信息能够给自动化测试工具选择下一步的覆盖方向提供帮助。考虑到用户操作流格式的统一性，这项操作只能由管理员完成，以防工具对待测应用的结构产生错误理解，让工具进行无意义的探索。将用户操作流上传后，它会被自动保存在待测应用包名所对应的存储位置中，等待开启自动化测试时自行调用。该用例的用例描述如表3.11所示。

表 3.10: 审核自动化测试任务用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC3 |
| 用例名称 | 审核自动化测试任务用例 |
| 参与者 | 系统管理员 |
| 优先级 | 低 |
| 用例描述 | 系统管理员审核一份审核中的测试任务申请，并将任务状态修改为审核完毕。 |
| 前置条件 | 系统中存在审核中的测试申请。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 系统管理员打开测试任务列表，并选中审核中的任务。 2. 系统管理员查看任务细节，并通过该任务审核。 3. 系统显示审核成功。 |
| 后置条件 | 任务列表中，该任务的状态由审核中转换为审核完毕，等待执行。 |
| 扩展事件流 | 无 |

表 3.11: 上传用户操作流用例描述

| 描述项 | 说明 |
|-------|--|
| 用例编号 | UC4 |
| 用例名称 | 上传格式化操作流 |
| 参与者 | 系统管理员 |
| 优先级 | 低 |
| 用例描述 | 系统管理员为一次自动化测试任务申请上传用户操作流。 |
| 前置条件 | 系统中存在状态为审核中的任务申请。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 系统管理员选择自动化测试任务申请。 2. 系统管理员选择上传用户操作流。 3. 系统显示上传成功。 |
| 后置条件 | 上传的用户操作流文件被成功上传到 OSS 中待测应用对应的文件夹下。 |
| 扩展事件流 | 无 |

执行自动化脚本是指，用户在执行 Appium 自动化脚本运行时，只需要将 Appium 框架所依赖的两个 Jar 包替换为本系统所提供的修改之后的依赖 Jar 包，在运行的同时，后台就能够自动生成并记录下本次测试流程的相关信息，并在脚本执行完成后，将这些信息整合为用户操作流并进行上传。该用例的用例描述如表3.12所示。

执行自动化测试任务是本系统的核心功能之一，它是指直接对某一款应用在相应的设备上执行自动化测试。系统会先对用户操作流进行融合，生成应用的控件覆盖树，再根据覆盖树帮助自动化测试工具了解应用的部分控件框架。在对应用框架有所了解的情境下，自动化测试也能达到更多的应用页面。测试完毕后，自动化测试工具本身的操作流程也会被记录下来，用于生成新的用户操作流并进行保存。该用例的用例描述如表3.13所示。

表 3.12: 执行自动化脚本用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC5 |
| 用例名称 | 执行自动化脚本 |
| 参与者 | 测试人员 |
| 优先级 | 高 |
| 用例描述 | 测试用户在替换完 Appium 依赖包的情况下执行自动化测试脚本。 |
| 前置条件 | 测试项目中的 Appium 依赖包被替换为系统提供的特制依赖包。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 测试用户运行自己编写的自动化测试脚本。 2. 系统在脚本执行的过程中生成相关记录文件。 3. 系统在脚本执行完成后将相关文件进行整合。 4. 系统根据待测应用信息，将整合后的用户操作流上传到 OSS 中的特定文件夹内。 |
| 后置条件 | OSS 中可以看到上传成功的用户操作流文件。 |
| 扩展事件流 | <ol style="list-style-type: none"> 2a. 自动化测试脚本内容有误，执行失败。 2b. 系统在该待测设备上的测试结果显示自动化脚本执行失败。 |

表 3.13: 执行自动化测试任务用例描述

| 描述项 | 说明 |
|-------|--|
| 用例编号 | UC6 |
| 用例名称 | 执行自动化测试任务 |
| 参与者 | 测试人员 |
| 优先级 | 高 |
| 用例描述 | 测试人员开始自动化测试。 |
| 前置条件 | 系统内存在可用待测应用。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 用户选择选择进行自动化测试。 2. 系统为待测设备开启 Appium 连接。 3. 系统从 OSS 中下载相关的用户操作流，并进行融合，生成控件覆盖树。 4. 系统根据控件覆盖树进行路径复现，复现完毕后转移控制权至自动化测试逻辑。 5. 系统重复第 4 步直到覆盖树被完全遍历。 6. 系统返回测试结果。 |
| 后置条件 | 系统内可以查看待测设备上的相关测试结果。 |
| 扩展事件流 | 3a. 系统内不存在任何该待测应用相关的用户操作流，直接开启自动化测试逻辑。 |

用户操作流记录在执行自动化测试脚本或任务时发生，系统在执行自动化测试脚本或任务时，会将整个测试过程记录下来，最后生成格式一致的用户操作流文件。执行自动化脚本时，只会产生一份用户操作流文件。但是本系统的自动化测试含有多轮，每一轮都会将结合已存在用户操作流信息与自动化算法产生的测试路径记录下来，并生成一份新的用户操作流文件，因此，一轮自动化测试会产生多份用户操作流文件。该用例的用例描述如表3.14所示。

用户操作流融合的目的，在于，将多份用户操作流文件进行融合，生成待测

表 3.14: 用户操作流记录用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC7 |
| 用例名称 | 用户操作流记录 |
| 参与者 | 测试用户 |
| 优先级 | 高 |
| 用例描述 | 测试用户开始自动化测试的过程中，系统会记录下全部的流程，并生成与系统所需输入格式一致的文件，从而进行迭代式测试。 |
| 前置条件 | 自动化测试任务已被审核。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 测试用户选择执行自动化测试任务或执行自动化脚本。 2. 系统在每一轮执行过程中，生成对应的用户操作流文件。 3. 系统将文件上传至对应的 OSS 文件夹下。 |
| 后置条件 | 无。 |
| 扩展事件流 | 无。 |

应用当前被覆盖信息的控件覆盖树，便于测试工具了解应用应用页面与控件的框架组成。在执行自动化测试任务的一开始，该用例由系统执行，同时，在自动化测试工具进行完每一轮的测试后，也需要将自身测试的轨迹进行融合，与控件覆盖树进行比对，从而减少重复测试的时间。当测试任务完成后，系统会将执行过程中生成的用户操作流融合为最终的控件覆盖树，并输出在测试结果中。该用例的用例描述如表3.15所示。

表 3.15: 用户操作流融合用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC8 |
| 用例名称 | 用户操作流融合 |
| 参与者 | 测试人员 |
| 优先级 | 高 |
| 用例描述 | 测试用户开始自动化测试的过程中，系统将用户操作流进行融合，从而生成待测应用的已知控件框架。 |
| 前置条件 | 自动化测试任务已被审核。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 测试用户选择执行自动化测试任务。 2. 系统在测试开始时，将 OSS 中对应应用的用户操作流文件进行融合，生成应用控件覆盖树。 3. 系统在每一轮执行过程中，将工具自身测试流程融合，并与控件覆盖树进行比对。 4. 系统在完成整场自动化测试任务后，将所有的用户操作流进行整合，生成应用控件覆盖树图形信息。 |
| 后置条件 | 测试结果中可以看到应用的控件结构。 |
| 扩展事件流 | 无。 |

查看测试结果是指测试用户在自动化测试任务执行完毕后，查看相关的自

自动化测试结果。这些测试结果包含应用在每一款测试设备上的页面覆盖情况、控件覆盖情况以及设备本身的耗能情况、BUG 触发情况等等。该用例的用例描述如表3.16所示。

表 3.16: 查看测试结果用例描述

| 描述项 | 说明 |
|-------|---|
| 用例编号 | UC9 |
| 用例名称 | 查看测试结果 |
| 参与者 | 测试用户 |
| 优先级 | 高 |
| 用例描述 | 测试用户在一次自动化测试任务执行完毕后，查看测试结果。 |
| 前置条件 | 系统内存在关于该待测应用的测试记录。 |
| 主事件流 | <ol style="list-style-type: none"> 1. 测试用户选择某一次测试。 2. 测试用户选择查看测试结果。 3. 测试用户选择待测设备。 4. 系统显示该待测设备上，本次测试的相关结果。 |
| 后置条件 | 无 |
| 扩展事件流 | 无 |

3.4 系统设计与模块设计

3.4.1 系统总体框架设计

本系统的总体部署情况如图3.3所示。其中，Appium 管理模块、自动化测试模块与操作流建模模块被部署在迭代式自动化应用测试服务器上，这也是本系统的核心部分。自动化测试服务器与移动设备通过物理连接设备相连，从而能够直接获取到相应的移动设备信息。当用户在自身 PC 端的 Web 页面对移动应用平台发出 HTTP 请求时，后端会接受请求，并启动迭代式自动化测试服务器中相应的自动化测试模块，在测试完成后用户操作流建模模块会完成后续操作流程整理流程，并将操作流文件进行数据库存储，完成持久化过程。而自动化脚本流程获取模块需要用户在本本地 PC 端通过修改后的 Appium 依赖包，在执行自动化脚本时，将系统所需的资料上传到云端 OSS 存储空间内。

图3.4则给出了整个迭代式安卓自动化测试系统的系统架构，以及部署在各个服务器上的构件情况。本节会为系统的核心模块分别做一个简要的介绍，接下来的几小节会通过模块使用流程对核心模块的设计进行分析与阐明。

首先，测试用户可以通过 Web 端浏览器，与移动应用测试平台进行交互。测试用户和管理员可以发出上传待测应用、上传用户操作流、申请与审核测试、查看测试结果等请求。移动应用测试平台服务器会在后台捕获这些请求，并转到

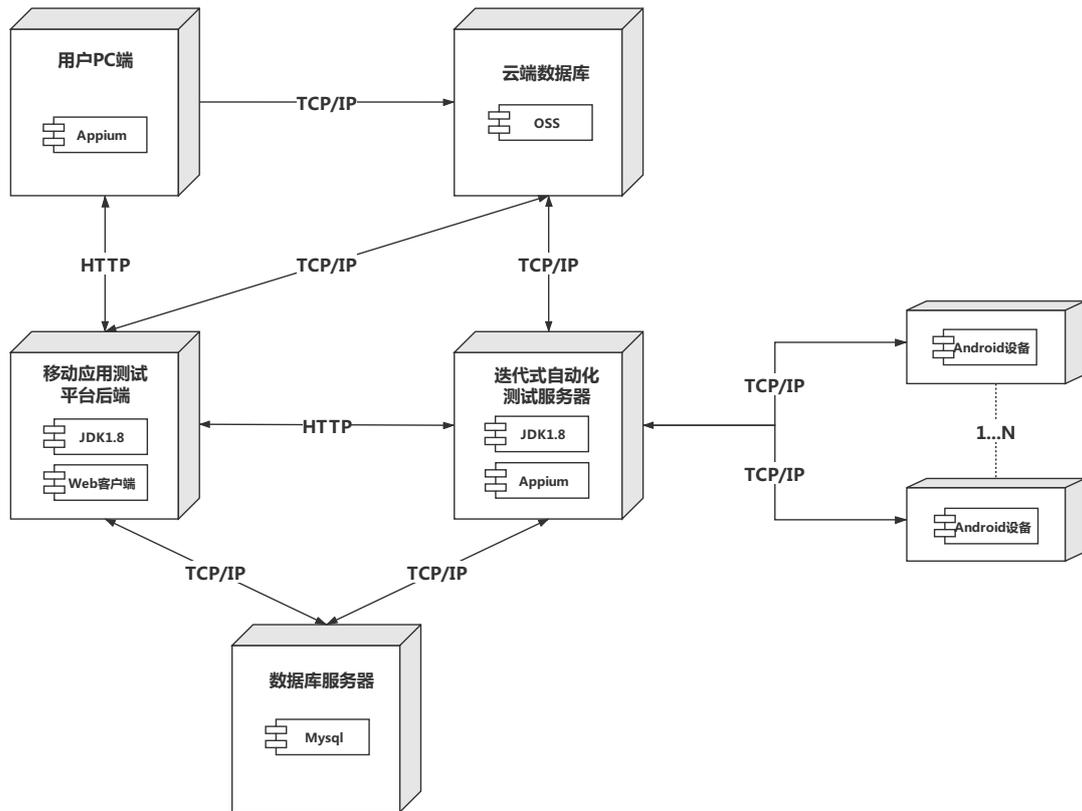


图 3.3: 迭代式安卓自动化测试系统部署图

迭代式自动化测试服务器中进行处理。不同的申请会触发迭代式自动化测试服务中的不同模块。正常使用过程中的申请、查看等流程的实现方式与其他 Web 端系统十分类似，因此此处不再赘述。下面对本系统的核心模块分别进行简要的介绍。

Appium 管理模块负责每一款设备在进行测试时，对应的 Appium 程序的建立、管理和关闭。由于每一份 Appium 应用都对应的系统的一个端口，因此在建立时，该模块会查询预定端口是否空闲，若已被占用，则进行端口号顺延，直至 Appium 被启动。当测试执行完毕后，该模块会自动对对应端口号下的 Appium 程序进行关闭，保证整个测试流程的执行完备性。

自动化脚本流程获取模块在用户运行自动化测试脚本时会被启动。在脚本执行的过程中，后台会为脚本执行操作的每一步进行记录，存储下这一步操作的操作信息、控件信息、设备操作前截图、设备操作后截图、XML 文件记录、AppiumLog 记录、设备 Log 记录等，并将这些文件保存在测试项目的一个特定路径下。当脚本被复现完毕后，这些文件会被整合成本系统所需的用户操作流，

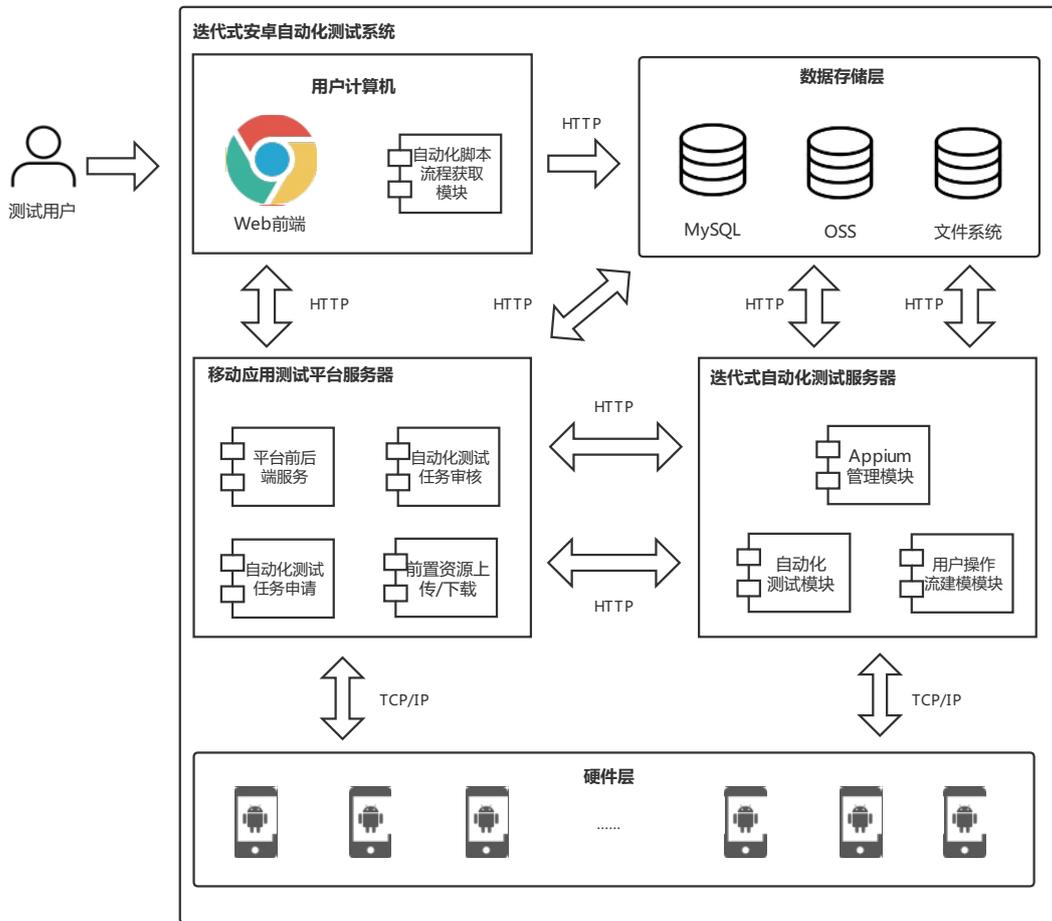


图 3.4: 迭代式安卓自动化测试系统框架图

并上传至 OSS 中。

自动化测试模块会在用户选择执行自动化测试任务时被启动。正常情况下，该模块会对应用进行一次普通的自动化测试，并生成一份用户操作流。接着，系统会将待测应用原有的操作流从 OSS 上进行下载，融合后生成一颗控件覆盖树，并将本次测试的操作流与该覆盖树进行比对。根据覆盖树未遍历到的部分，继续进行多轮测试覆盖。直至测试流程包含了并遍历了覆盖树中所有的结点，系统才会认为测试结束。测试完成后，用户操作流程建模模块会将本次测试的操作流作为应用建模的最新操作流进行更新。

操作流程建模模块除了处理测试完成后的操作流外，在自动化测试任务执行完毕后，模块还需要将系统内保存的操作流进行合并融合，生成一份最新的应用控件覆盖树，并通过 echarts 绘制出覆盖控件的力引导图，以及使用 dot 绘制

出一张测试人员可以理解的 PNG 格式应用覆盖树文件，以便于测试人员理解目前应用的覆盖情况，并开展后续的测试研究。

3.4.2 Appium 管理模块设计

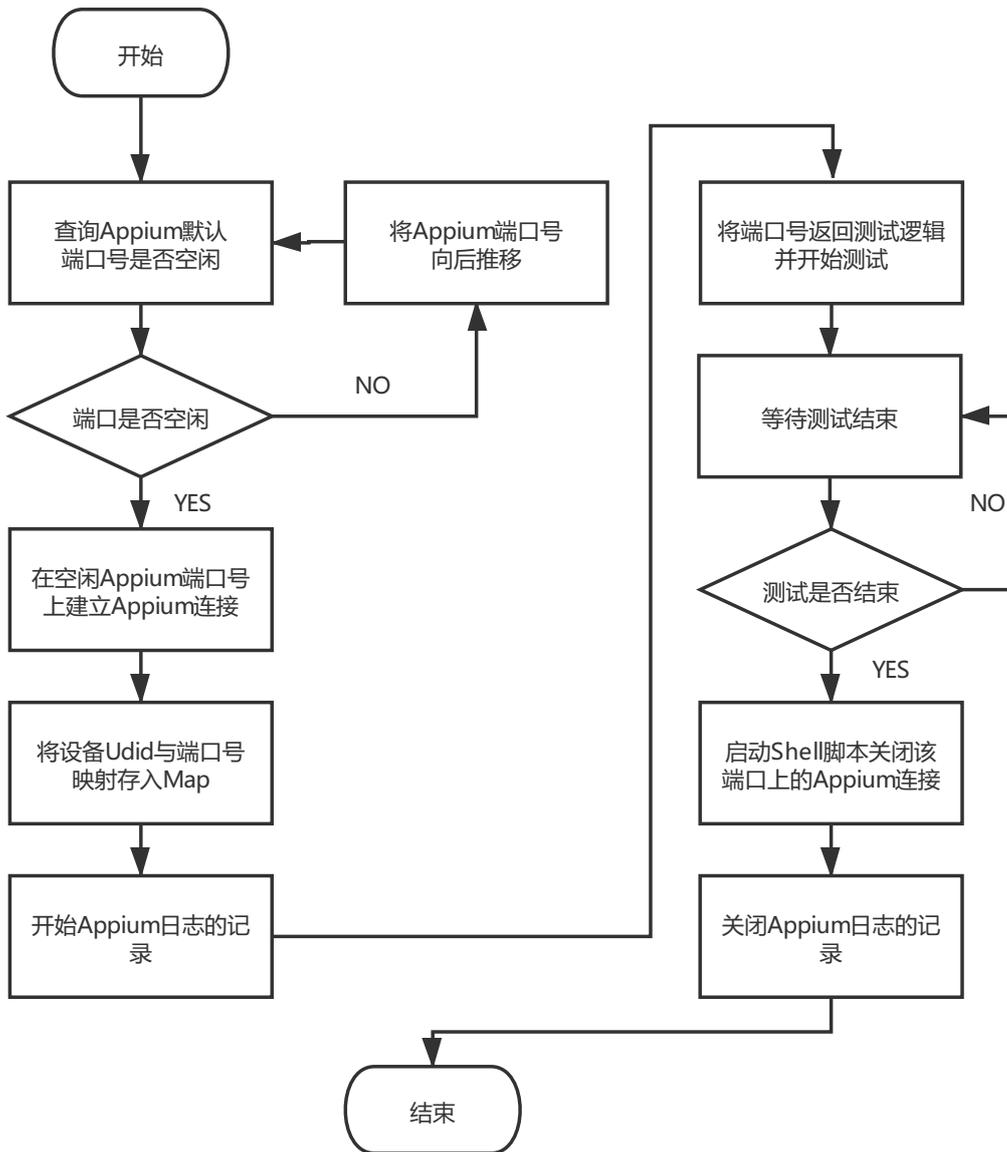


图 3.5: Appium 管理模块流程图

由于 Appium 框架本身功能的限制，一个 Appium 客户端只能对应一台移动设备。因此，我们需要一个 Appium 管理模块对一次测试中所有需要被开启的

Appium 进行统一管理。本模块的操作流程如图3.5所示。

在多设备同时运行的情况下，Appium 管理模块需要为每一台设备寻找到可用且的空闲的端口，并将端口号进行保存，当测试执行完毕后，释放该端口号，将相应的 Appium 程序进行自动化关闭。开启一个 Appium 应用需要两个端口号，默认端口号设定为 4723，同时，将 4823 设定为手机中 BootStrap 的对应端口。本模块会先检查 4723 是否空闲，若空闲，则将这一次设备对应的 Appium 程序开启在 4723 端口号上。若忙碌，则认为已经有其他设备在该端口号上开启了 Appium 程序，则将数字加一，到 4724 上查询该端口号是否忙碌，直到找到一个空闲的端口号，成功打开 Appium 程序为止才会结束。测试模块会根据存储的开启 Appium 的端口号，对已经打开的 Appium 程序一一进行关闭操作，保证这一次测试不会影响到下一次测试。

