

硕士学位论文

(工程硕士)

Android 平台的自动化测试系统的实现与
优化

**THE IMPLEMENTATION AND OPITMIZATION
OF AUTOMATED TESTING SYSTEM FOR THE
ANDROID PLATFORM**

张兆鹏

哈尔滨工业大学

2013 年 6 月

国内图书分类号：TP311
国际图书分类号：621.3

学校代码：10213
密级：公开

工程硕士学位论文

Android 平台的自动化测试系统的实现与 优化

硕 士 研 究 生：张兆鹏

导 师：黄虎杰教授

副 导 师 康克军高级软件工程师

申 请 学 位：工程硕士

学 科：软件工程

所 在 单 位：软件学院

答 辩 日 期：2013 年 6 月

授予学位单位：哈尔滨工业大学

Classified Index: TP311

U.D.C: 621.3

Dissertation for the Master Degree in Engineering

**THE IMPLEMENTATION AND OPITMIZATION
OF GUI AUTOMATED TESTING SYSTEM FOR
THE ANDROID PLATFORM**

Candidate:	Zhang Zhaopeng
Supervisor:	Prof. Huang Hujie
Associate Supervisor:	Senior Software Engineer Kang Kejun
Academic Degree Applied for:	Master of Engineering
Speciality:	Software Engineering
Affiliation:	School of Software
Date of Defence:	June, 2013
Degree-Conferring-Institution:	Harbin Institute of Technology

摘 要

课题来源于东软集团股份有限公司嵌入式事业部进行的针对 Android 系统的某移动终端设备自动化测试框架研发项目。该项目与美国某知名平板电脑品牌公司合作完成。主要内容是设计一个自动化测试系统来完成对该公司开发的 Android 设备进行测试。整个项目包括用来控制测试用例运行的 Android 平台自动化测试系统与用来管理和组织大量测试用例的自动化管理框架两部分。本系统为其中的 Android 平台自动化测试系统。它作为测试框架和测试用例之间的中间层，主要是为了驱动测试用例运行，提高测试用例运行的稳定性，并提高测试用例的统一性及可复用性以及规范化输出测试结果而设计的。

本文研究的自动化测试系统为充分实现预期目标，通过 Android SDK 提供的 ADB (Android 调试桥) 工具与 Python 脚本技术相结合，并借助调用如 Quadant, GLBenchmark, NenaMark 等多种第三方性能测试工具来实现对 Android 设备的功能和性能进行全面测试。在完成基本自动化测试功能之余通过对 ADB 命令序列的封装，为测试开发人员直接提供操作接口。根据设计，将自动化测试系统分为五个功能模块：测试用例加载模块、测试状态运行监控模块、ADB 接口模块、常用操作封装模块、测试结果输出模块。通过各模块功能的紧密联系，完成了自动化测试的前期准备和后续工作，最终实现了使自动测试框架能够更加稳定高效的运行，提高测试用例的稳定性和可移植性、减少测试用例的运行的时间、提高测试过程中对被测设备有效 Bug 的检出，以及对设备发生异常时及时进行的处理功能。本文的主要内容也是围绕各个模块的设计方案和实现方案进行逐一阐述。最后对系统的每个功能点进行了测试，验证了本次课题满足了实际需求。

本系统已经在公司内部投入使用，大大提高了测试工作的效率并很大程度上节省手工劳动同时满足产品的测试要求，完全满足了系统的需求。

关键词：Android 系统；自动化测试系统；Android 调试桥； Python 脚本技术

Abstract

This project is based on the research and development of automatic test platform for a mobile terminal running Android system in the on chip department of Neusoft Corporation. This project is co-operated with a famous tablet company in America. The main purpose is to design an automatic platform for Android devices developed by the company mentioned above. The whole project is consisted of two parts: one is the automatic test platform of Android system to control the operation of test subject, and the other is automatic management platform for organization of massive test objects. Our system is the automatic test platform of Android system. As the intermediate layer between test platform and test subject, it should drive the running of test subjects, improve the stability of this running, improve the integrity and repeatability of test subjects, and standardize test results.

In order to meet these expectations, this automatic test platform uses ADB(Android Debug Bridge) provided by Android SDK and Python script, and other third party tools such as Quadant, GLBenchmark and NenaMark are also used to test the performance of Android devices fully. Besides the basic automatic test functions, this system provides the packaging of ADB command sequences and the operational ports for testers. According to the design, this automatic test platform can be divided into five functional parts: loading module for test subjects, monitoring module, ADB interface module, routine operation module and output module for test results. By the tight connections among different modules, the early preparation and follow-up works are done. The stability and high performance of the automatic test platform are achieved. Meanwhile, the stability and the ability for transportation is improved. The running time of test subjects is reduced and the identification of valid Bugs during test process is improved. Also, this system has the ability to handle abnormal situations when devices are malfunctional. This article describes the design and realization of the system based on different modules. Then, every functional point is tested to make sure the system meet the practical uses.