3.4.3 自动化脚本流程获取模块设计

自动化脚本流程获取模块的主要目的，是为了收集脚本运行过程中的控件覆盖信息，将被封存的脚本进行再利用，并从中提取出人类的智慧，辅助自动化工具继续测试。这一模块的主要流程如图3.6所示，包括记录测试过程、生成测试结果等。

对于安卓应用测试而言，最重要的并不是哪些控件被覆盖到了，而是控件被覆盖的顺序。因为一些控件可能需要进行一系列固定的操作才能够到达，所以本模块的重要意义，除了记录下每一个被覆盖的控件本身的情况外，还记录下了被覆盖控件之间的相互关系，这一关系会被用户操作流建模模块进行整合，生成系统所需的用户操作流文件。

本模块的具体实现方式，以修改 Appium 依赖包源码的方法来进行。Appium 自动化测试脚本在执行时每一种动作时，都会调用两个类 RemoteWebDriver 以及 RemoteWebElement，这两个类继承自 Selenium 包中的 WebDriver 以及 WebElement 类。每一个具体的操作，如点击、滑动、触摸、输入系统按键等，都在 RemoteWebDriver 和 RemoteWebElement 类中有对应的方法。通过对这些方法进行代码注入，我们可以获取到每一步操作中，被操作的控件的所有信息，以及这次操作的具体内容。同时，Appium 本身自带查询 Activity 状况的函数接口，根据在进行操作前后对该函数进行调用，我们可以获取到操作进行前后的页面状态。代码层面的实现方式在第四节中会进行详细的叙述。

通过对控件信息的获取，本系统将每一次操作都封装成了一个包含控件信息、操作信息、执行前后页面状态的三元组 $W = (WidgetInfo, behavior, Activities)$ 。一次脚本执行的完成，意味着我们获得了一个包含着所有操作信息的列表 $List(W)$ ，

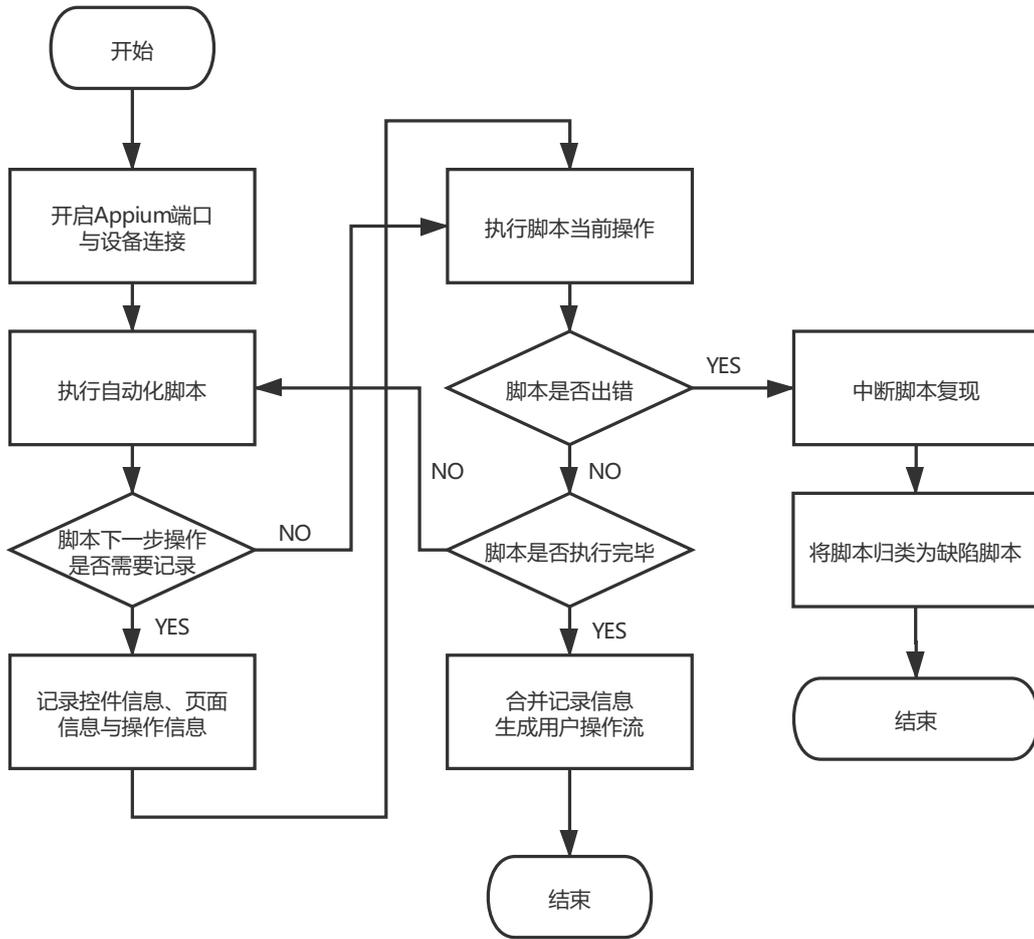


图 3.6: 自动化脚本流程获取模块流程图

该列表即为用户操作流建模模块所需的用户操作流，同时，由于一份脚本的执行顺序是固定的，故而列表索引的顺序能够自动转化为用户操作流的顺序。

3.4.4 自动化测试模块设计

自动化测试模块为本系统的核心模块。该模块所包含的自动化测试算法的执行方式与一般的基于模型策略的自动化测试工具较为类似，都是以页面爬虫的形式对应用进行覆盖。但也有不同点，就是引入了前文提到的包含控件部分控件框架信息而形成的用户操作流。

本模块的流程图如图3.7所示，自动化测试逻辑会根据引入用户操作流生成控件覆盖树，帮助工具了解应用的控件框架，并对应用进行多轮测试。第一轮测

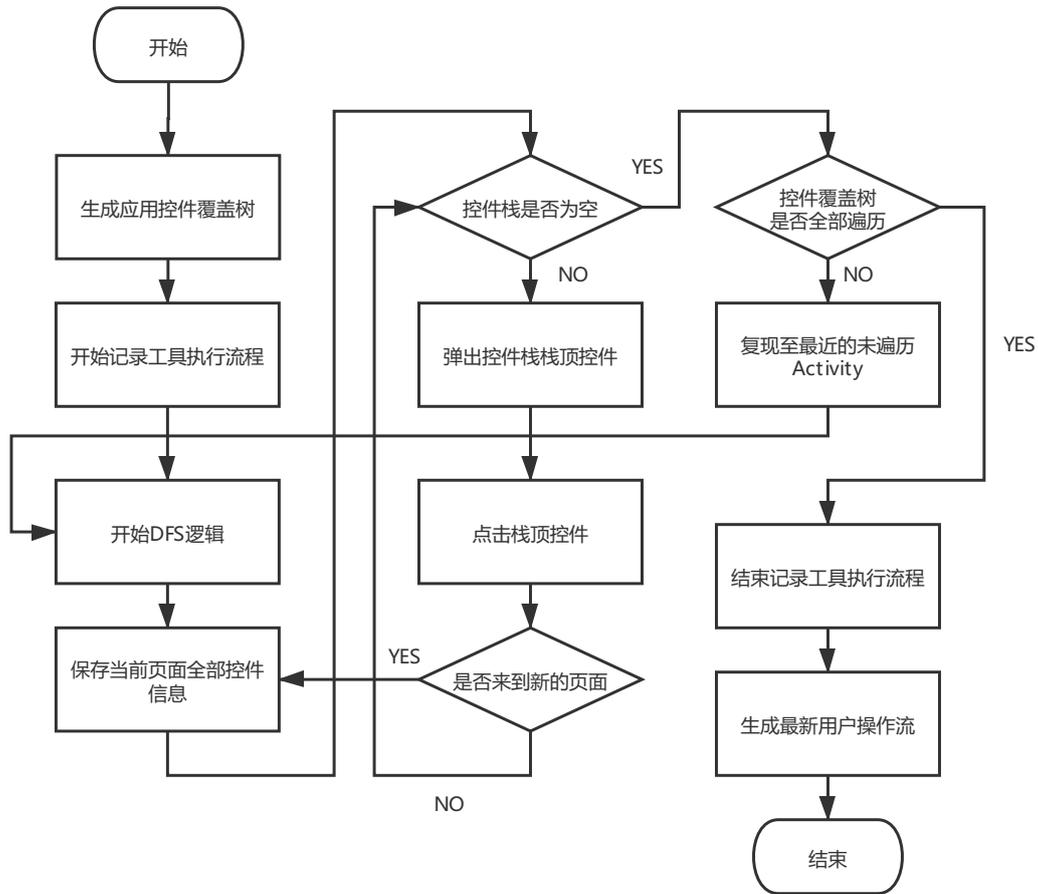


图 3.7: 自动化测试模块流程图

试为单纯的 DFS 遍历测试，模块会在到达一个新的应用页面时，记录下该页面的所有控件信息，并将控件存入到一个栈中，一一取出，并进行点击操作。在遍历的过程中，如果页面发生了改变（本模块维护了一个页面列表。其中，每一个页面包含一个列表，存放着放置在其上的所有控件，一旦页面和页面之间的控件差异率超过了 50%，则认为页面发生了改变），则将当前页面的所有信息进行记录，在新的页面开启新一轮的 DFS 测试。直到新的页面被测试完毕，回到上一个页面时，模块会将保存的信息重新恢复，并继续原本一轮的测试。和执行自动化脚本并获取用户操作流这一操作相同的是，本模块进行的测试，会记录下每一次操作所对应的控件的全部信息，操作本身的信息，以及操作前后的页面状态，用于后续流程的分析。

当第一轮测试执行完毕后，这一轮测试的流程会被记录，并生成属于工具

的控件覆盖树。同时，原本应用自带的用户操作流也会被用户操作流建模模块融合，恢复为数据，生成一颗被引入的应用控件覆盖树。通过对这两份数据进行对比，本模块可以获知在第一轮测试完成后，在引入的控件覆盖树中还有哪些控件没有被遍历，并依据覆盖树的边数据复现未覆盖的路径。

在第二轮测试中，模块会关注控件覆盖树中，仍没有被工具覆盖到达的 Activity 有哪些，并根据覆盖模型复现出到达这些 Activity 的路径。将应用恢复到这些未被覆盖的 Activity 上后，重新将操作权交还给自动化测试程序，并继续进行自动化测试，直至这一轮测试被执行完毕。接着再次进行检查，迭代式的进行新一轮新的测试，直到应用控件覆盖树模型被全部覆盖为止。此时，自动化操作模块会生成一份融合了原本应用用户操作流以及本次自动化测试流程的用户操作流文件，这份文件会作为最新的用户操作流文件得到保存，而原先的其他操作流会被封存，以减少下一次在进行操作流融合时所消耗的系统能耗。

当测试完成后，一份全新的操作流文件会被生成，这份文件作为下一次自动化测试人物的引导文件，也会用于在测试报告中生成控件覆盖模型。同时，系统会将每一款测试设备上的测试情况进行收集，返回给移动应用测试平台，让测试人员能够了解在每一台设备上的测试情况。

3.4.5 用户操作流建模模块设计

用户操作流建模模块的模块流程图如图3.8所示。本模块的核心作用，为将用户的操作流融合，生成能够让系统读懂的应用覆盖数据，以及让人能够读懂的应用操作树。

由于一部分安卓应用并不开源，同时，即使开源应用也很难通过代码静扫来获取全部的应用控件信息，所以本系统将安卓应用的整体控件覆盖情况通过覆盖树的方式进行展示。当测试执行完毕后，本系统会自动生成一份三元组列表 $List(W)$ ，本文将之称为格式化用户操作流。通常情况下，多条操作流之间，会有相互重叠的控件，因此，本模块会将这些操作流进行融合，最终生成一颗无重复控件操作信息的应用控件覆盖树，这棵树代表了目前应用已经被覆盖到的控件以及 Activity 情况。

从数据层面上来看，这棵操作树可以被自动化测试工具所读取，从而帮助工具了解到已经被覆盖的控件信息，以及测试过程中还未覆盖的控件信息，从而帮助自动化测试工具进行路径的选择。而从测试人员的视角来看，这棵操作树也可以被本模块转换为人类可理解的图形，以便于测试人员更加了解目前的覆盖情况，从而对将来的测试方向作出判断。

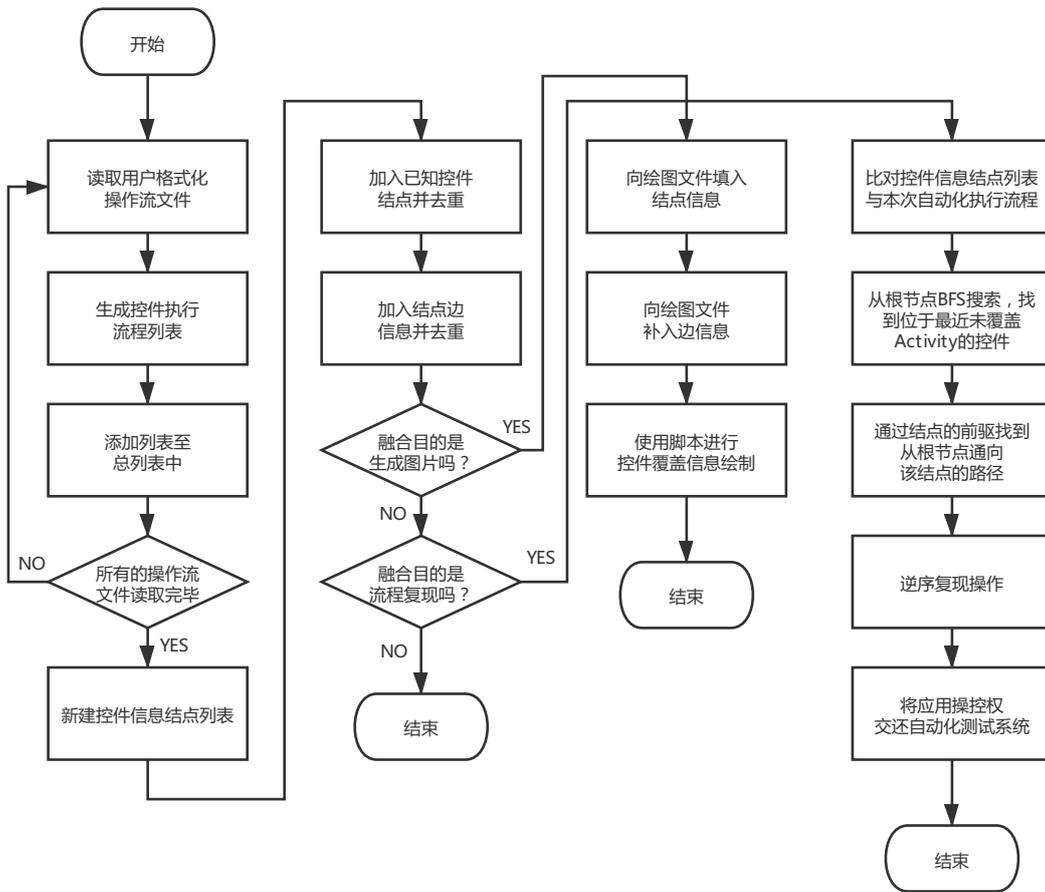


图 3.8: 用户操作流建模模块流程图

3.5 用户操作流设计

3.5.1 用户操作流数据化

本系统的核心问题在于如何将自动化脚本、自动化工具甚至人类操作本身(实质上均为操作流,本文中统称为用户操作流)凝练成一种格式化的数据,让工具能够理解其内容。在受到多款自动化测试工具的启发后,本文作者认为,用户操作流需要在用户操作进行的同时,就通过开启辅助工具等方式进行记录。

由于本系统为实验室移动应用测试平台的子系统,而移动应用测试平台的测试任务必须于多设备环境下完成,即一份脚本需要在移动应用设备集群,包含数款到数十款不同型号的设备中进行运行,因此,用户的操作不能从像素层级进行记录,以防设备分辨率变化后用户操作复现失败(即使计算了设备间的分辨率比值,仍然有一定风险导致操作复现失误)。所以,本系统选用了 Appium,

一款能够对应用控件进行精准定位的自动化测试框架，作为记录用户操作的基准。该框架可以通过控件本身的属性对控件进行定位，即使在不同的移动设备上，控件的属性是不变的，因此这可以使得定位操作能够更加具有稳定性。

本文作者认为，我们可以将用户操作流以控件和页面信息的组合形式进行保存。举例来说，有这样一个测试流程：测试用户点击了“用户名栏”控件，输入了用户名，点击了“密码栏”控件，输入了密码，点击了“登录”控件。这样一个操作流程就可以被简化为一个三元组 $W = (WidgetInfo, behavior, Activities)$ 的列表：

$$\begin{aligned} List(W) = \{ & W_1(\text{ID 控件信息 (包含控件 id, text, desc 等)}, \text{Click}, \text{LoginActivity}), \\ & W_2(\text{ID 控件信息}, \text{input(用户真实 ID)}, \text{LoginActivity}), \\ & W_3(\text{Password 控件信息}, \text{Click}, \text{LoginActivity}), \\ & W_4(\text{Password 控件信息}, \text{input(用户真实密码)}, \text{LoginActivity}), \\ & W_5(\text{登陆控件信息}, \text{Click}, \text{LoginActivity}) \} \end{aligned}$$

由于用户的操作一定会接触到应用的控件，所以每一份用户操作流程必然可以数据化为上述所述的 $List(W)$ ，而该列表之中的索引顺序，自然代表了本次操作的流程顺序。