This system has been utilized inside our company. It improves the efficiency of software testing and saves the labor work of manually testing significantly. In all, test requirements for products are fully achieved.

Keywords: Android System, Automatic Testing System, ADB, Python Scripting technologies

目 录

摘 要.....	I
ABSTRACT	II
第 1 章 绪 论	1
1.1 课题背景及研究的目的和意义	1
1.1.1 课题背景	1
1.1.2 研究的目的和意义	1
1.2 与本课题有关的国内外发展概况	2
1.2.1 Android 系统发展概况	2
1.2.2 自动化测试发展概况	3
1.3 与本课题有关的技术概述	6
1.3.1 Android 系统架构	6
1.3.2 Android 应用程序构成	8
1.3.3 自动化测试技术	10
1.3.4 Python 技术语言	11
1.4 本文的主要研究内容	11
1.5 本文组织结构	12
第 2 章 自动化测试系统的需求分析与总体方案	14
2.1 自动化测试系统的需求分析	14
2.1.1 自动化测试系统的功能需求	14
2.1.2 自动化测试系统的非功能需求	16
2.2 系统的实施方案	17
2.2.1 自动化测试系统整体逻辑架构	17
2.2.2 测试用例控制模块	18
2.2.3 测试用例控制模块的业务流程	20
2.3 预期达到的目标	22
2.3.1 功能性目标	22
2.3.2 非功能性目标	22
2.4 本章小结	22
第 3 章 Android 平台的自动化测试系统的设计	23
3.1 系统的总体设计	23

3.2 测试用例加载模块的设计	24
3.3 测试状态运行监控模块的设计	26
3.4 ADB 接口模块设计的设计	28
3.5 常用操作封装模块的设计	30
3.6 测试结果输出模块的设计	32
3.7 测试用例基类模块的设计	34
3.8 本章小结	36
第 4 章 Android 平台的自动化测试系统的实现	37
4.1 测试用例加载模块的实现	37
4.1.1 模拟外部测试框架调用功能的实现	38
4.1.2 创建测试用例模板功能的实现	39
4.2 测试状态运行监控模块的实现	39
4.2.1 ADB 工具运行监控功能的实现	39
4.2.2 待测设备运行监控功能的实现	41
4.3 ADB 接口模块的实现	41
4.4 常用操作封装模块的实现	43
4.5 测试结果输出模块的实现	43
4.6 本章小结	44
第 5 章 自动化测试系统的测试	45
5.1 测试方法	45
5.1.1 测试脚本的选取和执行	45
5.1.2 测试运行的结果分析	47
5.2 其它功能测试	47
5.2.1 ADB 接口模块的测试	48
5.2.2 常用操作封装模块测试	49
5.3 非功能测试	49
5.4 测试结果分析	50
5.5 本章小结	50
结 论	51
参考文献	52
哈尔滨工业大学学位论文原创性声明和使用权限	55
致 谢	56
个人简历	57

第 1 章 绪 论

1.1 课题背景及研究的目的和意义

1.1.1 课题背景

本课题名称为“Android 平台的自动化测试系统的实现与优化”。

课题来源于东软集团股份有限公司嵌入式事业部进行的基于 Android 系统的某移动终端设备研发项目。鉴于公司产品还处于保密阶段，本文中不出现具体的设备名称。“Android 平台自动化测试系统的实现与优化”为其中的一个子项目。一般来讲，测试脚本在自动执行时，经常会出现执行不稳定不完全的问题，并且由于输出的信息中同时包括系统的常规运行 log 和测试用例的运行结果，很难将生成测试报告所需的测试结果提取出来。本课题的自动化测试系统重点研究的是该项目中的测试框架与测试用例之间的测试用例控制部分以及各种系统测试用例的开发与设计部分。测试用例控制部分的作用是为了使测试用例能够高效稳定的运行，并且可以适用于不同终端设备。

1.1.2 研究的目的和意义

对于软件来说，如何保证软件在使用的过程中不发生各种 BUG 是一个难题。这就需要软件在开发的过程中就按照规范的开发流程来执行，并且需要检查软件本身的质量。目前因为形式化的检查方式和程序正确与否的证明还不能在实际工作中应用的情况下，在未来的很长时间里软件测试依然是确保软件质量的重要方式。近年来，虽然我国软件发展速度比较快，但陷入和一个误区，我国的软件公司大多将人力和财力重点放在发展软件的开发技术和算法，但是在软件开发过程尤其是软件测试的探究很少。这也是导致和国际有较大差距的一个根本原因。要想提高在国际软件市场上的竞争力，唯有确保有具有很高的产品质量，所以软件测试是非常重要的。所以研究如何通过软件测试来确保软件质量是一个很有价值的课题。

在移动终端迅猛发展之时，移动终端手机发展的速度更加让人为之惊叹。从最早的 GSM、CDMA 到当下日渐火爆的 3G，和正跃跃欲试的移动通信的第四代，通信技术的发展在某种程度上在手机发展上得到了充分的体现。移