3.5.2 用户操作流融合生成应用控件覆盖树

通常来说，一份用户反馈很难将整个应用覆盖完全。对应到本系统中，即为一条用户操作流很难将整个应用的控件覆盖完全。但是，我们可以通过将多条操作流融合，来对应用控件架构进行更好的建模。本系统可以与安卓 UI 自动化测试考试相结合，从几十份学生脚本中一次性提取出数十条用户操作流、或者在测试人员使用本系统时，将其所有操作进行记录、也可以与基于图像识别的跨平台测试脚本录制与回放子系统 LIRAT [43] 相结合，直接从用户点击操作中获取到多条用户操作流信息。

首先，这些操作流通常会有许多重合的控件，比如操作流 A 直接进行了登录，操作流 B 先走了一遍注册流程，再进行了登录，这其中，对“登录”控件的操作，就与上一小节中的所述登录流程 $List(W)$ 就产生了重复，融合流程中需要做的，就是对这些重复的控件进行删减，并将不重复的控件加入到最终结果的覆盖树中。

其次，虽然一条操作流的流程是单向的，没有任何分支，但是当多条操作流融合时，就会产生控件分支，因此，我们需要了解到控件之间的移动顺序，所以，我们将控件覆盖树描述为：

$$\text{List}(WNode) = \{WNode_1(W_1, \text{List}(\text{father}), \text{List}(\text{next})),$$

..... ,

$$WNode_N(W_N, \text{List}(\text{father}), \text{List}(\text{next}))\}$$

其中，WNode 包含了上文所述的 W，即 widget 控件的所有信息，以及用户对该控件进行的操作信息与该控件所属的页面信息。最终得到的结果是一个结点信息的列表，列表中的结点包含了控件信息，以及控件的父子控件关系。当我们获取到其中任意一个结点时，都可以通过直接由 father 进行逆推的方式（加入 visit 数组防止无限循环）找到从 root 结点到达该结点的最短路径。同时，以正向 BFS 的方式进行搜索，我们也能找到距离目标 Activity 最短的一条路径。而由于 WNode 包含了 W 三元组的所有信息，到达每一个结点的方式都能直接通过 Appium 精确定位，并进行复现。

对于生成覆盖树图形的方式，我们则选择使用 echarts 与 DOT + graphviz 的方式进行输出，前者便于生成节点相关的力引导图，着重与描述页面覆盖情况，后者则是一种快速便捷的绘图方式，着重于描述控件覆盖情况。这两种方式都支持以先声明结点，再声明边的方式进行测试结果的绘制，这和本系统的控件信息结点的创建方式不谋而合，因此可以快捷的输出最终结果，具体的截图可见第五章中的系统演示。

3.6 数据库设计

本系统的数据库设计如图3.9所示，包含的实体有：测试用户、移动设备、格式化用户操作流、待测应用、测试任务和测试结果。

图中，测试用户与待测应用、格式化用户操作流都为一对多的关系，代表每个测试用户可以上传多个待测应用与用户操作流。测试任务代表了开展一次测试后，相关的测试任务信息，由于可以对一款应用测试多次，所以待测应用和测试任务间为一对多的关系。测试结果则是指每一个测试任务执行完毕后，在每一台移动设备上产生的结果信息。由于开展一次测试，可能会在多台设备上执行测试流程，因此也就可能产生多条测试结果，测试结果和移动设备之间通过设备 ID 与应用 ID 进行对应，也因此，测试结果与测试任务和移动设备都为多对多的关系。格式化操作流表则是执行测试任务时系统会查询的表，系统会在执行任务时根据应用 ID 查找可用的格式化操作流，并进行下载。

其中，测试用户表、移动设备表的内容都主要为一些固定的用户或硬件信息，数据结构较为简单，因此不进行过多描述。而格式化用户操作流表的内容和待测应用表较为类似，因此只详细阐述后者，以作代表。

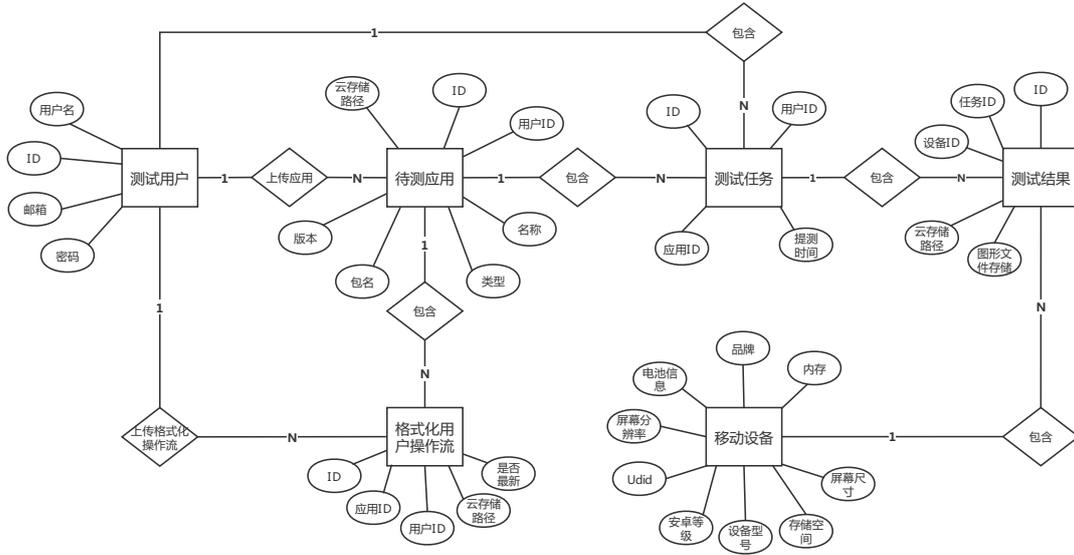


图 3.9: 系统数据库关键实体关系图

待测应用表的数据结构如表3.17所示，主要包含了应用的各类信息，以及存储路径。本系统的存储方式均为直接连接至 OSS 云的下载路径，因此数据库内只需要存储下载地址即可。

表 3.17: 待测应用表数据结构

| 字段 | 名称 | 数据类型 | 可否为空 | 详细描述 |
|-------------|-------|--------------|--------------|-----------------------|
| id | ID | int | 否 | PK，待测应用的唯一标识 |
| userid | 用户 ID | int | 否 | FK，用于检索上传待测应用的用户 |
| name | 应用名称 | varchar(255) | 否 | 应用名称，应用的称呼 |
| categories | 类型 | varchar(255) | 否 | 应用类型，应用的分类 |
| packagename | 包名 | 否 | varchar(255) | 应用包名，用于进行应用标识 |
| version | 版本 | varchar(255) | 否 | 应用版本，同一个包名的应用可以拥有不同版本 |
| ossurl | 云存储路径 | varchar(255) | 否 | 应用在 OSS 中的下载地址 |

测试任务表的结构则如表3.18所示，该表内容包含了一条测试任务的全部信息，当用户申请测试任务时，就会为该表插入一条数据。一款待测应用会对应多条测试任务记录，需要进行记录查询时，可以根据相应的用户 ID、应用 ID 与其他表进行一一对应。

表 3.18: 测试任务表数据结构

| 字段 | 名称 | 数据类型 | 可否为空 | 详细描述 |
|------------|-------|----------|------|-----------------------------------|
| id | ID | int | 否 | PK, 测试任务的唯一标识 |
| userid | 用户 ID | int | 否 | FK, 用于检索测试任务的发起用户 |
| appid | 应用 ID | int | 否 | FK, 用于检索测试任务的应用信息 |
| createtime | 提测时间 | datetime | 否 | 本次测试提交申请的时间, 用于进行格式化用户操作流文件有效性的判断 |

测试结果表的结构则如表3.19所示, 该表内容为测试结果的最基本单元, 当测试执行完毕时, 会根据测试任务执行时连接的设备, 为每一台设备中的执行结果进行记录。后续需要进行记录查询时, 可以根据相应的应用 ID 以及任务 ID 与其他表进行一一对应。

表 3.19: 测试结果表数据结构

| 字段 | 名称 | 数据类型 | 可否为空 | 详细描述 |
|-----------|--------|--------------|------|---|
| id | ID | int | 否 | PK, 测试结果的唯一标识 |
| udid | 设备 ID | varchar(255) | 否 | FK, 移动设备的唯一标识, 为移动设备本身自带的 udid 而非表的 id |
| taskid | 任务 ID | int | 否 | FK, 用于检索该测试结果来源于哪一位用户发起的测试任务 |
| ossurl | 云存储路径 | varchar(255) | 否 | 测试结果包含多个文件与文件夹, 会在生成后直接被打包并上传到 OSS 中, 此处为下载该 Zip 包的地址 |
| picossurl | 图形文件存储 | varchar(255) | 否 | 为了方便前端获取覆盖树图形数据, 将覆盖树图形分离出来, 使前端能够直接获取 |

3.7 本章小结

本章描述了迭代式安卓自动化测试系统的整体概述, 首先以功能结构的方式对本系统进行了分析。接着, 本章对本系统的功能性需求以及非功能性需求进行了描述, 并给出了这些需求所对应的用例分析。然后, 本章描述了本系统的整体框架设计, 对本系统的相关模块: Appium 管理模块、自动化脚本流程获取模块、自动化测试模块以及用户操作流建模模块进行了一一简述, 并通过流程图的方式详细介绍了这些模块的执行顺序与执行内容。接着, 本节给出了本系统的核心数据: 用户操作流的组成介绍, 以及相关使用, 以简介明了的示例图展示了格式化用户操作流的构成。最后, 本章描述了本系统的数据库设计, 给出了系统数据库的实体关系图以及数据库表结构信息。

第四章 详细设计与实现

本章会为读者介绍本系统的数个核心模块的详细设计与具体实现，其中包括 Appium 管理模块，用于管理一轮测试中所有需要被开启的 Appium 客户端；自动化脚本流程获取模块，是用于获取用户操作流数据方法的具体实例之一，该模块将用户运行自动化脚本时产生的脚本路径记录并生成用户操作流，最终进行上传；自动化测试模块，用于执行用户提出的自动化测试任务，读取用户操作流并进行多轮测试；以及用户操作流建模模块，用于管理所有与用户操作流相关的方法以及各项相关文件的生成与读取。

除开上述核心模块以外，本系统的具体实现部分还包含用于连接实验室移动应用测试平台的相关前后端代码，考虑到篇幅限制，以及其内容较为简单与公式化，因此不在本章进行详细说明。以下开始，为各核心模块的详细介绍：

4.1 Appium 管理模块详细设计与实现

4.1.1 Appium 管理模块概述

由于移动应用测试平台拥有多设备环境下进行测试的需求，因此本系统需要在单移动设备环境与多移动设备环境下都能够正常运行。多设备环境运行时，需要针对每一款连接迭代式自动化测试服务器且被选中的设备进行区分。这需要依靠每台设备自带的设备 Udid 进行区分。同时，由于 Appium 在每一个接口上，只能和一台设备进行连接，因此，我们需要一个管理模块来专门管理每一台安卓设备所对应的 Appium 的建立、维护与关闭。本节将阐述 Appium 管理模块在新建、管理与关闭 Appium 服务端上的详细设计与实现。此外，由于该模块本身的内容不多，系统内部也还有一些各种模块都需要使用到的工具类，因此将这些工具类也归类到该模块中，以保证该模块的量级。

4.1.2 Appium 管理模块详细设计实现

通过 Appium 管理模块进行一次自动化测试任务的时序图如图4.1所示，当迭代式自动化测试服务器接收到来自移动应用测试平台的测试要求后，会向 AppiumManager 类发送一个 Appium 的启动申请。此时，AppiumManager 会通过一张设备 Udid 与 Appium 接口 port 进行映射的 Map 来检测需要启动 Appium 的设备是否已经启动了相应 Appium。如果没有，则请求 PortManager 类获取一个空

闲的 port 号。PortManager 类会通过 Socket 类从默认 port 号开始扫描，如果端口已经被占用，则将 port 号数量加一，并继续检测，直到找到空闲的端口号为止。

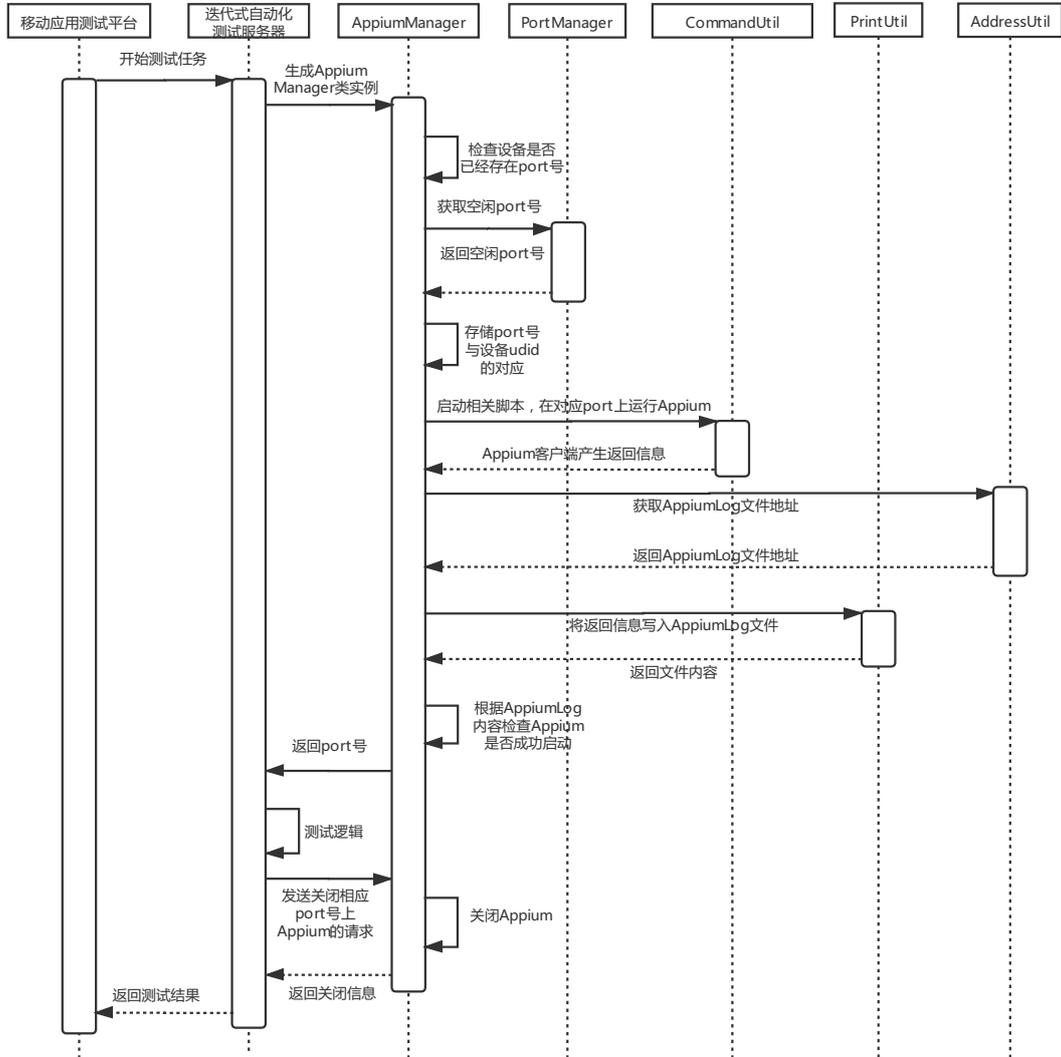


图 4.1: Appium 管理模块时序图

将该 port 号返回给 AppiumManager 后，AppiumManager 会通过 CommandManager 类（管理所有脚本运行的工具类，其余核心模块也会使用到）运行预先设定好的 shell 或 bat 文件来启动 Appium 客户端。等待 10 秒后，AppiumManager 会通过 AddressUtil 类（管理所有输出文件地址的类，其余核心模块也会使用到）获取到 AppiumLog 的输出地址，并通过 PrintUtil 类（管理所有输出内容的工具类，其余核心模块也会使用到）将 Appium 客户端产生的运行信息输入到

AppiumLog 文件中，最后对其内容进行检查。如果 Appium 启动成功，Log 内会产生相应的标识，此时可以将 port 号返回给迭代式自动化测试服务器进行后续的测试逻辑。

当测试完成后，迭代式自动化测试服务器会给 AppiumManager 类发送关闭相应 Appium 的请求，AppiumManager 类则可以直接通过 port 端口号对相应 Appium 进行关闭并返回关闭结果，此时，迭代式自动化测试服务器中产生的结果数据会存储至数据库及 OSS 中，同时会将测试结果返回给移动应用测试平台以便令测试用户知晓测试结果。

Appium 管理模块涉及到的主要类说明如表4.1所示，AppiumManager 类用于进行 Appium 的新建、管理与关闭，PortManager 类用于进行对应端口的检查以及获取空闲端口，CommandUtil、AddressUtil、PrintUtil 为三个工具类，分别管理整个系统中与系统 cmd 命令相关的操作、系统所有输出文件的地址以及系统所有的内容输出。这几个工具类在其他模块中也会被使用到，由于本模块本身功能较少，大多数功能为后台 Appium 客户端的管理，因此将这几个类归并到本模块中。在本模块中，CommandUtil 用于为系统命令行发送新建和关闭 Appium 的请求，AddressUtil 用于获取 AppiumLog 的输出地址，PrintUtil 类用于输出 AppiumLog。这三个模块在后续模块中也几乎会被使用到，为了减少类描述的重复性，仅在本模块中进行描述。

表 4.1: Appium 管理模块主要类

| 类名称 | 分类 | 作用 |
|---------------|-----|--|
| AppiumManager | 服务类 | 启动、管理与关闭 Appium |
| PortManager | 服务类 | 查询端口是否空闲、返回空闲端口号 |
| CommandUtil | 工具类 | 用于执行预设好的命令行脚本。本模块中的作用为提供 Appium 的启动与关闭脚本 |
| AddressUtil | 工具类 | 用于直接返回各文件的存储地址。本模块中的作用为提供 AppiumLog 的存储地址 |
| PrintUtil | 工具类 | 用于进行各种级别的输出。本模块中的作用为输出格式化的 Appium 日志，即 AppiumLog |

这些类的关系如图4.2所示。由于 AppiumManager 需要存储所有的设备与 Appium 端口对应，所以只能存在唯一实例。当其他模块需要开启测试任务时，会通过 getInstance 方法获取 AppiumManager 的唯一实例，接着，AppiumManager 会调用 PortManager 中的方法判断与获取到目前空闲的端口，并通过 CommandUtil 中的脚本，在该端口上开启 Appium 客户端。当开启完成后，AppiumManager 会利用 PrintUtil 类的方法输出该端口号上正在运行的 Appium 的日志文件，并通过 AddressUtil 类的方法确定到系统中 AppiumLog 文件的具体为止，通过对内容

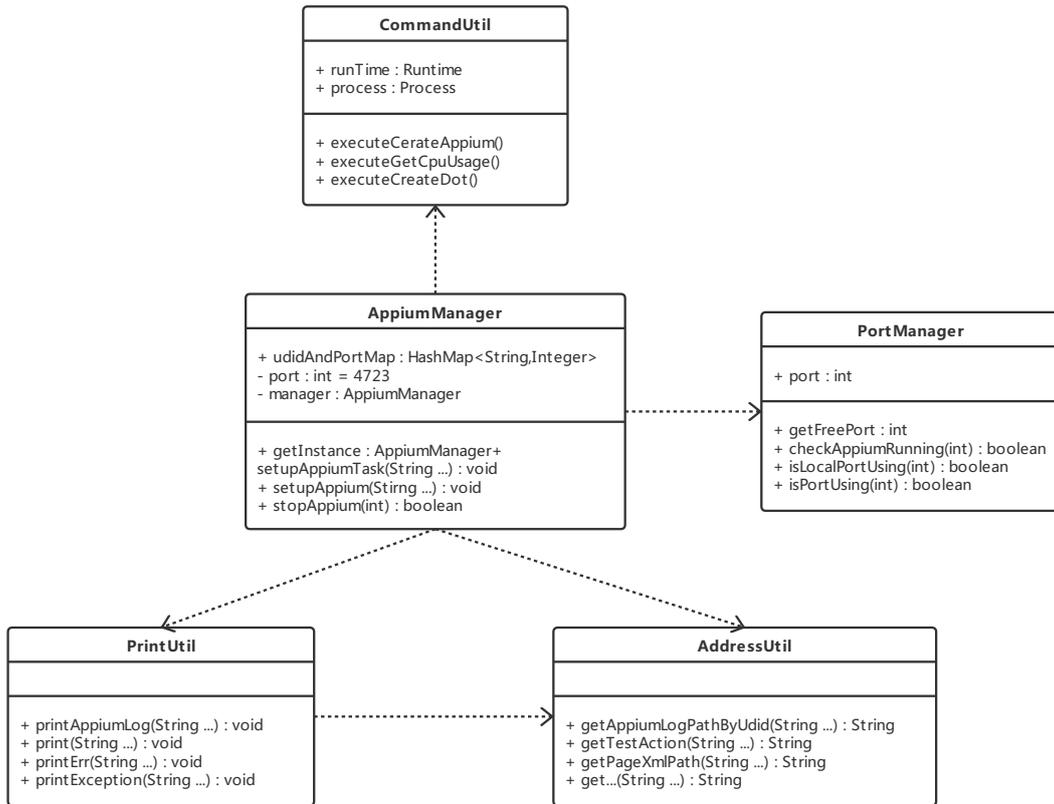


图 4.2: Appium 管理模块主要类图及部分方法

进行判断来确定 Appium 是否启动成功。当启动成功后，就可以将 port 号返回给申请启动 Appium 的测试类进行后续测试逻辑了。当测试完成后，直接调用 stopAppium 方法就可以停止该端口上 Appium 的运行。

启动 Appium 客户端方法的具体代码如图4.3所示，该方法包含整个启动流程，以及其中主要的各种判断细节。由于长度上的考量，一些文件的创建部分经过了省略，其余重要的操作内容都以注释的形式进行了解释。

```

public void setupAppium(String apkName,int taskId,String udid, String testLogsPath,
String appiumLogsPath, String exceptionLogsPath) {
    String port = checkDevicePort(udid);//检查设备是否已经开启Appium
    if (port == null) { //获取新的端口号并放入map
        port = getFreePort();
        udidAndPortMap.put(udid, port);
    }
}

```

```
// ..... 初始化各输出文件的部分代码
try {
    CommandUtil.executeCerateAppium();//开启Appium
    PrintUtil.printAppiumLog(log,output);//开始输出AppiumLog
} catch (IOException e1) {
    // ..... 异常处理的部分代码
}
//循环读取内容, 根据Log判断Appium是否启动成功, 直到成功开启为止跳出循环
try {
    FileReader fr=new FileReader(new
        File(AddressUtil.getAppiumLogsPathByUdid(apkName,taskId,udid)));
    BufferedReader br=new BufferedReader(fr);
    String line;
    boolean judge = true;
    while(judge){
        line=br.readLine();
        if(lineIsLegal(line)){
            judge=true;
        }else{
            judge=false;
        }
    }
} catch (FileNotFoundException e1 | IOException e) {
    // ..... 异常处理的部分代码
}
}
```

图 4.3: AppiumManager 类的启动 Appium 方法核心代码部分

4.2 自动化脚本流程获取模块详细设计与实现

4.2.1 自动化脚本流程获取模块概述

自动化脚本流程获取模块是获取用户操作流的方法之一。其作用在于，在用户执行自动化测试脚本时，记录下运行过程中，脚本每一步操作时页面的状态、操作本身的信息、被操作控件的信息，并将这些信息整合成自动化测试模块所需的操作流文件进行上传存储。简而言之，系统在运行的过程中，通过两个修改过后的 Appium 依赖包进行测试信息的截取，并保存下所有信息，最后将这些信息整合成整个系统的关键——用户操作流。

为保证测试数据不会产生遗漏，我们将控件相关的属性元素全部进行了保存，具体的数据内容如表4.2所示，这些信息是针对每一次控件操作时，系统所需要保存下的所有属性信息，根据这些信息，在之后的自动化测试过程中系统可以对控件进行唯一匹配以及复现。

表 4.2: 控件相关元素表

| 存储属性名 | 控件属性名 | 属性描述属性描述 |
|---------------|---------------|--|
| activity | 无 | 控件被点击前所处的 Activity 位置，不同的 Activity 中可能会出现属性完全一样的控件，但是相同 Activity 中只能存在一个，以此进行区分。 |
| behavior | 无 | 控件被进行的操作，与 Appium 的操作函数名对应。 |
| arg | 无 | 控件操作的参数，比如当操作为需要输入内容的操作，如向 TextView 输入内容，此处则会保存脚本中具体输入的内容值。 |
| index | index | 控件在所有相同层级控件中的序号。 |
| id | resource-id | 控件的唯一标识，在同一个 Activity 中，id 必然唯一。 |
| text | text | 控件的文本信息，如确定按钮上的“确定”二字。 |
| className | class | 控件本身的类信息，如 android.widget.ImageButton 代表一个按钮控件。 |
| desc | content-desc | 控件的描述信息，全称为 description，一般为开发者对该应用的描述，大部分情况为空。 |
| packageName | package | 控件在代码中所处位置的包名，大部分情况下就是应用包名。 |
| enabled | enabled | 布尔类型值，代表控件功能是否可用。 |
| checkable | checkable | 布尔类型值，代表控件是否可以被检查。 |
| checked | checked | 布尔类型值，代表控件目前是否被检查。 |
| clickable | clickable | 布尔类型值，代表控件功能是否可以被点击。 |
| focusable | focusable | 布尔类型值，代表焦点是否可以维持在该控件上。 |
| focused | focused | 布尔类型值，代表焦点是否已经维持在该控件上。 |
| longClickable | longClickable | 布尔类型值，代表控件功能是否可以被长按。 |
| scrollable | scrollable | 布尔类型值，代表控件功能是否可以被滑动。 |
| selected | selected | 布尔类型值，代表控件功能是否被选中。 |
| displayed | displayed | 布尔类型值，代表控件功能是否可见。 |

4.2.2 自动化脚本流程获取模块实现

自动化脚本流程获取模块的时序图如图4.4所示，首先，用户需要启动一定版本(大于 9.0)以上的 Appium 客户端，并将相关测试项目的 Appium 依赖包替换为系统接着在 ide 上运行自动化测试脚本，后续工作就由系统自动完成了。当测试流程开启后，通过对 Appium-java 这一 jar 包中的 AppiumDriver，以及 selenium-java 这一 jar 包中的 RemoteWebDriver、RemoteWebElement 类进行修改、插桩以及重新打包，每当脚本中的操作进行时，后台就会自动获取被操纵的 WebElement

的所有控件信息，并通过 Appium 函数自带的 `currentActivity()` 接口获取到执行操作前的页面信息。这样的获取操作会将每一次的操作信息从控件、页面以及脚本自身三个维度进行记录，直到测试脚本执行完成。而必须从三个维度同时记录信息的原因，是由于 Appium 执行各类方法的接口来源不同，需要分别从三个类种输出信息才能获取到完整的控件操作信息。

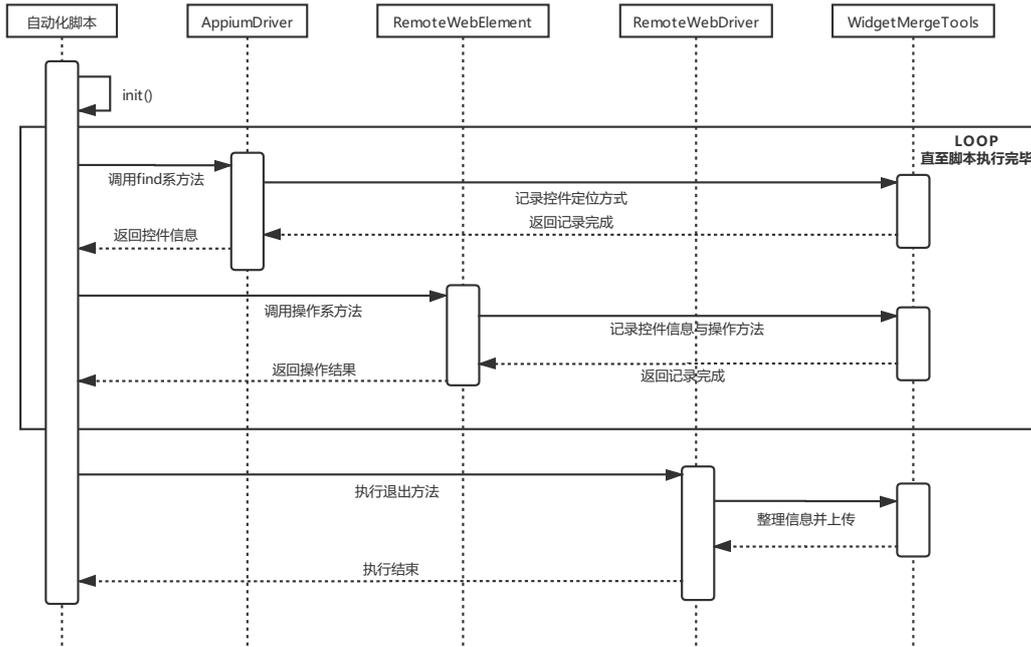


图 4.4: 自动化脚本流程获取模块时序图

自动化脚本流程获取模块涉及到的主要类如表4.3所示，AppiumDriver 类继承自 RemoteDriver 类，主要用来记录当前状态下的 Activity 情况，同时也可以通过 find 系的函数记录脚本中的代码获取到 WebElement 的方式。RemoteWebDriver 类是 WebDriver 接口的子类，当测试完成后，我们需要通过该类调用用户操作流程建模模块中的测试结果整理，将测试结果整合为应用操作流。RemoteWebElement 类则是 WebElement 类的子类，它用于收集 webElement 的所有信息，以及记录脚本中的操作信息，如 `click()`、`swipe(args[])` 等。在上述类执行各项自动化操作时，会调用 WidgetMergeTools 类中的各种记录方法，并将内容写在项目文件夹下的固定位置中，最后当测试完成时，AppiumDriver 会调用 `quit()` 方法，该方法内引用了 WidgetMergeTools 类用于整合所有测试过程生成文件的方法，并将内容上传至 OSS 内。

表 4.3: 自动化脚本流程获取模块主要类

| 类名称 | 分类 | 作用 |
|------------------|-----|--|
| WidgetInfo | 数据类 | 该类原则上属于用户操作流建模模块，但是由于本模块是在用户本地执行，因此相关功能被分离出来加入了本模块中。该类数据为一条用户操作信息，在最终会被用于合成完整的用户操作流文件。 |
| WidgetMergeTools | 工具类 | 该类原则上属于用户操作流建模模块，但是由于本模块是在用户本地执行，因此相关功能被分离出来加入了本模块中。该类包含了各项记录用户操作的方法，最终通过该类可以将用户操作整合为用户操作流文件并进行上传。 |
| WebElement | 接口类 | 代表着应用中的一个控件元素。 |
| RemoteWebElement | 服务类 | WebElement 接口类的具体实现，除了包含控件本身的属性外，还包含了针对控件的一些控制方法。 |
| WebDriver | 接口类 | 代表着自动化测试过程整体的控制器。 |
| RemoteWebDriver | 控制类 | WebDriver 接口类的具体实现，用于控制整个测试流程的控制器，负责页面上的元素查找，以及测试流程的建立与结束。 |
| AppiumDriver | 控制类 | 继承自 RemoteDriver，是具有 Appium 特色的测试控制器，对父类的大部分方法增加了 Appium 服务端的过滤，使得测试脚本的操作都会经过该类。 |

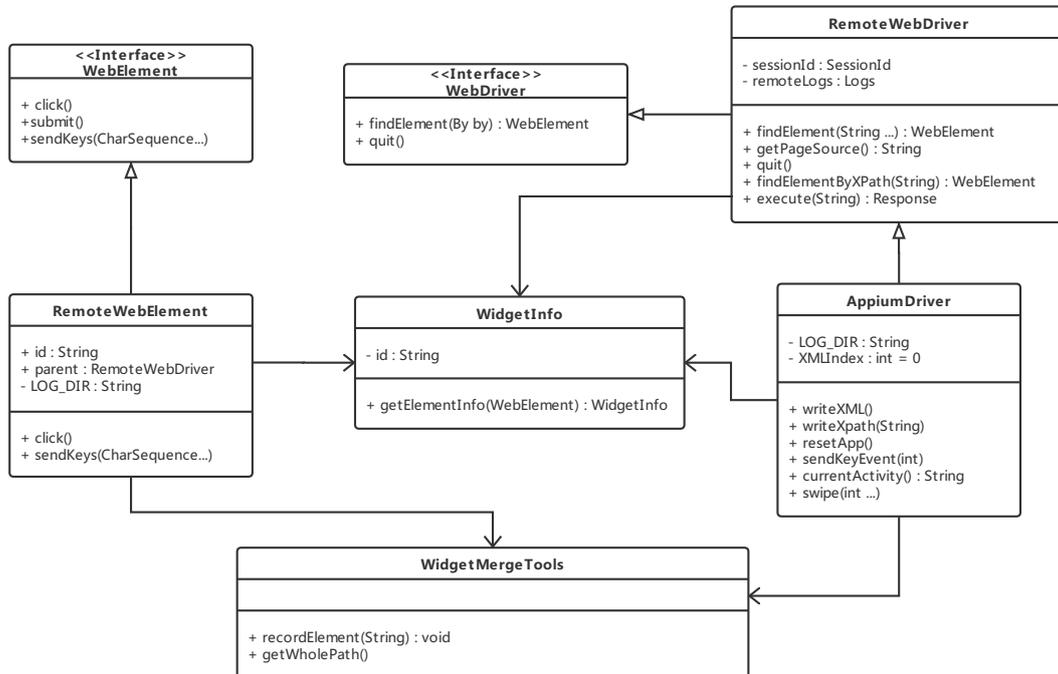


图 4.5: 自动化脚本流程获取模块类图及部分方法

这些类的关系如图4.5所示，当脚本运行的过程中，RemoteWebElement 类、RemoteWebDriver 类以及 AppiumDriver 类在执行 find 系方法、click、swipe 等方法时，系统会调用用户操作流建模模块中的 WidgetMergeTools 类中的 recordElement()、recordXml() 等方法，用来记录脚本的执行过程，将每一步的测试数据暂存到预设好的文件系统地址内。当全部测试执行完毕后，自动化脚本会调用 RemoteWebDriver 类的 quit() 方法，该方法位于修改后的 Appium 依赖包中，已经经过系统定制，因此会调用 WidgetMergeTools 类中的 getWholePath() 方法，将所有用户操作信息进行整合，最终将生成的用户操作流文件上传至 OSS 中对应该待测应用的文件夹内。

RemoteWebElement 类内对于该类原本方法的修改如图4.6所示，以 click 方法作为示例。相较于修改前的 click 方法，新方法增加了对 WebElement 元素所有属性的获取，Activity 的获取以及相关内容的存储。

```
public void click () {
    //获取输出地址，XML文件夹以时间戳方式命名
    //因此此处调用的函数内容为找到文件夹内最新的XML存放文件夹名
    String XMLName=WidgetInfo.getNowXML();
    //获取当前元素的信息，封装成WidgetInfo类
    WidgetInfo wi = WidgetInfo.getElementInfo(this);
    //为元素信息设置行为内容
    wi.setBehavior("click", null);
    //根据本次测试对应的时间戳获取命名并生成中间文件
    int num = Integer.parseInt(new File(XMLName).getName().split("\\.")[0]);
    String txtName = XMLName.substring(0, XMLName.lastIndexOf('\\'));
    txtName = txtName +File.separator+ num+".txt";
    //将本次操作信息记录到中间文件中，等待测试结束后一并处理
    wi.recordElement(txtName);
    //该函数原本的执行逻辑
    execute(DriverCommand.CLICK_ELEMENT, ImmutableMap.of("id", id));
}
```

图 4.6: RemoteWebElement 类的 click 方法内容修改及注释说明

4.3 自动化测试模块详细设计与实现

4.3.1 自动化测试模块概述

自动化测试模块为本系统的核心模块之一，当测试用户提出自动化测试申请时，本系统会对待测应用存在的最新格式化用户操作流进行整合，并合成一

颗应用控件覆盖树。接着，系统会先对应用进行一次自动化测试，并从此时开始记录下所有的测试流程。当第一次自动化测试完成后，自动化测试流程同样会生成一颗应用控件覆盖树。这两棵树会在系统内部进行对比，并查询出用户操作流中还未被覆盖到的控件。此时，系统会根据覆盖树复现用户操作，使应用到到达未被覆盖的控件处，接着再次将操作权返还给自动化测试逻辑。在经过多次循环后，用户操作流已经被全部覆盖的情况下，系统认为本次自动化测试结束，并停止记录测试流程，将全程记录下的测试流程进行整合，生成全新的用户操作流，并上传至 OSS 中进行存储。

4.3.2 自动化测试模块实现

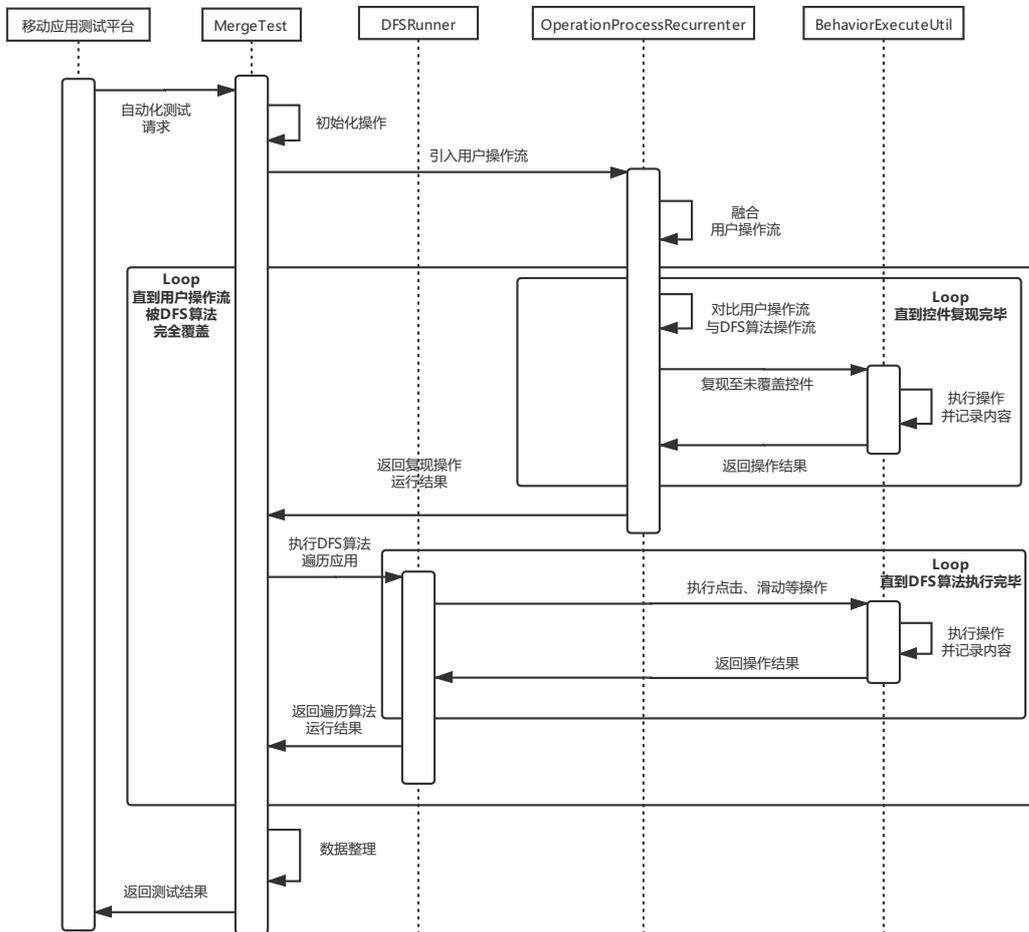


图 4.7: 自动化测试模块时序图

自动化测试模块开始运行后的时序图如图4.7所示。考虑到篇幅长度问题，

本图省略了迭代式自动化测试服务器接收到移动应用测试平台后的各种信息转接、DFSRunner 执行时的第一次自覆盖过程，以及一些初始化过程：如 Appium 的建立过程被省略，直接从引入格式化用户操作流开始进行。完整的执行流程是：MergeTest 类在初始化自身后，就会开始应用预测试逻辑，接着就会将应用控制权交给 DFSRunner 类和 OperationProcessRecurrenter 类，这两个类交互式的对应用进行操作，直到自动化测试结束为止。接着，MergeTest 类会将信息整合返回给移动应用测试平台。将自动化测试逻辑和通过用户操作流进行操作复现这两个方法拆开，是为了便于今后对 DFS 逻辑进行改进，将这两种方式进行拆分可以大大减少他们的耦合性，今后对两种逻辑进行修改时都会更加便捷，也可以直接使用更加优秀的移动应用遍历逻辑替换目前使用的简单 DFS 测试逻辑。