动终端的发展同时使得手机客户端软件这样的移动终端软件得到了发展。目前传统的手机应用软件已经不能再满足人们的需求，手机应用软件的功能也不再是单一的游戏和系统应用。

随着移动互联网的发展，从手机客户端软件设计的开始到最后软件产品的上线，手机客户端软件自动化测试所占的比重也变得越来越来大。传统的测试方法中，手工测试一直占很大的比重。但是手工测试在某些方面仍然存在弊端，例如在测试资源冲突方面精确度有限；或者是需要消耗大量的人力来进行常用的压力测试上，由于以上原因开展自动化测试系统的设计是必要的。

它能够真正的把测试人员从简单反复的测试劳动中解放出来，达到规范测试流程，节省测试时间的目的。Android 系统的开源特性为自动化测试带来了很多方便的地方。

1.2 与本课题有关的国内外发展概况

1.2.1 Android 系统发展概况

Android 是一种基于 Linux 的移动操作系统，它一般应用在平板电脑，智能手机，智能电视等设备上。Android 的源代码是开放的。Android 的创始人是 Andy Rubin，最初开发时是针对智能手机市场，到后来慢慢发展，并应用在平板电脑等多种设备上。它在 2005 年被 Google 收购，但这反而成为 Android 发展得一个契机，Google 在收购 Android 之后对其非常重视，投入很多资金，并将 Android 不断改善，使其越来越被人们认可。在 2011 年，Android 的市场份额便已经超过了塞班，牢牢占据智能手机市场的头把交椅。

截止到 2012 年，智能移动终端一般采用四种操作系统。苹果公司的 IOS 系统是专门为苹果公司自家的产品所服务，而黑莓系统和微软研发的 Windows Phone 系统又没有很好的发展壮大，只有 Google 公司研发的 Android 系统以其独特的开源行和对多种硬件很好的支持迅速获得大量用户及硬件厂商的青睐。据统计，在 2012 年第一季度，Android 的市场占有率已经超过了 68%，稳居智能移动终端市场的第一位^[1]。

Android 是基于 Linux 的，但又不是纯 Linux 操作系统。Android 系统上的应用也是利用 Java 语言编写的，这些应用程序运行在 Android 系统的 Linux 底层的 Dalvik 虚拟机上。根据这个特点我们可以发现，Android 是一种基于 Linux 的操作系统^[2]。

Android 平台具有如下优点：

(1) 开放性

Android 的一个很大的优势便是它的开放性，它的开放可以吸引大量的移动设备厂商加入，可以使越来越多的移动设备搭载 Android 系统。这使 Android 拥有大量的用户，如此可以促成 Android 的良性发展。

(2) 不被制约

Android 系统会被运行商所制约，Android 操作系统可以自由的选择接入何种网络。同时，Android 系统可以使它的用户利用各种方式随时随地的享受上网的快乐。

(3) 庞大的硬件支持

Android 的开放性使其被大量厂商所青睐，这样便使得 Android 能够拥有在各个方面的硬件上前沿的支持，用户可以选择自己需要的硬件类型，不同的硬件并不会对系统的稳定性造成差异性。这对它的性能有了很大的帮助，并且大大增加了 Android 应用的兼容性。

(4) 开发门槛低

Android 平台的开放性不仅仅体现在对厂商，对开发者更是非常开放，只要你对开发 Android 感兴趣，那么你便可以尽情发挥你的创意和能力，为 Android 开发各种应用。这使得 Android 平台的应用数量大大增加，其中优秀的作品也层出不穷。

(5) Google 应用

Android 平台的移动设备可以完美与 Google 应用相结合，如 Google Map、Gmail、Google Search 等多种应用，使开发应用软件时非常方便快捷。开发人员能够直接在其所开发的应用中嵌入 Google 的各种服务，使很多复杂的应用软件在开发时大大减省了难度。

综上所述，由于 Android 平台具有以上优势，使得它很短时间内便拥有了大量的支持者，受到了开发人员和消费者的喜爱，Android 设备被越来越多的消费者所接受。

1.2.2 自动化测试发展概况

自动化测试系统就是使用自动化测试软件来实现系统测试,减少由人力操作来对软件的质量是否过关进行验证的系统，它不但可以直接的进行测试动作，并且可以对测试工作进行监控，分配和跟踪。自动化测试的实现一般是由测试脚本和第三方测试工具来完成，它的主要作用是使测试人员解放开来，避免了大量人力的浪费，并且可以使测试工作的工作效率得到有效的改

善，使软件的开发周期变得更加紧凑。

软件测试贯穿于整个 IT 行业，它对于整个软件开发行业非常重要，它的发展速度并不受某一个具体技术的影响，而是以非常迅速状态在发展。

在 1960 年之前，那时的软件一般不像如今的那样庞大和复杂，在那个时代，软件开发并不是十分规范，软件工程的知识体系也没有形成，人们对软件测试的重视程度也不够。那时的软件测试往往都是由开发软件的小组直接对其开发的软件进行调试。虽然调试的过程与软件测试看起来很相似，但其实这与我们现在成熟的软件测试的体系结构相差甚远。

1960 年至 1970 年这段时间里，虽然软件的结构和规模并没有发展得特别迅速，但是软件的开发已经变得比较规范，人们不仅仅的只是随意的写一些代码来编程，而是越来越重视软件开发过程中每个阶段的规范性，希望通过软件开发流程控制来起到提高开发效率，优化软件质量的效果^[3]。那时，软件开发过程已经有了当代的需求分析、概要设计、详细设计、编码实现以及调试测试这样的一个基本结构。不过人们还是没有对软件测试这部分的工作给予足够的重视，还仅仅是通过对开发出来的软件进行大量的使用，在使用过程中随机的发现一些软件漏洞及错误。软件测试也并没有形成专业的理论体系。不过在 1972 年，在北卡罗来纳州举办的一次会议上，比尔阐述了对软件测试理论的深刻理解，这成为了软件测试发展中的一个节点。从此之后，软件测试便成为了一个具体的学术项目。在 1979 年，格兰富特撰写了一本关于软件测试的书《软件测试的艺术》^[4]。这本书不但介绍和总结了很多关于软件测试的方法，而且第一次发表了软件测试的本质是证明 BUG 的存在而不是证明软件没有问题。这个观点对于软件测试理论来说非常重要，使软件测试的概念更加的明确了。也为今后软件测试的发展奠定了基础。

在 1980 年到 1990 年之间，个人电脑成为了计算机世界的主流产品，由于个人电脑数量的急剧增加，软件的数量和功能也越来越复杂。在这样的背景下，软件测试的重要性也被各大软件公司关注起来^{[5][6]}。越来越多的公司组建了专门的软件测试项目组来完成软件测试的工作，以满足对软件质量的保证。

到了 1995 年以后，由于计算机网络的飞速发展，越来越多的网络软件被开发出来，这些软件结合了因特网的技术，使得软件本身的规模和复杂度越来越高，这也导致软件测试工作的工作量越来越大，测试人员不仅仅是对软件产品的使用，还需要处理很多复杂的技术难点^[7]。并且大量的测试工作使软件测试人员苦不堪言。软件对质量的要求越来越高，由于测试人员的疏忽

导致的软件质量事件层出不穷。所以由于这种强烈的需求，软件开发人员开发出了各种自动化测试工具，自动化测试的概念也随之被提出，它做为软件测试理论的发展和延伸，形成了一个非常具体的理论体系^{[8][9]}。

软件测试是整个软件开发过程中很重要的一部分，传统的瀑布开发模型只在软件维护阶段前期对软件进行测试，以此来保证软件在投入使用前的质量。^[10]但是在最近几年，整个软件行业逐渐将软件测试的重要性更加重视起来了。研究人员认为应该将软件测试融入到软件开发的每一个阶段中，保证软件在开发过程中的每一个阶段都能够符合计划的目标，能够将问题消灭在萌芽之中，防止错误由于没有及时发现而导致问题越来越严重。在开发后期使问题成为无法解决的灾难性错误。

软件测试可以保证软件的质量，加快开发进度，节约开发成本，并防止在开发过程中由于失误导致严重倒退。最近几年，软件的规模越来越大，软件开发流程越来越规范，软件开发技术也飞速发展。所以软件开发的效率也大大的提高了^[11]。由于这些变化，软件测试的工作量也大大增加了。并且由于软件测试越来越受到重视，在软件开发的每个阶段中都加入了测试工作，所以按照传统的人力手工测试已经很难满足测试工作的需要了。针对这些需求，软件的自动化测试系统与自动化测试工具被大量的开发出来，使软件测试发展进入了一个更高级的阶段^[12]。人们普遍应用自动化测试技术来提高软件开发的效率，节约开发成本。

相对来说，中国对于软件测试的研究比较落后，开始发展得时间也比较晚。美国和欧洲在2002年左右便已经对自动化测试理论专门的进行了关注和研究，自动化测试技术在欧美也发展得比较成熟。很多自动化测试工具都在那段时间里如雨后春笋般不断涌现出来，在这里面也涌现除了很多游戏的自动化测试工具。随着自动化测试工具的发展，自动化测试框架也浮出了水面^[13]。

国外有很多专门的机构来对软件测试制定标准，使其更加规范化，如IEEE、ACM等。也有一些公司开发很优秀的测试工具，如Mercury Interactive公司、ATTOLTESTWARE公司和IBM Rational公司，正是由于它们对软件测试的重视及研究使得软件测试的发展非常迅速。