自动化测试模块涉及到的主要类如表4.4所示，MergeTest 类为整个测试流程的中枢，他负责进行任务的开启，测试控制权的调度以及测试结果的整理。DFSRunner 类则是一个完全 DFS 逻辑的安卓应用遍历器，它通过 BehaviorExecuteUtil 类进行统一的应用操作。由于 DFSRunner 内置了页面和控件的数据存储，因此并不会在每一轮的测试流程中对相同的控件进行操作。OperationProcessRecurrenter 类则负责对应用的用户操作流信息进行融合，并对这些操作进行复现。同时，他也负责对比自动化测试的操作流和应用的操作流，并找出哪些控件仍未被覆盖到。剩余的几个类均为数据类，Device 包含了待测设备的信息，Task 包含了本次任务的信息，Activity 为自动化测试算法在遍历的过程中为应用进行的建模，它包含了每一个 Activity 和其中包含的控件的信息，ActivityNode 则是建立完成的 Activity 模型，代表着 Activity 之间的到达关系。

这些类之间的关系如图4.8所示。首先，MergeTest 类在收到自动化测试请求后，会接收 Task 和 Device 信息并初始化自身。接着，MergeTest 会使用 install、coverInstall 等方法进行应用的设备兼容性测试，并开启整个自动化测试流程的记录。然后，MergeTest 类会将应用的控制权先初始化一个 DFSRunner 对象，并用其中的 DFS 方法进行一遍应用遍历。接着，MergeTest 类会调用 OperationProcessRecurrenter 类的 getResList 方法融合生成该应用的用户操作流，并开始一个 while 循环：先对比上一次 DFS 方法遍历的控件和用户操作流，找到覆盖过的控件，将这些控件从用户操作流内通过修改开关的方式屏蔽。接着 BFS 找到离用户操作流根节点最近的未被覆盖到的 Activity 所处的结点，通过 OperationProcessRecurrenter 类的 runOnePath 方法复现该路径。再接着将应用操控权交还给 DFSRunner 的 DFS 逻辑继续自动化测试。循环直到用户操作流内的控件全部被覆盖为止，认为测试结束。无论是 DFSRunner 类，还是 OperationProcessRecurrenter 类，他们对于应用的操作统一通过 BehaviorExecuteUtil 类提供的操作方法

表 4.4: 自动化测试模块主要类

| 类名称 | 分类 | 作用 |
|-----------------------------|-----|--|
| MergeTest | 控制类 | 负责控制整个自动化测试流程。开启各项测试预备进行, 控制测试过程中的应用控制权, 最后负责测试结果的整理。 |
| DFSRunner | 工具类 | 自动化测试算法, 可以从应用任意页面作为起点开展测试。 |
| OperationProcessRecurrenter | 工具类 | 负责用户操作流的融合, 生成应用控件覆盖树, 也负责进行覆盖树的比对, 并找出最近未覆盖页面, 以及将应用复现至该位置。 |
| BehaviorExecuteUtil | 工具类 | 统一化对应用的操作, 同时格式化输出测试过程。该类会调用一部分前文提到的 AddressUtil 类和 PrintUtil 类的方法, 使输出能够满足后续系统的格式化要求。 |
| Task | 数据类 | 移动应用测试平台的任务单元, 包含了任务 ID 等信息。 |
| Device | 数据类 | 包含了待测设备的全部信息。 |
| Activity | 数据类 | 包含了被记录下的页面情况。 |
| ActivityNode | 数据类 | 页面树的一个结点, 可以通过父子结点理清已被覆盖的页面之间的关系。 |

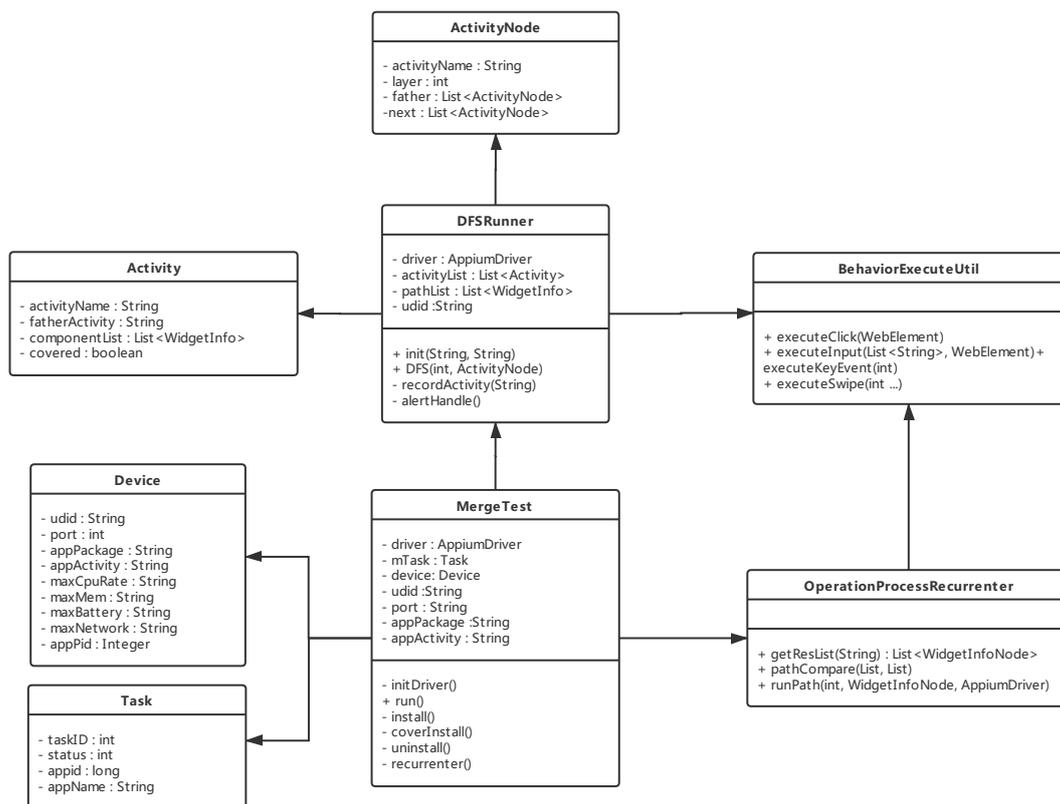


图 4.8: 自动化测试模块类图及部分方法

来达到格式化执行的目的。由于执行过程是完全自动化的，系统也能够得到格式化的输出，因此 MergeTest 类在测试结束后可以将整个自动化测试的操作信息进行整合，通过用户操作流建模模块的融合方法生成新的用户操作流并进行保存。最后，将移动应用测试平台需求的其他测试信息进行打包并上传。

DFSRunner 类中的 DFS 方法中，涉及到的格式化处理较多，如各种 Log 输出、每次执行操作后的 sleep 等待、父页面的验证以及 try-catch 语句等等。因此为了防止篇幅过长，此处省略了代码中各种非核心逻辑的部分，只留下了 DFS 算法最重要的一部分内容，具体的代码如图4.9所示。

```
public void DFS(int i,ActivityNode fatherNode){
    alertHandle();//处理弹窗
    //获取当前所有可点击控件
    List<WidgetInfo> widgetList = getCurrentClickablewidgetList(i, true);
    recordActivity(driver.currentActivity());//记录目前页面
    //查看当前页面是否重复，若控件重合率低于50%则认为来到了新的页面
    int currentActivityIndex = indexOfCurrentActivityInActivityList(currentActivity);
    if(currentActivityIndex >= 0){
        // ..... 本页面不是新页面，一些更新页面关系的操作
    }else{
        // ..... 本页面是新页面，一些添加新页面的操作
    }
    Activity thisActivity = activityList.get(currentActivityIndex);
    if(thisActivity.getCovered()){
        //该页面已经被测试过，返回上一层
        //具体的返回操作，通过点击左上角返回键或
        //点击2-3次系统返回键之后进行当前Activity是否成功返回的判断
        returnPreActivity();
        return;
    }else{
        for(int wIndex = 0; wIndex < widgetList.size();wIndex++){
            String locator = widgetList.get(wIndex).getLocator();
            WebElement element = driver.findElementByXPath(locator);
            if(locator.contains("EditText")){
                //搜索框或密码框，尝试输入
                List<String> inputWords= getInput();
                BehaviorExecuteUtil.executeInput(inputWords, element);
            }else{
                BehaviorExecuteUtil.executeClick(element);
            }
        }
    }
}
```

```

    alertHandle();//处理弹窗
    String activityAfterClick = driver.currentActivity();
    int pageHashAfterClick = driver.getPageSource().hashCode();
    if (pageHashAfterClick != currentPageHash){
        //点击后页面发生改变
        // ..... 一些页面处理操作
        //进入新的DFS逻辑
        DFS(i + 1, childNodeList.get(childNodeIndex));
    }
    // ..... DFS结束返回本页面后, 一些控件处理与防止出错的操作
}
}
//当前页面控件全部遍历完毕, 设置页面属性已覆盖
activityList.get(currentActivityIndex).setCovered(true);
returnPreActivity();
return;
}
}

```

图 4.9: DFSRunner 类的 DFS 测试算法以及核心代码

OperationProcessRecurrenter 类中的代码复现方法则如图4.10所示, 由于覆盖树上的结点包含了 father, 即上一个结点的信息, 而上一个结点内又包含该结点到达本结点时所做的操作, 因此, 系统可以直接根据这些内容调用 BehaviorExecuteUtil 类的操作接口进行设备操作。

```

public static void runPath(int maxSize, WidgetInfoNode node, AppiumDriver driver){
    //node为需要到达的控件, maxSize为覆盖树的最大深度
    WebElement we;
    String using;
    try {
        Stack<WidgetInfoNode> path = new Stack<>();
        //得到从root到达node的最短路径
        //该方法为一个简单的BFS搜索, 从node结点进行倒推
        //找到最近的一条倒推回root结点的路径
        path = getPath(0, maxSize, new Stack<WidgetInfoNode>(), node);
        //由于入栈顺序为node, node的father, ..., root
        //因此直接出栈, 顺序即为正序
        path.pop();//去除空结点root
    }
}

```

```
while (!path.empty()) {
    Thread.sleep(3000);
    WidgetInfoNode wInfoNode= path.pop();
    wInfoNode.setVisited(true);
    WidgetInfo wInfo = wInfoNode.widgetInfo;
    // ..... 根据id、text、className、desc和package修改定位属性using
    // ..... 根据定位属性using定位控件，查看控件是否存在
    // ..... 控件存在，即对using进行一些拼接处理
    try {
        //根据控件信息中操作内容的不同，调用不同的操作执行方式
        if (wInfo.behavior.equals("click")) {
            we = driver.findElementByXPath(using);
            BehaviorExecuteUtil.executeClick(we);
        } else if (wInfo.behavior.equals("sendKeys")) {
            we = driver.findElementByXPath(using);
            BehaviorExecuteUtil.exectureInput(wInfo.behaviorArgs,we);
        } else if (wInfo.behavior.equals("sendKeysEvent")) {
            BehaviorExecuteUtil.executeKeyEvent(
                Integer.parseInt(wInfo.behaviorArgs.get(0)));
        } else if (wInfo.behavior.equals("swipe")) {
            ArrayList<String> args = wInfo.behaviorArgs;
            BehaviorExecuteUtil.executeSwipe(args);
        }
    } catch (NoSuchElementException ex){
        PrintUtil.print("can not find" + using, udid, myTestLogWriter,
            PrintUtil.ANSI_RED);
    }
    return;
} catch (Exception e){
    PrintUtil.print("rerun failed", udid, myTestLogWriter,
        PrintUtil.ANSI_RED);
}
}
```

图 4.10: OperationProcessRecurrenter 类的复现用户操作方法以及核心代码

4.4 用户操作流建模模块详细设计与实现

4.4.1 用户操作流建模模块概述

用户操作流建模模块为本系统的核心模块之一。其涉及的功能较广，包括融合自动化脚本流程获取的结果、整理自动化测试过程的测试流、将多条操作流进行融合以及将现有的操作流整理成测试人员可读的图形。只要是涉及到处理用户操作流有关的操作，几乎都和本模块有所关联。

4.4.2 用户操作流建模模块实现

由于本模块的核心操作功能几乎都聚集在 `WidgetMergeTools` 类中，大部分为零散的内容处理方法，成型的使用路径较少，因此从此处列举其中之一：通过本模块生成应用控件覆盖树流程的时序图，如图4.11所示。

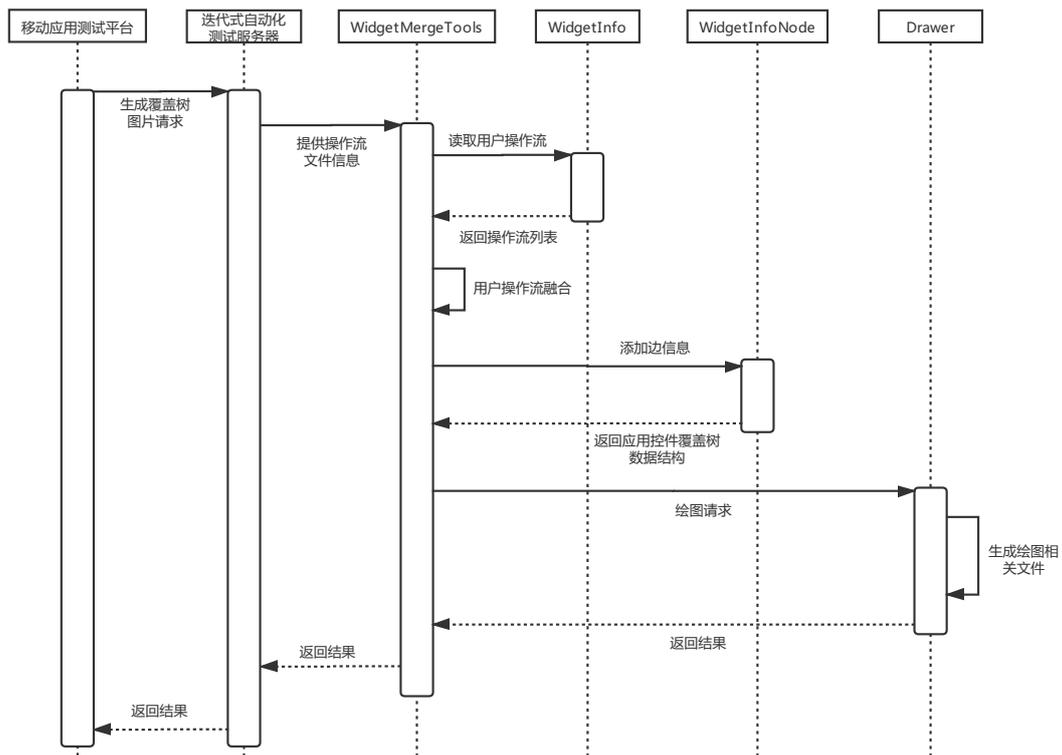


图 4.11: 用户操作流建模模块时序图

本模块的其余功能较为分散，部分流程与本流程相似，也有部分流程属于本流程中的一部分，被单独抽取出来直接使用，因此只列举本流程作为示例。

在系统收到生成覆盖树图片的请求后，迭代式自动化测试服务器会将相关的信息提供给 `WidgetMergeTools` 类，首先，该类会先格式化读取与待测应用相关的所有操作流，为每一条操作流在代码层面生成一条 `WidgetInfo` 对象的操作流列表。接着，将这些列表中的结点进行去重，生成一条只包含不重复操作流结点的 `WidgetInfoNode` 对象的答案列表。紧接着根据操作流列表将边的信息加入到答案列表中，在加入的过程中进行去重，最终能得到一条包含所有结点和边信息的 `WidgetInfoNode` 对象的操作流答案列表。由于拥有边信息，该答案列表即可以看作应用的控件覆盖树。这样的方法相较于将操作流列表进行两两合并的方式，能够节省更多的时间。当覆盖树，即一条 `WidgetInfoNode` 的列表生成完毕后，系统会通过 `Drawer` 类生成绘图所需的 `.dot` 文件与 `.Json` 文件，并通过脚本进行操作，前者会直接生成 `PNG` 文件结果，后者则会在结果报告读取时生成关于控件覆盖的力引导图。

表 4.5: 用户操作流建模模块主要类

| 类名称 | 分类 | 作用 |
|-------------------------------|-----|--|
| <code>WidgetInfo</code> | 数据类 | 包含了用户操作流中的结点信息，即被点击控件的全部信息，以及对该控件的操作内容。 |
| <code>WidgetInfoNode</code> | 数据类 | 在 <code>WidgetInfo</code> 的基础上增加了父子结点，帮助用户操作流进行去重。通过该对象，我们可以了解到目前应用已经被覆盖到的控件信息，以及这些控件之间的到达关系。 |
| <code>ToolsFormatter</code> | 接口 | 用于接入其他可输出用户操作流工具的接口。 |
| <code>LIRATReader</code> | 服务类 | 可以将 LIRAT 产生的用户操作流进行读取，生成系统所需的 <code>WidgetInfoNode</code> 列表。 |
| <code>WidgetMergeTools</code> | 工具类 | 包含了对 <code>WidgetInfo</code> 对象以及 <code>WidgetInfoNode</code> 对象所有处理方法的类。向外提供了 <code>WidgetInfo</code> 对象的融合、去重， <code>WidgetInfoNode</code> 对象的读取和复现方法等等。 |
| <code>Drawer</code> | 服务类 | 用来绘图的类，输入为 <code>WidgetInfoNode</code> 列表，输出为应用的控件覆盖树图片。 |

用户操作流建模模块涉及到的主要类如表4.5所示。`LIRATReader` 类用来读取 LIRAT 记录下的操作流。LIRAT [43] 是一款网页端操作安卓设备，并记录用户操作流的工具，目前同样配置在了移动应用测试平台上。由于工具作者与本文作者为同实验室成员，因此根据本文作者要求，该工具可以获取系统所需格式的用户操作流信息，在经过该类将信息格式化后，便可以成为本系统所需的用户操作流文件。由于将来不排除为其他工具进行修改，同样获取到本系统所需的用户操作流，因此增加了 `ToolsFormatter` 接口类，成为 `LIRATReader` 类的父类。`WidgetInfo` 类是包含一个控件全部属性的类，除开属性，他也包含了对这个控件的操作内容以及操作参数。`WidgetInfoNode` 类则在 `WidgetInfo` 的基础上