我国软件测试起步较晚，最初是在“六五”期间开始研究的，软件测试技术在我国是伴随着软件工程理论一起发展的。由于国内水平相对落后，所以与国际先进水平有明显差距，并且很少有国内的公司或研究部门针对软件测试技术进行深化研究。不过随着我国对软件质量的重视程度的提高，软件测试也越来越被重视，逐渐在发展壮大^[14]。

上海市计算机软件评测重点实验室^[15]以及中国赛宝实验室软件评测中心这两家机构是我国软件测试的领头机构，他们在专业领域内保持强劲势头，对我国软件测试技术的发展做出了突出贡献^{[16][17]}。

1.3 与本课题有关的技术概述

1.3.1 Android 系统架构

Android 的整体系统架构主要分为四部分，如图 1-1 所示。

由图 1-1 我们可以看到，Android 系统的架构主要分成四个部分，即应用层 (application)，应用框架 (application framework)，库 (libraries)以及 Linux 核心层 (Linux Kernel)^[18]。

Android 的 Java 编程语言的接口功能，有无穷无尽的变化。Activity 类的任务是生成窗口 (window)，同时在 Activity 过程中的活动相当于 前景在前台模式运行的程序，服务 (Service)在后台运行的程序^{[19][20]}。由服务连接 (ServiceConnection)和的 AIDL 来建立两者的联系，以实现复杂程序在同一时间同时运行。在操作中的活动全部画面均由其余的活动取代了，这个 Activity 将被终止，或可能有系统将其清理掉^[21]。

控件 (View)与 J2ME 的 Displayable 等价，开发者能够运用控件类和“XML layout”实现 UI 在窗口中的布局。Android 1.5版能够运用控件创造名称为 Widgets 的视图小工具，事实上视图小工具属于控件，为此在布局策划上能够采用 xml^[22-24]。

ViewGroup 是各种不同布局的抽象类，ViewGroup 内部还可存在有 ViewGroup。在活动中可以不访问控件的构造函数，但是一定要访问 Displayable 的构造函数。在活动中，想从 XML 中获得控件可以使用活动 findViewById 函数来实现^[25]。Android 的控件类大部分是通过从 XML 中获得后展示的。控件与事件 (Event)是紧密联系的，运用监听将两者联系在一起。每一个控件均能够获得一个事件监听，例如：若控件想解决操作者触摸的操作时，需先通过 Android 框架声明 View.OnClickListener^[26]。此外图片等价于 J2ME 的位图。

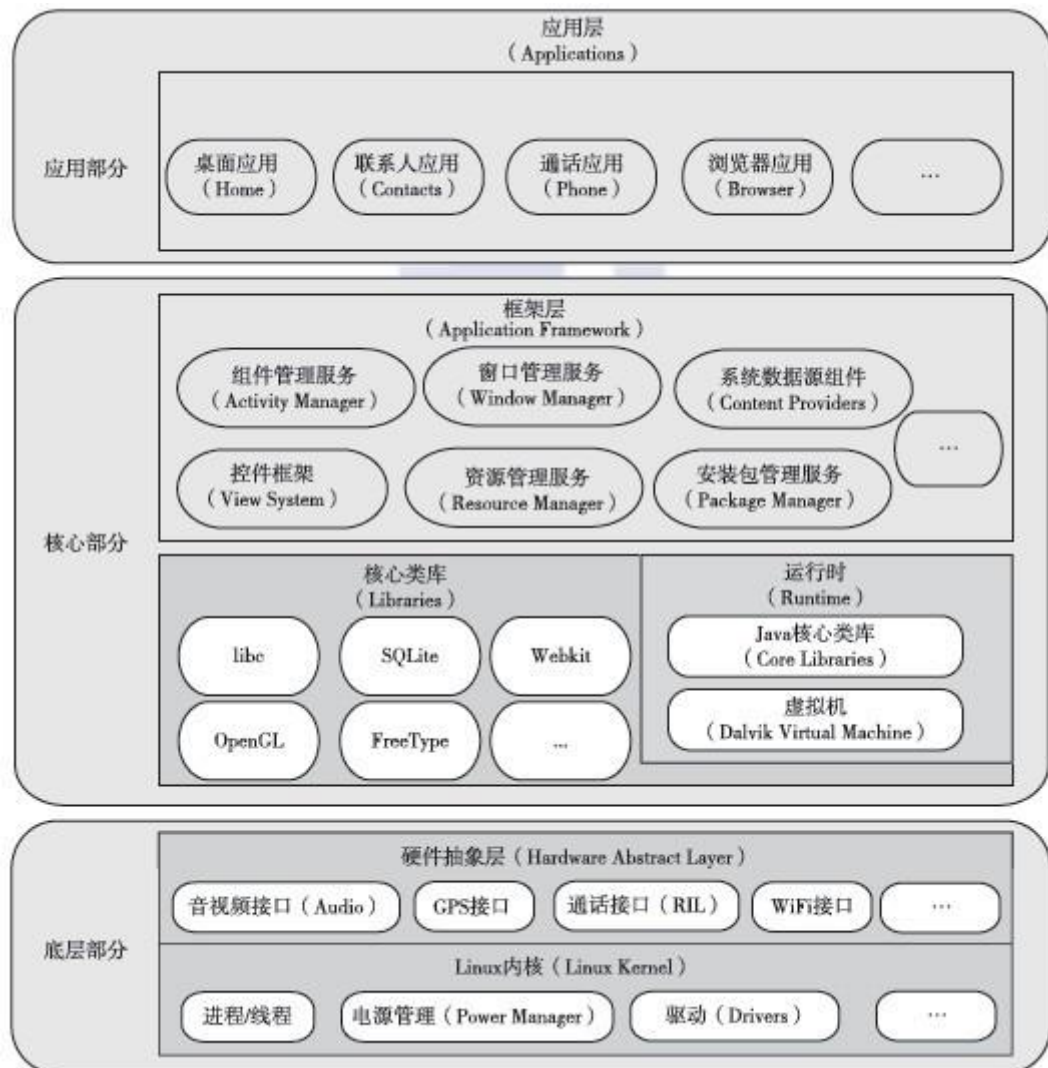


图 1-1 Android 系统架构图

1) 硬件抽象层 (Hardware Abstraction Layer)

Android 的硬件抽象层 (HAL)是一个关闭代码的硬件驱动模块^[27]。硬抽象层主要目标是将 Android Framework 与 Linux 内核分隔，使得 Android 不过度依靠 Linux 内核，从而达到 内核独立 的概念。这也让 Android 框架的开发可以在不考虑驱动程序先决条件下实现发展。

硬件抽象层存根代理是代理 (Proxy)的概念，存根文件是用 *.so 文档来存储。存根由硬件抽象层“提供”操作函数 (Operations)，并通过 Android 运行向硬件抽象层获取存根的操作函数，最后回调这些操作函数。硬件抽象层里包含了许多的存根。在运行时只需要说明“类型”，即模块 ID，就可以获取操作函数^[28]。

2) 编程语言

Android 是在 Linux 内核上执行的。在普通的 GNU/Linux 里包含的功能，在 Android 中大部分不存在。Android 想要实现产品的商业化，就一定要删除受 GNU GPL 授权证限制的全部，形如 Android 把驱动转移到用户空间的驱动，导致 Linux 驱动程序与 Linux 内核完全独立的。仿生/libc 中/内核/与内核头文件不是一个标准。Android 的内核头是通过工具从 Linux 内核头生成的，这是为了保持常量、数据结构与宏。

3) 中介软件

系统和应用之间建立联系的关键，化为 2 层：功能层和虚拟机。仿生是 Android 修改后的 libc。表面 flinger 代表 2D 或 3D 的内容在屏幕上的显示。Android 运用 Google 自制的仿生工具链 (Toolchain)。

Android 使用可称为基本的多媒体框架 OpenCORE。OpenCORE 划分为 7 大块：PV 播放器、PV 文本编辑器、编解码器、Packet 多媒体框架、作业系统相容性库 (OSCL)，常见的 OpenMAX。

Android 采用 skia 与 OpenGL/ES 作为其主要图形引擎。skia 与 Linux 开罗比较来看两者在作用上是同等的，但前者作用仅仅是准系统。Android 大多采用 Java 实现中间层，同时使用特别的 Dalvik 虚拟机。Dalvik 是“寄存器形式”的 Java 虚拟机，数据都存储在寄存器中，这样就使得虚拟机的指令数量有所下降。Dalvik 虚拟机能够实例多个实例，每个 Android 应用皆可使用自身的 Dalvik 虚拟机来执行应用，使得系统在执行应用时能够同时进行优化。Dalvik 虚拟机未执行 Java 字节代码 (Bytecode)，相反是执行 .dex 形式的文档。

1.3.2 Android 应用程序构成

一般来说，Android 应用程序主要由五部分组成^[29-31]，如图 1-2 所示：

(1) Activity

Activity 是 Android 应用程序五部分之中最基本的一部分，简单来说，每一个 Activity 都可以看做是一屏画面，Activity 一般会显示若干个 Views 控件，以此来作为系统与用户的接口，一般情况下，一个应用软件会包含若干个 Activity，通过互相之间的传递来达成画面的切换。