加上了边信息，由于每个 `WidgetInfo` 对象对应着对唯一控件的唯一操作，因此 `WidgetInfoNode` 类在大多数情况下只有一个子结点（也有可能多个，比如输入正确与错误的密码后，点击登录控件，会通向不同的子结点），但是可能有多个父结点。`WidgetMergeTools` 类用来进行各种 `WidgetInfo` 类以及 `WidgetInfoNode` 类相关的操作，如将多条 `List<WidgetInfo>` 融合为一颗覆盖树 `List<WidgetInfoNode>`。`Drawer` 则负责进行绘图方面的功能，将融合用户操作流后生成的应用控件覆盖树展现到测试人员的眼前。

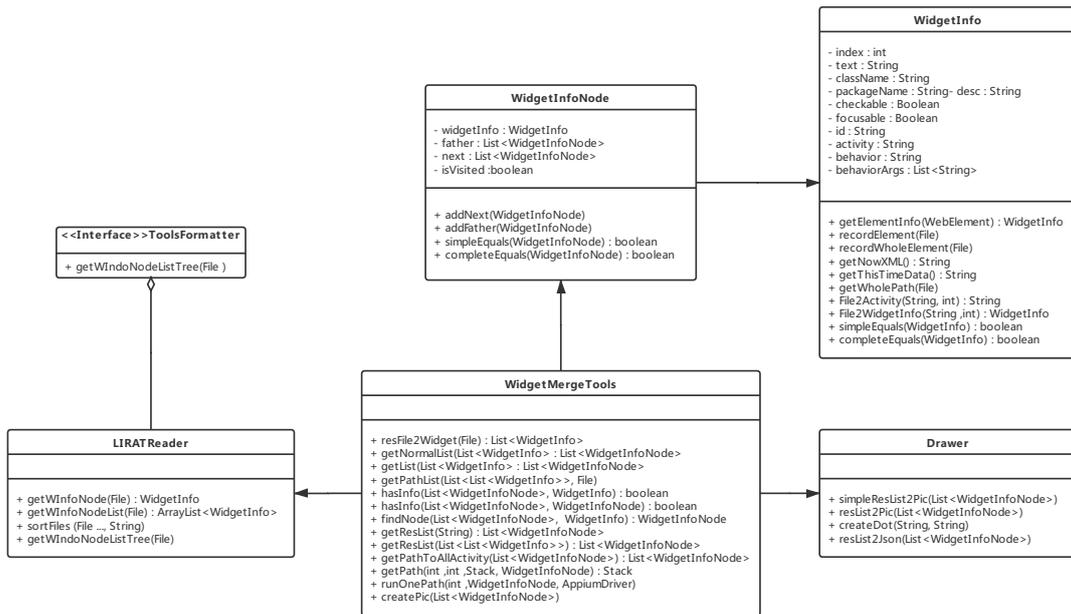


图 4.12: 用户操作流建模模块类图及部分方法

这些类的关系如图4.12所示，当系统需要对当前的用户操作流进行融合时，迭代式自动化测试服务器会调用 `widgetMergeTools` 中的 `getPathList` 的方法，先将每一份文件中的操作流数据化，转换为 `List<WidgetInfo>` 格式，并将它们再加入一个 `List` 中，接着调用 `getResList` 方法，对这些操作流继续一一合并。具体的代码内容在之后的代码列表中进行详细阐述。最终，系统会获得一个控件列表 `List<WidgetInfoNode>`，其中包含了控件本身的全部信息以及控件之间的前后连接关系。如果系统此时想将覆盖树与自动化测试的操作流进行比对，会使用 `getPathToAllActivity` 方法，首先获取到目前已经被覆盖到的页面信息，接着和自动化测试覆盖到的页面信息进行比对，将已经被覆盖的页面去除，在调用 `runOnePath` 方法对未覆盖的路径一条条的进行匹配与复现，同时调用自动化测

试逻辑。如果想要将控件覆盖树生成为图片，系统则会调用 `createPic` 方法，通过调用 `Drawler` 类中的 `resList2Pic` 方法，先向文件中写入结点信息，再写入边信息，最后调用 `createDot` 方法使用脚本生成覆盖树图片。而想要生成力引导图的话，则是调用 `resList2Json` 方法，生成前端所需的 `Json` 文件。

用户操作流融合的相关代码如图4.13所示，首先，系统会扫描一遍全部路径中包含的结点，并将这些结点全部加入到答案列表中，在加入的过程中进行去重，即如果发现下一个要加入的结点是重复的，则不加入。接着加入边关系，每一条 `ArrayList<WidgetInfo>` 中，其索引顺序就代表了结点的边关系，系统会扫描所有的前后结点，并将边在不重复的情况下加入答案列表中，这样就能生成所有的父到子的关系。最后，通过父到子的关系反向添加子到父的关系，就能完成这条双向链表。从绘制角度来看，这条双向链表所形成的树状图就是系统所需的应用控件覆盖树了。

```
public static ArrayList<WidgetInfoNode>
getResList(ArrayList<ArrayList<WidgetInfo>> pathList){
    //输入值为操作流列表，其中：widgetInfo即为一个控件的全部信息
    //ArrayList<WidgetInfo>代表了一条操作流，他们的List就是全部的操作流信息
    //本函数的任务为生成一条ArrayList<WidgetInfoNode>
    //其中包含了全部结点与结点之间的相连关系，即应用控件覆盖树
    ArrayList<WidgetInfoNode> res = new ArrayList<>();
    WidgetInfoNode head = new WidgetInfoNode();//定义头节点
    //加入所有路径中的所有节点，同时去重
    for(int i =0;i<pathList.size();i++){
        ArrayList<WidgetInfo> path = pathList.get(i);
        for(int j =0;j<path.size();j++){
            WidgetInfo wInfo = path.get(j);
            //在原本list里查重
            if(!hasInfo(res,wInfo)) {
                WidgetInfoNode wInfoNode = new WidgetInfoNode(wInfo);
                if (j == 0) {
                    //此处使用的add方法自带结点查重及去重
                    head.next.add(wInfoNode);
                }
                res.add(wInfoNode);
            }
        }
    }
}
```

```

res.add(head);
//加入所有路径关系
//扫描所有路线，w1和w2指向前后用来添加关系
for(int i =0;i<pathList.size();i++){
    ArrayList<WidgetInfo> path = pathList.get(i);
    for(int j =0;j<path.size()-1;j++){
        WidgetInfo w1 = path.get(j); WidgetInfo w2 = path.get(j+1);
        WidgetInfoNode wn1 = findNode(res,w1);
        WidgetInfoNode wn2 = findNode(res,w2);
        wn1.addNext(wn2);//同理，此方法自带去重功能
    }
}
//逆向加入父关系
for(int i =0;i<res.size();i++){
    WidgetInfoNode nowNode = res.get(i);
    ArrayList<WidgetInfoNode> next = nowNode.next;
    for(int j =0;j<next.size();j++){
        WidgetInfoNode nowNext = next.get(j);
        nowNext.addFather(nowNode);
    }
}
return res;
}

```

图 4.13: widgetMergeTools 类的将操作流进行融合 getResList 方法

生成可读树形图的相关代码则如图4.14所示，主要内容为配合 DOT 的语法，写入相关的结点和边的信息，最后生成 PNG 格式的图片。生成 Json 文件（用于在前端生成力导图）的方法流程和本方法类似，因此不再赘述。

```

public static void resList2Pic(ArrayList<WidgetInfoNode> resList) {
    //dot文件和图片均生成在默认位置
    File dotFile = new File(DIR_NAME+DOT_NAME+".dot");
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(dotFile));
        Set<String> activityName = new HashSet<>();
        //先读取所有的Activity信息，为dot文件添加分块
        for (int i = 0; i < resList.size(); i++){
            // ..... 记录所有Activity信息的操作
        }
    }
}

```

```

String [] activityNames = activityName.toArray(new
    String[activityName.size()]);
// ..... 向输出文件中写入图片配置内容的操作
//write subgraph
for(int i = 0;i<activityNames.length;i++){
    // ..... 向输出文件中写入Activity相关配置的操作
    for (int j = 0; j < resList.size();j++){
        // ..... 向输出文件中写入属于该Activity的结点内容的操作
    }
// ..... 向输出文件中写入其他配置项的操作
//向输出文件中写入边信息
for (int i = 0; i < resList.size();i++) {
    WidgetInfoNode wInfoNode = resList.get(i);
    ArrayList<WidgetInfoNode> nextList = wInfoNode.getNext();
    for(int j = 0 ;j<nextList.size();j++){
        WidgetInfoNode nextNode = nextList.get(j);
        int nextIndex = resList.indexOf(nextNode);
        bw.write("    "+i+"->" +nextIndex+"\n");
        bw.flush();
    }
}
// ..... 结尾的一些处理操作
}catch(Exception e){
    e.printStackTrace();
    PrintUtil.print("draw dot failed", udid, myTestLogWriter,
        PrintUtil.ANSI_RED);
}
}
}

```

图 4.14: DOTDrawer 类的生成控件覆盖树图片 resList2Pic 方法

4.5 本章小结

本章将本系统的四个核心模块：Appium 管理模块、自动化脚本流程获取模块、自动化测试模块以及用户操作流建模模块的详细设计以及实现进行了具体的介绍。首先，本章介绍了这些模块的时序图，从使用流程上讲解了系统核心模块的功能。接着，本章给出了这些模块的核心类说明以及核心类图，从类的分工进行模块功能的讲解和说明。最后，本章给出了每个模块中较为常用的方法的具体代码，从代码流程上讲解了这些模块的使用流程。

第五章 系统测试与实验分析

为了验证迭代式安卓自动化测试系统的可用性，本文从两个方向对本系统进行测试。其一为功能测试，通过在实验室移动测试平台上运行本系统，来检验本系统的流程是否可靠。功能测试的具体测试用例通过与本文第三章所述的用户用例一一对应，以确保系统的可用性。如若在测试过程中未出现报错，且所有输出数据没有产生遗漏，则可以说明本系统的真实可用。

其二为效果检验。本文开展了一次检验系统自动化测试效果的实验，使用 50 份针对 10 款世界级安卓应用的操作流程作为系统输入，以未输入脚本前与输入脚本后系统的控件覆盖率、页面覆盖率进行比较。由于覆盖率只是一种检验效果的方式，因此获取应用覆盖率的流程并不被包含于系统的执行流程内。安卓目前没有一个能够直接获取到应用覆盖率的工具，我们只能从应用侧获取覆盖率来解决这个问题。最终，我们选择使用 Jacoco 为这些应用的源码进行手动插桩，并生成修改过后的 APK 包，从而在测试的过程中获取到应用的各项代码覆盖率，这些覆盖率数据会在应用被执行后，自动生成在移动设备的固定存储空间中，接着，我们会手动将这些数据全部进行提取并进行 Jacoco 的覆盖率输出处理，即可获得测试过程的各项覆盖率。同时，我们引入了 Monkey 工具，通过横向对比的方式从侧面进行实验结果对照，从而检验本系统的能力。通过开展实验，本文希望能够验证以下几个问题，并以此说明系统的可用性：

- 问题一: 本系统将用户操作流进行融合后，其中的用户操作信息是否会有所丢失？
- 问题二: 本系统的自动化测试效果在输入人工操作流后，对比未输入时是否有所提升？
- 问题三: 相较于其他测试工具，本系统的自动化测试工具结合用户操作流之后，能否赶超其覆盖率？

5.1 测试环境

5.1.1 功能测试环境

如表5.1所示，该表展示了本系统展开功能测试时的测试环境，包含了部署服务器以及相关移动设备等硬件的全部信息。本测试模拟了真实用户环境，部

署于实验室专门设立的机柜服务器上进行，该机柜可同时连接 10 台以上的移动设备，因此测试所使用的移动设备均为实机。本系统可单独部署于 Windows 或 Linux 系统上，系统测试于实验室 Linux 系统下进行。本系统采用 MySQL 数据库记录简短信息，文件信息则存储在云上 OSS 内。模块实现方面均基于 Java 语言进行实现，与安卓设备的连接则由 USB 实现通讯。

表 5.1: 测试环境

| | |
|-----------|---------------------------------|
| 服务器信息 | 内存 16GB, 带宽 10MB |
| 部署系统 | Ubuntu 16.04 |
| 数据库 | MySQL 5.7, 阿里云 OSS |
| JDK 版本 | JDK 1.8 |
| Node 版本 | Node 8.12.0 |
| Appium 版本 | 1.9.1 |
| 安卓设备品牌 | 华为, 小米, 三星 |
| 安卓设备操作系统 | Android 5.0, 5.1, 6.0, 7.0, 8.0 |

5.1.2 实验待测应用信息

本文实验中采用的安卓应用，均来源于安卓谷歌商店，取自应用排行榜。本文根据谷歌商店当月排行榜，使用伪随机数算法随机选取了排行榜中的 10 个应用。待测应用的具体数据如表 5.2 所示。

表 5.2: 待测应用信息

| 编号 | 名称 | 应用类型 | 应用版本 | 应用下载量 |
|-----|---------------|-------------|------------|----------|
| A1 | Wikipedia | Internet | v2.7.50305 | 1000 万 + |
| A2 | Monkey | Development | v1.0.0 | 500 万 + |
| A3 | Jamendo | Media | v1.0.4 | 50 万 + |
| A4 | VLC | Media | v3.2.7 | 1 亿 + |
| A5 | MatomoMobile2 | Development | v2.4.5 | 5 万 + |
| A6 | openHAB | Internet | v2.10.3 | 10 万 + |
| A7 | OsmAnd | Map | v3.5.5 | 500 万 + |
| A8 | Linphone | Phone + SNS | v4.2 | 50 万 + |
| A9 | AdGuard | System | v3.3 | 500 万 + |
| A10 | Kiwix | Internet | v3.1.0 | 50 万 + |

表中包含了这些应用的名称、类型、版本、下载量等信息。在之后的实验中，我们会记录系统在这些应用上运行时，所覆盖的 Activity 覆盖率，并与成熟测试工具进行对比。同时，为了得到更准确的信息，我们选用一款较为成熟的应用覆盖率检测工具，Jacoco 工具来进行覆盖率的获取。对于能与该工具进行匹

配的应用，我们对它们进行预处理后打包生成了修改后的 Apk，从而获取到详细的覆盖率信息。

5.2 测试过程和实验设计

5.2.1 功能测试

系统功能测试根据需求分析与概要设计中描述的系统功能进行了用例设计，覆盖到了本系统的各项功能，以保证本系统的使用情况能够符合预期。各测试用例描述表详情如下：

表5.3为申请自动化测试任务的相关功能，包含测试用户从登录系统开始，到上传待测应用、申请测试任务的这一流程。测试用户需要先登录系统，接着选中申请自动化测试任务选项，在新出现的页面中上传应用、填写好测试信息，最终点击申请任务，完成整个测试用例。

表 5.3: 申请自动化测试任务测试用例

| 描述项 | 说明 |
|------|--|
| 用例编号 | TC1 |
| 用例名称 | 申请自动化测试任务 |
| 用例描述 | 用户上传待测应用，并申请自动化测试任务。 |
| 测试步骤 | 1. 测试用户登录系统，选择申请自动化测试任务。 2. 测试用户上传待测应用，填写相关测试信息，发出测试申请。 |
| 预期结果 | 1. 任务列表中增加该任务。 2. 管理员可以对该任务进行审核。 |

表5.4为审核自动化测试任务流程，包含管理员从登录系统开始，到为审核中任务上传用户操作流文件、审核任务的整个流程。管理员需要先登录系统，接着进入任务列表界面，选中需要审核的测试任务，点击上传用户操作流按钮并上传相应文件，最后点击审核按钮完成对该测试任务的审核。

表 5.4: 审核自动化测试任务测试用例

| 描述项 | 说明 |
|------|---|
| 用例编号 | TC2 |
| 用例名称 | 审核自动化测试任务 |
| 用例描述 | 管理员为审核中的自动化测试任务上传用户操作流，并通过该任务的审核。 |
| 测试步骤 | 1. 管理员登录系统，进入任务列表页面。 2. 管理员为一条未审核的任务上传用户操作流。 3. 管理员通过该任务审核。 |
| 预期结果 | 1. OSS 中增加待测应用相关的用户操作流。 2. 任务状态转变为审核完成，等待用户选择开始执行。 |

表5.5为测试执行自动化脚本流程，包含测试用户在替换测试项目 jar 包，执行自动化脚本，到执行完毕后检验数据是否成功生成的过程。测试用户首先需要替换自己 Appium 项目中的两个依赖包，接着运行自动化脚本。当脚本正常跑完后，观察项目的特定目录下是否生成了相应的用户操作流文件，以及观察 OSS 中是否成功上传了该文件。

表 5.5: 执行自动化脚本测试用例

| 描述项 | 说明 |
|------|---|
| 用例编号 | TC3 |
| 用例名称 | 执行自动化脚本 |
| 用例描述 | 用户替换依赖包之后执行自动化测试脚本。 |
| 测试步骤 | <ol style="list-style-type: none"> 1. 测试用户将测试项目的 Appium 依赖包替换为本系统提供的依赖包。 2. 测试用户执行自动化测试任务。 |
| 预期结果 | <ol style="list-style-type: none"> 1. 项目的 lib 文件夹下生成了关于本次测试的执行信息文件与融合后的用户操作流文件。 2. 测试流文件被上传至 OSS 中该应用对应的文件夹下。 |

表5.6为测试自动化测试任务执行流程，包含从登录系统开始，到执行自动化测试任务、连接设备开始启动、测试结束的整个流程。测试用户在登录完成后，进入任务列表界面，直接点击任务后跟随的执行按钮即完成了全部操作。在这之后，需要观察连接服务器的设备是否开始执行自动化工具逻辑，即开始自动的跑起了这个应用。

表 5.6: 自动化测试任务执行测试用例

| 描述项 | 说明 |
|------|---|
| 用例编号 | TC4 |
| 用例名称 | 自动化测试任务执行 |
| 用例描述 | 测试用户开始执行某一个审核后的测试任务，系统自动在连接的移动设备上自动化测试并返回测试结果。 |
| 测试步骤 | <ol style="list-style-type: none"> 1. 测试用户登录系统，进入任务列表。 2. 测试用户为某一个审核后的任务选择开始测试。 |
| 预期结果 | <ol style="list-style-type: none"> 1. 连接的移动设备上开始进行自动化测试。 2. 测试的任务状态修改为已完成。 3. 点击任务可以查看相关的测试结果页面。 |

表5.7为测试查看自动化测试结果的过程，包含测试用户从登录开始，到选择查看测试结果的全部过程。测试用户在登陆后，点击任务列表，在已完成测试的任务后会出现查看任务列表按钮。用户点击该按钮后查看输出的测试结果是否正确即可以完成该测试用例。

表 5.7: 查看自动化测试结果测试用例

| 描述项 | 说明 |
|------|--|
| 用例编号 | TC5 |
| 用例名称 | 查看自动化测试结果 |
| 用例描述 | 用户上传待测应用，并申请自动化测试任务。 |
| 测试步骤 | 1. 测试用户登录系统，进入任务列表。 2. 测试用户点击查看测试结果。 |
| 预期结果 | 1. 打开测试结果页面，其中可以查看整体测试情况、每一款连接设备上的测试情况以及应用的覆盖情况。 |

5.2.2 实验设计

本系统的测试结果验证流程设计如下：

- (1) 随机挑选 10 款有名的安卓应用作为待测应用。
- (2) 寻找这些应用中，在 github 上存在源码的例子，将这部分应用的源码使用 Jacoco 进行预处理，并进行打包生成插桩后的 apk。
- (3) 使用未引入用户操作流的系统自动化测试算法对这些应用进行测试，保存覆盖率信息。
- (4) 使用其他成熟自动化测试工具对这些应用进行测试，保存覆盖率信息。
- (5) 为这些安卓应用创建用户操作流信息，每一款创建 5 份信息，并上传至 OSS 对应文件夹下。
- (6) 使用引入用户操作流的系统自动化测试算法对这些应用进行测试，保存覆盖率信息。
- (7) 统计上述几种测试方式下的测试覆盖率，并进行相互比较。

对于步骤 2，由于安卓应用的特殊性，目前没有一款工具能够得到黑盒应用（即只有 apk 文件而没有源码的情况）的覆盖率。因此，必须要使用开源应用，对应用进行预处理，才能够获得应用覆盖率。