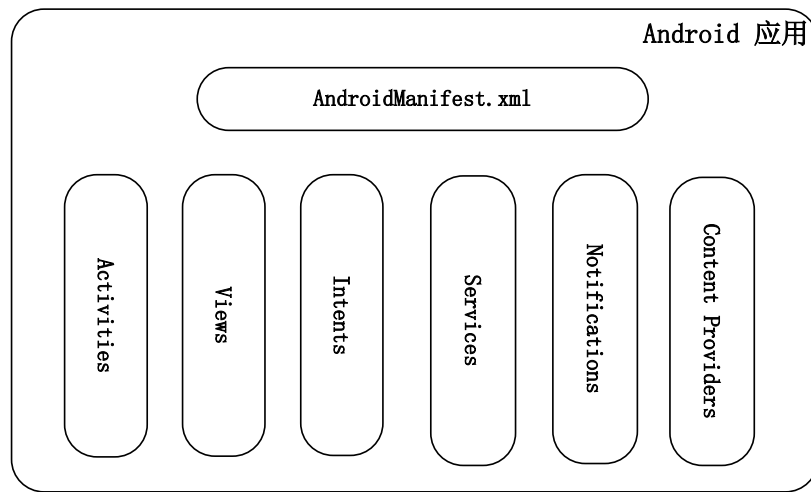


图 1-2 Android 应用程序结构图

(2) Service

Service 是一个程序，它并不存在与用户的洁面接口，它的生命周期一般很长，可以运行在后台，为应用提供服务，例如音乐播放便属于一个 Service，即使画面切换到其它部分，它依然可以在后台运行，我们还可以通过还清它来对其进行控制操作。

(3) Intent

Intent 是对消息进行简单传递的框架。你可以通过 Intent 在整个 Android 系统内对消息进行广播，并且可以对 Activity 进行消息传递，使 Activity 按照开发者的目的来执行相应的动作。并且一般来说，每个 Activity 之间相互的跳转动作都是利用 Intent 来完成的。

(4) Broadcast Receiver

Broadcast Receiver 是 Android 应用程序结构中负责监听的部分，它对 Intent 进行监听，当 Intent 产生时，如果符合设定好的响应条件，它就会被触发，并会对接收到的 Intent 事件做出相应的响应，根据其所接收到的 Intent 的描述完成对应的动作。

(5) Notification

Notification 会对用户进行提醒，每当接收到新的消息时，Notification 会通过图表、提示音或弹出消息框等方式通知用户系统接收到了新的消息。它在工作过程中并不需要获得 Activity 的焦点，并且也不需要中断正在执行的 Activity。

在 Android 应用程序中，Androidmanifest.xml 是一个配置文件。它负责对控件进行布局 and 定义。所有添加在页面上的控件都必须事先在该文件中声

明。

1.3.3 自动化测试技术

自动化测试技术就是使用自动化测试软件来实现系统测试,减少由人力操作来对软件的质量是否过关进行验证的系统,它不但可以直接的进行测试动作,并且可以对测试工作进行监控,分配和跟踪^[32-34]。自动化测试的实现一般是由测试脚本和第三方测试工具来完成,它的主要作用是使测试人员解放开来,避免了大量人力的浪费,并且可以使测试工作的工作效率得到有效的改善,使软件的开发周期变得更加紧凑^{[35][36]}。

在自动化测试的实际工作中,将自动化测试系统从驱动方式的角度来分类的话,一般可分为以下三类^[37]:

(1) 脚本回放类型

这种自动化测试系统使用程序设计的方式,测试人员可调用系统提供的接口编程实现自动执行 GUI 交互的流程,有时还需加入相应的期望检查点和预期测试结果。常见的编程语言有 Python、Perl 等;同时有针对 GUI 操作的编程语言,如 AutoIT; 和单元测试编程语言 JUnit、JFCUnit 等^[38-40]。

该类型自动化测试系统是专门针对某一被测系统定制的,因此该系统具有良好的适应性。但是同时系统也具有很差的灵活性,用例一旦完成很难进行修改和变更^[41]。当 GUI 有变动时系统就无法支持,一旦软件变动过大使得该测试系统在测试过程中无法正常使用,唯有对测试程序进行相应的修改,这样效率就会有所下降。

(2) 录制回放类型

该类型的自动化测试系统要求测试人员先对 GUI 进行手工操作,自动记录测试执行的过程和校验点作为测试用例或是测试数据^[42]。随后将记录的操作作为测试用例在被测试软件上执行并获取测试结果数据,以此来实现 GUI 的功能测试。

这种测试方式是更加成熟的测试类型,是现下使用最为广泛的测试方式,但是仍需较多的人工参与。在软件开发的过程中,软件在不断变更的情况下其录制的测试过程或是校验点可能会不同,就需要重新进行录制。由此可看出这种驱动方式的自动化测试系统对于自动生成测试用例基本没有什么有效的帮助。

(3) 数据驱动类型

该类型的自动化测试系统是根据系统交互进行的封装,也可认为是对类型

(2)的改善,系统把要实施的测试步骤信息采用数据的形式传送给回放系统^[43]。也可以采用记录的方式获取测试数据,同时仅仅通过修改关键字用户就可操控测试的执行过程^[44]。该方式会通过创建一个与系统相对应的测试数据来进行管理,往往是一个测试用例配有相对的一个测试数据池的使用权,如IBM Rational Functional Tester^[45]。