对于步骤 4，本文选择使用 Monkey，一款目前最为成熟的自动化测试工具进行对比。根据以往研究者的实验结果 [41]，Monkey 在运行一小时左右，覆盖率即会达到一个相对稳定的峰值，之后仍然会有所提升，但是幅度较小。由于 Monkey 并没有自带停止命令，因此我们编写了一段脚本，让 Monkey 无休止的运行，脚本则在监听 1 小时后，使用命令行强制关闭 Monkey 进程。执行完毕后，手动收集手机中的覆盖率信息。

对于步骤 5，我们使用前文所述的 Lirat 工具，请几位测试者在相关应用上进行了少量的点击，从而生成系统所需的用户操作流信息，并在后续通过本系

统的建模模块进行处理以生成用户操作流文件。为了证明少量操作就能引导工具提高大量覆盖率，我们要求每一次执行流程的操作固定为 15 个操作（如点击、滑动、输入，均记为一次操作，而一份由专业人员在 2 小时内完成的自动化测试脚本，通常会包含上百个操作），具体操作内容由测试人员自行发挥，但是建议每一条路径之间不应有或只有少量重合。

5.3 测试结果

5.3.1 功能测试结果与系统展示

表5.8记录了测试用例与系统用例之间的对应关系以及测试结果，测试人员严格按照上述测试用例步骤来执行这些测试用例，最终发现所有用例均符合预期结果。这能够证明本系统已经成功的完成了需求分析中所提出的所有的功能需求，并且能够良好的嵌入到实验室的移动应用平台总系统中，实现改进自动化测试算法的效果。

表 5.8: 测试用例执行结果

| 测试编号 | 用例编号 | 测试结果 |
|------|-------------|------|
| TC1 | UC1、UC2 | 通过 |
| TC2 | UC3、UC4 | 通过 |
| TC3 | UC5、UC7 | 通过 |
| TC4 | UC6、UC7、UC8 | 通过 |
| TC5 | UC9 | 通过 |



图 5.1: 上传任务页面

如图5.1所示，为上传待测应用的系统界面截图，用户需要为该自动化测试任务上传待测应用以及测试信息，接着点击提交测试，将申请传输到系统内。

如图5.2所示，为审核任务的页面。当测试用户上传任务申请成功后，管理员就可以在任务列表中选中该任务，并进行用户操作流的上传以及审核操作了。



图 5.2: 审核任务页面

如图5.3所示，为上传用户操作流文件的页面。在审核应用的过程中，管理员可以为该测试任务上传用户操作流信息，增强本次自动化测试的测试效果。



图 5.3: 上传用户操作流页面

点击执行任务后，连接至服务器的移动设备上就会开始进行自动化测试。当任务执行完成后，会生成相应的测试结果页面，如图5.4所示。该页面使用的是

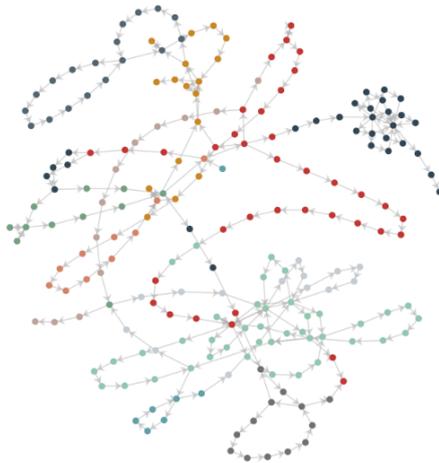
移动应用平台统一的测试结果模板，只需要传递正确的 Json 格式测试结果文档就可以得到该页面结果。



图 5.4: 测试结果页面

控件操作流程图

- org.wikipedia.main.MainActivity
- org.wikipedia.feed.configure.ConfigureActivity
- org.wikipedia.settings.languages.WikipediaLanguagesActivity
- org.wikipedia.page.PageActivity
- .BrowserActivity
- org.wikipedia.settings.SettingsActivity
- org.wikipedia.settings.AboutActivity
- org.wikipedia.readinglist.ReadingListActivity
- org.wikipedia.theme.ThemeFittingRoomActivity
- org.wikipedia.descriptions.DescriptionEditActivity
- org.wikipedia.gallery.GalleryActivity
- org.wikipedia.login.LoginActivity
- org.wikipedia.descriptions.DescriptionEditTutorialActivity
- org.wikipedia.onboarding.InitialOnboardingActivity
- org.wikipedia.language.LangLinksActivity
- org.wikipedia.createaccount.CreateAccountActivity



应用控件覆盖流程图

图 5.5: 页面信息页面

在原本测试结果的基础上，本文还增添了关于本次测试中，控件的覆盖情况，如图5.5与图5.6所示，控件覆盖情况可以在两种粒度下进行显示。前者为粗粒度下的控件覆盖信息，主要展示了页面信息和控件之间的关系，将鼠标移动至节点上可以观察到每一个节点，即控件的详细信息，箭头则代表了它们在执行过程中的执行顺序。后者则是详细的控件信息，包含了每一个控件的各项属性以及它们之间的执行顺序。这些信息会被留存下来，供之后的实验进行使用。

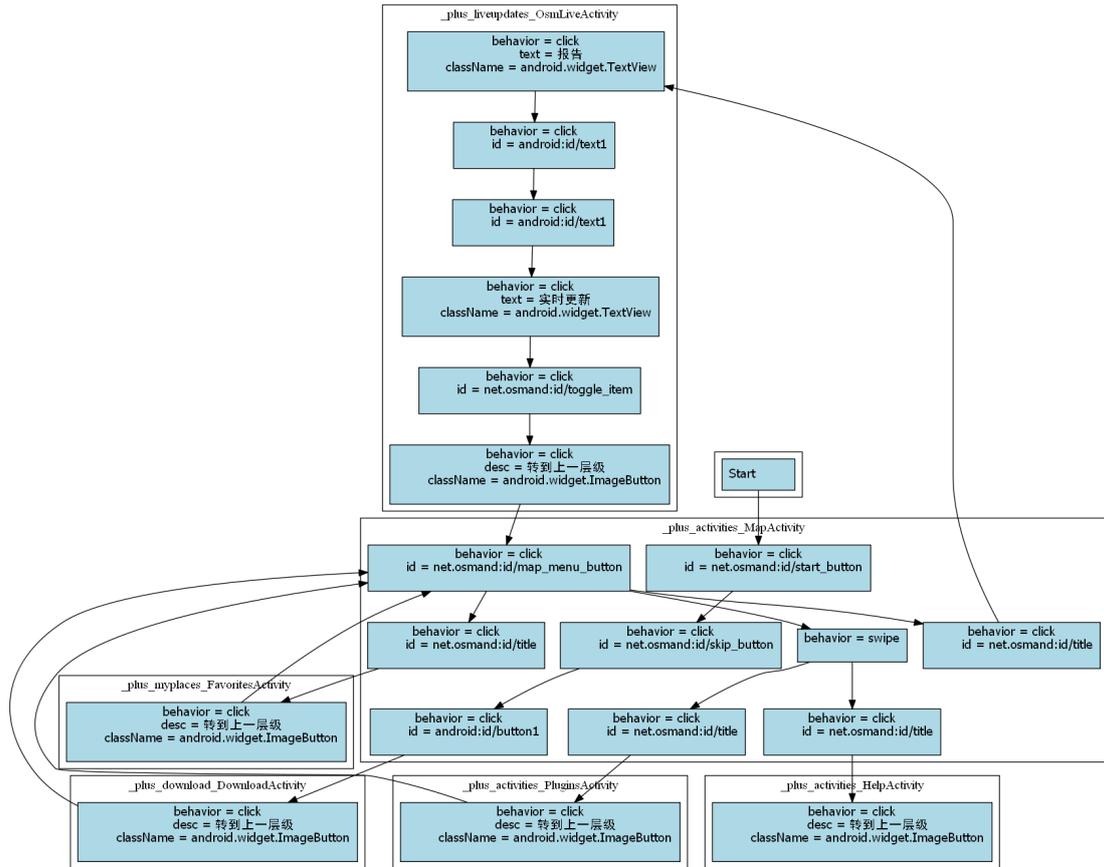


图 5.6: 控件信息页面

5.3.2 实验结果

对于开源代码而言，实验结果以代码覆盖率的方式给出。在我们随机挑选的 10 款应用中，AdGuard、Matomo Mobile 2 以及 OsmAnd Mobile 并不是开源应用，因此无法获取到覆盖率。对于这部分应用，我们通过使用程序在工具运行时循环查询当前 Activity 来记录工具在遍历过程中覆盖的 Activity 种类，并与通过

使用安卓自带的 Develop 工具包获得的应用的总 Activity 数进行比较，得出。最终的实验结果如表5.9所示。

表 5.9: 实验结果

| 编号 | 页面总数 | Monkey | | 简单遍历 | | 迭代式自动化测试系统 | | |
|---------|------|--------|--------|--------|--------|------------|------------------|------------------|
| | | AC | CC | AC | CC | SC | AC(v.s. highest) | CC(v.s. highest) |
| A5 | 4 | 50.00% | / | 50.00% | / | 50.00% | 50.00%(+0%) | / |
| A2 | 5 | 60.00% | 42.91% | 60.00% | 38.74% | 80.00% | 80.00%(+0%) | 62.02%(+19.11%) |
| A10 | 10 | 70.00% | 28.55% | 30.00% | 7.32% | 80.00% | 90.00%(+10%) | 37.60%(+9.05%) |
| A6 | 11 | 27.27% | 40.01% | 27.27% | 28.52% | 36.36% | 36.36%(+0%) | 36.24%(-3.77%) |
| A3 | 14 | 35.71% | 23.59% | 35.71% | 35.34% | 78.57% | 78.57%(+0%) | 52.23%(+16.89%) |
| A8 | 23 | 43.47% | 14.91% | 30.43% | 10.22% | 34.78% | 60.86%(+17.39%) | 16.92%(+2.01%) |
| A1 | 32 | 28.12% | 42.27% | 37.5% | 42.00% | 31.25% | 43.75%(+6.25%) | 48.19%(+5.92%) |
| A4 | 40 | 12.50% | 10.09% | 2.5% | 4.82% | 12.50% | 12.50%(+0%) | 11.59%(+1.5%) |
| A7 | 41 | 17.07% | / | 2.43% | / | 7.3% | 24.39%(+7.32%) | / |
| A9 | 53 | 43.47% | / | 1.88% | / | 20.75% | 24.52%(-18.95%) | / |
| Average | | 38.76% | 28.90% | 27.77% | 23.85% | 43.15% | 50.07%(+6.92%) | 37.83%(+8.93%) |

该实验结果中，页面总数代表该应用所包含的应用页面 Activity 的总数，AC 表示 Activity Coverage，即工具在测试过程中所覆盖的页面占总页面数量的百分比。CC 表示 Code Coverage，即工具在测试过程中覆盖代码的数量占总代码数量的百分比。最后的 SC 表示 Script Coverage, 表示引入到系统中的人工脚本，所覆盖的 Activity 的平均占比。

根据实验结果，我们对本章开始提出的三个问题进行一一解答：

问题一：本系统将用户操作流进行融合后，其中的操作信息是否会有所丢失？

回答：该问题的答案可以见实验结果中，迭代式自动化测试系统的 SC 行与 AC 行，AC 行的数值完全大于 SC 行的数值，首先可以表示的是，自动化测试的覆盖率完全大于原本脚本的覆盖率。同时，我们对两种覆盖种的具体的 Activity 信息以及控件信息也进行了对比。发现自动化测试覆盖的控件集合中，完全包含了用户操作流中覆盖的控件内容，这说明用户操作流的所有信息都不会有所丢失。

问题二：本系统的自动化测试效果在输入人工操作流后，对比未输入时是否有所提升？

回答：该问题的答案，可见实验结果中，迭代式自动化测试系统与简单遍历的 AC 行对比与 CC 行对比。可知本系统的 AC 行结果与 CC 行结果均大于未接入用户操作流时的简单遍历结果，即引入用户操作流之后，测试结果均得到了

极大的提升。由实验结果可知，接入后的页面覆盖率至少能够保持持平，而代码覆盖率全部得到了提升，最大提升值达到了 30.28%。同时，根据遍历过程中记录下的控件信息进行比对，简单遍历过程中覆盖的控件，在引入人工操作流后，依旧没有丢失。

问题三：相较于其他测试工具，本系统的自动化测试工具结合用户操作流之后，能否赶超其覆盖率？

回答：本文所对比的 Monkey 工具，其覆盖率为目前自动化测试工具中的翘楚，根据我们对多篇相关论文的研究，新发表的学术工具，其覆盖率即使能超过 Monkey，也不会超出太多（10% 以内）。并且，该工具由于是谷歌官方工具的原因，配置和使用是最简单的。因此，我们选用该工具作为对比。从实验结果中我们可以发现，在大部分情况下，引入用户操作流后，本系统的覆盖率超过了 Monkey，并且有多款应用超出了 10%，这显然说明了在引入用户操作流后，本系统越过了一部分必须要拥有人类知识才能跨越的界面。但是，同样也有少部分应用没有超越 Monkey，在复现测试并调查原因时，我们发现原因是系统所使用的简单遍历算法，其流程中仍然缺乏一些复杂的测试逻辑，导致在某些页面会因为一些逻辑的不足提前退出。这也是今后我们需要加强的一个方向，即加强系统基础遍历逻辑，提高基底覆盖率，从而使得最终覆盖率也得到提升。

5.3.3 实验结果分析

针对本实验进行的过程，本文作者也发现了许多意想不到的细节。在本节中，会对这些实验过程中出现的现象一一进行分析。

(1) 人工操作与自动化测试逻辑具有互补性

我们发现，实验中引入的自动化测试路径，都比较关注于直接进入新的页面，除非页面上出现了一些关键的功能，否则脚本路径都会倾向于直接点击能够进入新页面的控件。举例来说，引入的人类操作在进入主界面时，会首先点击导航栏，接着点击各个应用子页面，但是部分难以发现的控件会被遗漏。而自动化测试工具，则会在进入一个子页面后，将页面内的所有控件全部覆盖之后，在前往下一个子页面，不会产生控件上的遗漏。

举例来说，在维基百科等多个需要注册的应用中，登陆页面有一个控件会进入注册页面，但是，测试人员都没有点进该页面，并进行后续页面的操作。而自动化测试工具显然是发现了该控件的，因而能够进入到新的控件中，扩张自己的覆盖范围。

但是，自动化测试逻辑也会因为自身的不足产生一些问题。比如点击完一次导航栏后，DFS 策略会认为该控件已经被遍历，便不再去点击第二次，从而

遗漏掉后续的子页面。但是将用户操作流引入之后，测试策略会了解到到达后续页面的方法，那么在下一轮测试时直接复现该方法，到达新的路径即可。

同样的情况也出现在一些开关控件上。如 Jamendo 中的音乐播放顺序控件，通过多次点击会将音乐播放的顺序从单曲循环转到随机播放、再到列表循环等等。但是在自动化测试逻辑眼中，该控件无论再怎么修改，其属性值是一样的，因此只会点击一次。而用户操作流中则会对该控件进行多次点击，在进行复现时，工具就能够学习到这样的操作，从而增强自身的代码覆盖率。

因此，我们认为人工操作与自动化测试逻辑拥有互补性。这证明了本文开头的想法：人工操作能够引入应用测试的深度，而自动化测试注重于测试的广度，将两者结合，确实能够提高测试的各方面结果。

(2) 用户操作的引入能够让工具覆盖到应用更多的部分

通过对实验记录下引入用户操作流前后自动化测试逻辑遍历的控件信息，以及用户操作流中遍历的控件信息，我们发现，初始状态下的控件覆盖数加上用户操作流的控件覆盖数，几乎都小于引入用户操作流后的自动化测试逻辑遍历的控件数量。这意味着在引入用户操作流后，自动化测试逻辑确实有很大的可能性来到新的页面，或覆盖到更多前两者都没有覆盖到的控件。

举例来说，OpenHub 的设置页面中，可以通向日志页面，但是转移页面的控件需要经历向下的滑动操作才能到达。用户操作流中，虽然有向下滑动的操作，但是由于该控件没有被测试人员重视，因而忽略了这条路径。而自动化测试逻辑则由于仍未加入滑动逻辑，而无法到达。在引入用户操作流后，自动化测试逻辑发现了该控件，从而进入了日志页面，到达了新的 Activity，从而获取了更多的覆盖率。这个结果与本文在讲述设计思路时的想法，也是完全一致的。

(3) 简单遍历逻辑在一部分应用中覆盖率低下的原因

我们可以发现，在一些待测应用中，直接使用简单遍历逻辑仅能获得个位数的覆盖率，而引入用户操作流后就能够到达较高的覆盖率。经过一次重现测试过程后，我们发现，这是由于简单遍历逻辑无法成功越过引导页面导致。而引入用户操作流后，其中包含了正确越过引导页面的方式，因此系统后续能够进行应用功能页面的遍历。

如果将越过引导页面的逻辑加上呢？比如说，让测试逻辑在每次测试开始时刻意的左右滑动多次，或只点击右下角的控件，从而越过引导页面。这种想法可以越过大部分的引导页面，但是对实验对象中的 Osmand 而言，它的引导页面只能通过点击画面最中央的“使用入门”控件来越过。并且，即使解决了引导页

面问题，工具也有可能会遇到在某一个页面中没有任何返回上一个页面的按键，而系统的返回按键会直接导致退出应用的问题。这个问题也启发本系统逻辑设置为迭代式测试逻辑的原因。除此之外，由于安卓应用的页面千奇百怪，如果不能做到一一对应，依旧会有层出不穷的原因导致测试出错。由于并不存在一个将这些问题全部解决的自动化测试工具，甚至想要发现全部可能出现的问题都是一个困难的课题。因此，我们认为将测试人员的行为轨迹引入到测试工具中，是一个很好的，规避解决这些问题的方法。

(4) 对于覆盖率数值的观察分析

前文已经提及，越高的覆盖率越有可能发现应用的 BUG，覆盖率已经成为了近年来各大测试工具的一个统一评判标准。从实验结果中，我们不难发现，即使本工具的两种覆盖率在数值上超过了 Monkey，但是整体而言，其数值依旧徘徊在 50% 左右。在查阅了多篇论文后，我们发现，即使是近期发表的自动化测试工具，它们对于安卓应用的覆盖率依旧不是很高。通常来说，这些论文会将自己的工具与一些往期的工具加上 Monkey（谷歌官方工具，所以成为了业界基准）进行比较，然后发表自己工具的覆盖率超越了往期工具覆盖率的结果。但是，应用的总体覆盖率依旧不高，Sapienz 工具的论文实验的平均覆盖率为 53%，FSMDroid 工具的论文实验的平均覆盖率为 45%，EHBDroid 工具的论文实验的平均覆盖率为 43%，而第二章中所述的更加久远的自动化测试工具，如 A3E、puma、Dynodriod 等工具，根据一篇自动化测试工具综述 [41] 的测验结果重现来看，他们的平均覆盖率甚至达不到 40%。

诚然，实验中的待测应用组是一个关键的影响点，有一些应用即使是人类来进行测试也难以覆盖全部的页面，而不同论文中给出的覆盖率粒度也不一样，一些论文实验会以方法覆盖率作为基准，而另一些则会以页面覆盖率或是代码行覆盖率。但是这都无法成为平均覆盖率仍然保持在 50% 左右的理由。

在对实验过程进行复现与观察之后，我们发现了如下几个原因，可以作为工具的最终平均覆盖率结果仍然不高的参考原因：其一，是应用内置网页造成了测试逻辑的混乱。一部分应用，如 wikipedia，monkey 等，都将网页内置在了应用中。虽然有一些方法可以区分内置网页和内置元素，但是这些方法都并不精准。如 Appium 本身提供了查询页面属于网页、应用还是混合型页面。即使能够判断页面属于混合型，但是当功能按键和网页全部显示在同一页面中时，工具依旧难以判断哪些控件应该被舍弃，从而在网页中进行无限的探索，浪费大量的时间于无意义的点击中，或因为网页元素过多（普通的应用页面最多出现几十个控件，而网页通常会有上百个）而导致栈溢出。同样的情况也出现在新闻

应用的列表中，由于新闻可以通过滑动无限加载，而这部分控件内容是属于应用动态生成的，就会导致工具对大量只有 text 属性不同的同类型自动生成控件进行无意义的长时间点击。

其二，是一部分代码本身就属于报错才会触发的，如 try-catch 中 catch 里的后续处理处理中调用相关的处理函数。正常情况下这部分代码是一定不会被覆盖的，但是在计算覆盖率时，这部分处理函数仍旧会计算进去。

其三，是目前的工具在人类不介入干涉情况下仍然不属于完全成熟的范畴，有一部分页面总是会由于工具本身的缺陷而无法覆盖。

但是无论如何，即使受到待测应用不同导致的结果影响，目前工具的平均覆盖率确实就处于 40% 至 50% 左右。在本组实验过程中，Monkey 更是只能达到约 29% 的平均覆盖率（这并不代表 Monkey 弱，只是由于本组实验挑选的应用较难测试，在上文所述的其他工具的对比实验中，Monkey 的平均覆盖率仅比那些工具低上几个百分点），而本系统仅通过融入了多份测试人员可以在 2 到 3 分钟内完成的操作，就令平均覆盖率超越了 Monkey，这无疑能够说明本系统方法的有效性。同时，根据本系统输出的工具覆盖信息，测试人员可以结合源码了解到工具仍未覆盖到的页面，接着再去进行的操作会对未覆盖的部分更有针对性，而如果将这些操作再次引入回系统中后，必然能够使覆盖率再次上升。

5.4 本章小结

在本章中，本文为本系统进行了系统功能的检验，并通过一场实验来证明本系统自动化测试方法的可用性。根据测试结果，本系统通过了所有的功能测试，这说明了本系统的可用性。同时，本章给出了系统的具体展示，说明了系统的实际页面情况。而根据实验结果，本系统在引入用户操作流后，覆盖率得到了极大的提升，且没有丢失以往的控件覆盖，完全符合了系统最开始定下的需求。最后，本章给出了实验过程中发现的各种现象，并为这些现象从各种角度一一进行了解释，探究了这些现象发生的原因，并给出它们带来的启示。

第六章 总结与展望

6.1 总结

随着移动设备的不断普及，手机网民数量的越发增长，移动应用开发与修改的周期也会因此变的越来越短，而自动化测试在这种场景下的效率远远高于人工测试，必然会成为未来移动测试中的主流方法。但是目前自动化测试的两个发展方向：自动化测试框架和自动化测试工具在使用难度和效果上各有优劣。为了取长补短，即能够复用自动化测试框架在执行自动化脚本中产生的用户操作信息（可以推广至从任意方式获取的用户操作），又能够运用自动化测试工具进行便捷的测试，本文设计与实现了迭代式安卓自动化测试系统。

本文首先介绍了本系统的项目背景，以及国内外类似研究的研究现状，接着介绍了与本系统相关的各类知识，包括自动化测试的相关知识以及安卓自动化测试与普通自动化测试的差异、连接与控制移动设备所使用的底层框架 ADB 与 UiAutomator、控制自动化测试的框架 Appium、国内外较为有名的自动化测试工具以及其分类，还有用于在实验中获取应用覆盖率的测试框架 Jacoco。

接着，本文在第三章对系统功能进行了分析，以模块化的方式将本系统划分为多个模块，并为每一个模块设计出的功能提出需求。通过 UML 用例图、用例描述，以实例说明的方式将用例与需求一一对应，并通过 UML 部署图、框架图的方式展示了本系统的物理部署情况以及具体框架，接着通过流程图的方式，说明了各模块的执行流程。在本章的最后，本文介绍了本系统的核心——用户操作流，一种描述用户在待测应用中的控件操作顺序的文件。它可以帮助测试工具了解待测应用的部分框架，扩大测试中能够探索到的场景范围。在第四章中，本文通过 UML 时序图、类图、类表、部分功能性代码的方式说明了系统核心模块的详细设计方式以及具体实现。

最后，本文在第五章中为需求中所阐述的用例设计了功能测试，并设计了一场实验以证明本系统算法的可用性。测试结果和实验结果均表明，本系统的可用性达到了预期。

6.2 展望

迭代式安卓自动化测试系统目前已经完成了初步开发需求，核心功能的用户操作流建模功能，以及自动化测试中读取操作流功能均已得到完成，且通过

了实验证明。但是，在测试与使用的过程中，还存在以下几个问题，有待于后续进行改善：

(1) 本文所选用的自动化测试算法，其流程非常简单，与学术界已经发表的各种工具是无法相比的，这也是实验结果中，简单遍历的结果较低的原因。本系统目前对相关的算法接入部分封装好了接口函数，后续会尝试将不同的成熟测试算法接入到本系统中，同时也会为目前所使用的简单遍历算法进行更多的增强，以增进待测应用的覆盖率。

(2) 本文所选用的用户操作流，目前的来源仅包含执行自动化脚本自动生成，以及使用 LIRAT 工具生成这两种方式。对于其他测试流程，仍需要实现本系统预设好的接口，才能获得到与本系统所需用户操作流格式相一致的文件。在今后的改进中，本文作者会在这两种方式的基础上，为一些其他的热门工具加上用户操作流的获取，使本系统的输入文件的获取方式能够更加简易。

参考文献

- [1] 中国互联网协会, 中国互联网发展报告 (2019) , <https://news.znds.com/article/39223.html>.
- [2] 互联网数据中心 IDC, 2019 年安卓手机占 87% 市场份额, <https://baijiahao.baidu.com/s?id=1644272367710328052>.
- [3] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, D. Lo, Understanding the test automation culture of app developers, in: 8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015, IEEE Computer Society, 2015, pp. 1–10.
URL <https://doi.org/10.1109/ICST.2015.7102609>
- [4] H. Wang, H. Li, L. Li, Y. Guo, G. Xu, Why are android apps removed from google play?: a large-scale empirical study, in: A. Zaidman, Y. Kamei, E. Hill (Eds.), Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, ACM, 2018, pp. 231–242.
URL <https://doi.org/10.1145/3196398.3196412>
- [5] M. Fazzini, Automated support for mobile application testing and maintenance, in: G. T. Leavens, A. Garcia, C. S. Pasareanu (Eds.), Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018, ACM, 2018, pp. 932–935.
URL <https://doi.org/10.1145/3236024.3275425>
- [6] X. Zhang, Z. Chen, C. Fang, Z. Liu, Guiding the crowds for android testing, in: L. K. Dillon, W. Visser, L. Williams (Eds.), Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume, ACM, 2016, pp. 752–753.
URL <https://doi.org/10.1145/2889160.2892659>
- [7] S. Shen, H. Lian, T. He, Z. Chen, Clustering on the stream of crowdsourced testing, in: 14th Web Information Systems and Applications Conference, WISA 2017,

- Liuzhou, Guangxi Province, China, November 11-12, 2017, IEEE, 2017, pp. 317–322.
URL <https://doi.org/10.1109/WISA.2017.47>
- [8] M. L. Vásquez, C. Bernal-Cárdenas, K. Moran, D. Poshyvanyk, How do developers test android applications?, CoRR abs/1801.06268.
URL <http://arxiv.org/abs/1801.06268>
- [9] 陈镜, 姬庆, 夏夜, 移动互联自动化测试框架创新研究与实践, 金融电子化 (5) (2016) 69–70.
- [10] G. Shah, P. Shah, R. Muchhala, Software testing automation using appium, International Journal of Current Engineering and Technology 4 (5) (2014) 3528–3531.
- [11] 古锐, 肖璞, 基于 appium 的 android 应用自动化测试框架的研究, 现代计算机: 专业版 629 (29) 97–102.
- [12] H. Zadgaonkar, Robotium Automated Testing for Android, Packt Publishing Birmingham, 2013.
- [13] 祝阳阳, 侯永宏, 王宝亮, Android 自动化测试工具 robotium 的应用与研究, 信息技术 (10) 206–208+213.
- [14] Google, Monkey: a command-line tool that generates pseudo-random streams, <https://developer.android.com/studio/test/monkey.html>.
- [15] A. Machiry, R. Tahiliani, M. Naik, Dynodroid: an input generation system for android apps, in: B. Meyer, L. Baresi, M. Mezini (Eds.), Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE’13, Saint Petersburg, Russian Federation, August 18-26, 2013, ACM, 2013, pp. 224–234.
URL <https://doi.org/10.1145/2491411.2491450>
- [16] K. Mao, M. Harman, Y. Jia, Sapienz: multi-objective automated testing for android applications, in: A. Zeller, A. Roychoudhury (Eds.), Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016, ACM, 2016, pp. 94–105.
URL <https://doi.org/10.1145/2931037.2931054>

-
- [17] S. Hao, B. Liu, S. Nath, W. G. J. Halfond, R. Govindan, PUMA: programmable ui-automation for large-scale dynamic analysis of mobile apps, in: A. T. Campbell, D. Kotz, L. P. Cox, Z. M. Mao (Eds.), The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014, ACM, 2014, pp. 204–217.
URL <https://doi.org/10.1145/2594368.2594390>
- [18] R. Mahmood, N. Mirzaei, S. Malek, Evodroid: segmented evolutionary testing of android apps, in: S. Cheung, A. Orso, M. D. Storey (Eds.), Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014, ACM, 2014, pp. 599–609.
URL <https://doi.org/10.1145/2635868.2635896>
- [19] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, Z. Su, Guided, stochastic model-based GUI testing of android apps, in: E. Bodden, W. Schäfer, A. van Deursen, A. Zisman (Eds.), Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017, ACM, 2017, pp. 245–256.
URL <https://doi.org/10.1145/3106237.3106298>
- [20] T. Azim, I. Neamtiu, Targeted and depth-first exploration for systematic testing of android apps, in: A. L. Hosking, P. T. Eugster, C. V. Lopes (Eds.), Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013, ACM, 2013, pp. 641–660.
URL <https://doi.org/10.1145/2509136.2509549>
- [21] N. Mirzaei, J. Garcia, H. Bagheri, A. Sadeghi, S. Malek, Reducing combinatorics in GUI testing of android applications, in: L. K. Dillon, W. Visser, L. Williams (Eds.), Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, ACM, 2016, pp. 559–570.
URL <https://doi.org/10.1145/2884781.2884853>
- [22] Y. Hu, I. Neamtiu, A. Alavi, Automatically verifying and reproducing event-based races in android apps, in: A. Zeller, A. Roychoudhury (Eds.), Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016,

- Saarbrücken, Germany, July 18-20, 2016, ACM, 2016, pp. 377–388.
URL <https://doi.org/10.1145/2931037.2931069>
- [23] L. Clapp, O. Bastani, S. Anand, A. Aiken, Minimizing GUI event traces, in: T. Zimmermann, J. Cleland-Huang, Z. Su (Eds.), Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, ACM, 2016, pp. 422–434.
URL <https://doi.org/10.1145/2950290.2950342>
- [24] X. Zeng, D. Li, W. Zheng, F. Xia, Y. Deng, W. Lam, W. Yang, T. Xie, Automated test input generation for android: are we really there yet in an industrial case?, in: T. Zimmermann, J. Cleland-Huang, Z. Su (Eds.), Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, ACM, 2016, pp. 987–992.
URL <https://doi.org/10.1145/2950290.2983958>
- [25] 戴汝为, “人机结合”的大成智慧, 北方工业大学学报 (3) (1996) 1–8.
- [26] 钟芳凌, 综合显控集成框架软件的人机结合自动化测试, 电子测试 (16) (2019) 32–33.
- [27] 黄延胜, 一款基于自动遍历的 app 爬虫工具, <https://github.com/seveniruby/AppCrawler>.
- [28] X. Li, Y. Jiang, Y. Liu, C. Xu, X. Ma, J. Lu, User guided automation for testing mobile apps, in: S. S. Cha, Y. Guéhéneuc, G. Kwon (Eds.), 21st Asia-Pacific Software Engineering Conference, APSEC 2014, Jeju, South Korea, December 1-4, 2014. Volume 1: Research Papers, IEEE Computer Society, 2014, pp. 27–34.
URL <https://doi.org/10.1109/APSEC.2014.13>
- [29] L. Gomez, I. Neamtiu, T. Azim, T. D. Millstein, RERAN: timing- and touch-sensitive record and replay for android, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 72–81.
URL <https://doi.org/10.1109/ICSE.2013.6606553>

-
- [30] K. Mao, M. Harman, Y. Jia, Crowd intelligence enhances automated mobile testing, in: G. Rosu, M. D. Penta, T. N. Nguyen (Eds.), Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017, IEEE Computer Society, 2017, pp. 16–26.
URL <https://doi.org/10.1109/ASE.2017.8115614>
- [31] Z. Dong, M. Böhme, L. Cojocaru, A. Roychoudhury, Time-travel testing of android apps, in: Proceedings of the 42nd International Conference on Software Engineering, ICSE '20, 2020, pp. 1–12.
- [32] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, Y. L. Traon, Static analysis of android apps: A systematic literature review, *Inf. Softw. Technol.* 88 (2017) 67–95.
URL <https://doi.org/10.1016/j.infsof.2017.04.001>
- [33] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, J. Klein, Automated testing of android apps: A systematic literature review, *IEEE Trans. Reliability* 68 (1) (2019) 45–66.
URL <https://doi.org/10.1109/TR.2018.2865733>
- [34] A. Méndez-Porras, C. Quesada-López, M. Jenkins, Automated testing of mobile applications: A systematic map and review, in: J. Araújo, N. Condori-Fernández, M. Goulão, S. Matalonga, N. Bencomo, T. C. Oliveira, J. L. de la Vara, I. S. Brito, L. Antonelli, E. Pimentel, J. J. Miranda, M. Kalinowski, O. Pastor, L. Olsina, R. S. S. Guizzardi, S. España, E. Cuadros-Vargas (Eds.), Proceedings of the XVIII IberoAmerican Conference on Software Engineering, CIbSE 2015, Lima, Peru, Apr 22-24, 2015, Curran Associates, Inc., 2015, p. 195.
- [35] M. Hoffmann, B. Janiczak, E. Mandrikov, M. Friedenhagen, Jacoco code coverage tool, <https://www.eclemma.org/jacoco/>.
- [36] J. A. Whittaker, What is software testing? why is it so hard? practice tutorial, *IEEE Software* 17 (1) (2000) 70–79.
URL <https://doi.org/10.1109/52.819971>
- [37] R. P. Testardi, Methods and systems for automated software testing, uS Patent 6,249,882 (Jun. 19 2001).

- [38] J. Park, Y. B. Park, H. K. Ham, Fragmentation problem in android, in: International Conference on Information Science and Applications, ICISA 2013, Hilton Pattaya Hotel, Pattaya, Thailand, June 24-26, 2013, IEEE, 2013, pp. 1–2.
URL <http://doi.ieeecomputersociety.org/10.1109/ICISA.2013.6579465>
- [39] L. Wei, Y. Liu, S. Cheung, Taming android fragmentation: characterizing and detecting compatibility issues for android apps, in: D. Lo, S. Apel, S. Khurshid (Eds.), Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016, ACM, 2016, pp. 226–237.
URL <https://doi.org/10.1145/2970276.2970312>
- [40] L. Nguyen-Vu, J. Ahn, S. Jung, Android fragmentation in malware detection, *Comput. Secur.* 87.
URL <https://doi.org/10.1016/j.cose.2019.101573>
- [41] S. R. Choudhary, A. Gorla, A. Orso, Automated test input generation for android: Are we there yet? (E), in: M. B. Cohen, L. Grunske, M. Whalen (Eds.), 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015, IEEE Computer Society, 2015, pp. 429–440.
URL <https://doi.org/10.1109/ASE.2015.89>
- [42] A. Mesbah, A. van Deursen, S. Lenseslink, Crawling ajax-based web applications through dynamic analysis of user interface state changes, *TWEB* 6 (1) (2012) 3:1–3:30.
URL <https://doi.org/10.1145/2109205.2109208>
- [43] S. Yu, C. Fang, Y. Feng, W. Zhao, Z. Chen, LIRAT: layout and image recognition driving automated mobile testing of cross-platform, in: 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019, IEEE, 2019, pp. 1066–1069.
URL <https://doi.org/10.1109/ASE.2019.00103>

简历与科研成果

基本情况 徐悠然，男，汉族，1995年8月出生，江苏省镇江市人。

教育背景

2018.9 ~ 2020.6 南京大学软件学院 硕士

2014.9 ~ 2018.7 苏州大学计算机科学与技术学院 学士

科研成果

1. 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
2. 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
3. 陈振宇，田元汉，房春荣，**徐悠然**，张欣，“一种基于人工辅助的 Android 半自动化测试方法”，申请号：201810325056.2。
4. 陈振宇，张天，房春荣，田元汉，**徐悠然**，“一种基于混合云的可扩展的移动应用自动化测试框架”，申请号：201810336477.5。
5. 房春荣，**徐悠然**，田元汉，陈振宇，李玉莹，“一种移动应用人机测试对战的框架及评价方法”，申请号：201810093017.4。
6. 房春荣，**徐悠然**，钟怡，田元汉，刘佳玮，“一种面向移动应用的一体化测试效率评估方法”，申请号：2019109518894。
7. 房春荣，**徐悠然**，田元汉，韦志宾，陈振宇，“一种结合群体智能与机器智能增强移动应用测试的方法”，申请号：201910952056X。

致 谢

迭代式自动化测试系统从最初的产生一个初步的想法，到现在的实现完成，整个过程经历了一年多的时间。在整个研究过程中，系统的实现方式出现了不少的问题，具体的实现方法也在老师和同学的帮助下多次改正，最终实现了现在较好的结果。在此，我要为支持和帮助过我的老师、同学、亲友致以最诚挚的感谢。

首先，我要感谢我的指导老师陈振宇教授以及房春荣老师，从系统的设计、研究的思路，到实验的设计中，他们都帮助并给与了我许多指导与指正。感谢陈老师能够同意让我加入 ISE 实验室，让我能够在具有优秀研发氛围的环境下不断提升自己，也感谢房老师一直以来对我学术上的指导，激励着我在研究生生活中的学习与工作。

其次，我要感谢实验室中的黄勇老师以及同样是我们移动测试小组中的田元汉学长、李灏宇学长以及韦志宾同学。黄老师为这个项目提出了许多建设性的需求意见，其他学长与同学的项目与我的项目共同组合成了移动应用测试平台，感谢他们平日里对于我遇到的问题的不吝赐教。

我也要感谢我的父母和亲友在研究生生涯中对我的支持。在撰写论文的这段疫情期间，他们一直对我鼓励，给与了我前进的动力。感谢这两年来所有关心过我，给予我帮助的老师、同学以及亲友。

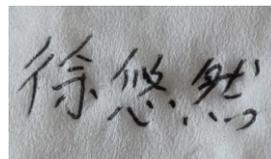
最后，感谢能够在百忙之中审阅论文以及参与答辩的各位评审专家，向你们致敬！

附件二

《学位论文出版授权书》

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

作者签名: _____



_____ 2020 年 5 月 23 日

| | | | | | |
|----------|--|------|------|------|------|
| 论文题名 | 迭代式安卓应用自动化测试系统的设计与实现 | | | | |
| 研究生学号 | MF1832202 | 所在院系 | 软件学院 | 学位年度 | 2020 |
| 论文级别 | <input type="checkbox"/> 硕士 <input checked="" type="checkbox"/> 硕士专业学位 <input type="checkbox"/> 博士 <input type="checkbox"/> 博士专业学位 (请在方框内画钩) | | | | |
| 作者 Email | 362029695@qq.com | | | | |
| 导师姓名 | 陈振宇 房春荣 | | | | |

论文涉密情况:

不保密

保密, 保密期(_____年_____月_____日至_____年_____月_____日)

注: 请将该授权书填写后装订在学位论文最后一页(南大封面)。