该方式使得创建和处理测试用例的过程变得简单。当由于软件 GUI 有很多改动而导致测试无法正常执行时,仅需改动相匹配的测试数据就可。但在实际自动化测试执行过程中还是有很大的程度上依赖于手工操作,仍缺少一个科学的方法来评价和控制测试用例。

1.3.4 Python 技术语言

Python 语言是一种语法相对简单,且应用极广的面向对象的编程语言。它拥有超强的功能,到目前为止,已经被广为使用,并且深受开发人员的喜爱,拥有一批坚实的支持者。由于其简单的语法特点和易操作性,使其能应用于各种 OS (Operating System) 中,且被越来越多的开发人员所认可^[46]。

Python 语言的特点有:

- (1) Python 语言能够使使用它的人员不需要将注意力集中在开发语言上面,仅仅关注需要解决的问题本身即可。它倡导简单主义思想。
- (2) Python 及其容易上手因为其语法结构简单。
- (3) Python 的可移植性非常强,无需修改就可以在任务平台下运行。
- (4) 可以把 Python 嵌入到 C/C++ 程序,从而向用户提供脚本功能。
- (5) Python 既支持面向过程的编程,同时也支持面向对象的编程。

1.4 本文的主要研究内容

本课题项目旨在于开发针对 Android 平台的自动化测试系统,使自动测试框架能够更加稳定高效的运行,提高测试用例的稳定性和可移植性、减少测试用例的运行的时间、提高测试过程中对被测设备有效 Bug 的检出,以及对设备发生异常时及时进行的处理等。测试用例控制模块作为测试框架和测试用例之间的中间层,主要是为了提高测试用例操作的共通性,以及测试用例运行的稳定性而设计的。重点要研究以下四方面内容:

- (1) 解决测试用例执行不稳定不完全的问题,导致测试用例可能出现执行不稳定不完全的问题的主要原因有两个:一个是 ADB 工具自身不稳定导致的问题。另一个是由于很多设备都是处在正在开发中的状态,所以经常会

出现很多未知的不稳定的状况,例如测试用例在执行的过程中设备会突然发生重启等情况。

(2) 解决测试结果输出混乱问题,由于输出的信息中同时包括系统的常规运行 log 和测试用例的执行 Log,它们混合在一起输出时很容易导致测试结果显示混乱,并很难将所需要的测试结果提取出来,所以需要将一般的运行结果和测试用例的测试结果分割开。并将需要的数据以清晰的格式输出。

(3) 对测试用例操作的共同部分进行封装,将 ADB 命令序列及常用的复杂操作封装起来,供测试开发人员在编写测试脚本时调用,减少测试脚本编写时的工作量,提高修改及维护测试脚本的效率。

(4) 测试用例的合理分类及详细设计。

1.5 本文组织结构

本论文的组织结构主要由下面几部分构成:

第 1 章:绪论。描述了“Android 平台的自动化测试系统”这一课题的来源及背景,分析了研究的目的、意义,描述了与本课题相关的国内外发展概况以及介绍了与本课题有关的一些理论概述。

第 2 章:自动化测试系统的设计与实现的需求分析与总体设计。分析了 Android 平台的自动化测试系统的设计与实现的需求。为满足测试质量要求,根据实际测试工作要求转化为针对于系统的需求,分别描述了 Android 平台的自动化测试系统的设计与实现的功能需求和非功能需求。其中系统的主要功能是自动执行测试用例、监控测试运行状态、格式化输出测试结果、提供自动化测试系统的本地调用、生成测试脚本模板和对 ADB 命令进行封装。并介绍了自动化测试系统的总体逻辑架构和模块设计。

第 3 章:Android 平台的自动化测试系统的设计。根据第 2 章 Android 平台的自动化测试系统需求分析结果,对 Android 平台的自动化测试系统进行了总体的设计。首先明确了 Android 平台的自动化测试系统的执行流程和整体框架。接着对 Android 平台的自动化测试系统进行了功能分解,分解成五大模块——测试用例加载模块、ADB 接口模块、测试结果输出模块、常用操作封装模块,部分模块又细分成多个子模块。最后分别对上述五大模块进行了概要设计和详细设计。这一章的设计工作为系统的具体实现奠定了基础。

第 4 章:Android 平台的自动化测试系统的实现。根据第 3 章 Android 平台的自动化测试系统的设计结果,对 Android 平台的自动化测试系统进行了实现。分别描述了五大模块——测试用例加载模块、ADB 接口模块、测试

结果输出模块、常用操作封装模块的实现方案。

第 5 章：Android 平台的自动化测试系统的功能测试。描述了 Android 平台的自动化测试系统的功能验证方法，根据第 2 章的需求分析设计测试用例，分别对自动执行测试用例、监控测试运行状态、格式化输出测试结果、提供自动化测试系统的本地调用、生成测试脚本模板和对 ADB 命令进行封装进行了功能测试，进行了非功能测试，最后对测试结果进行了总结。

结论：作为论文的最后一部分，对论文的研究成果进行了总结，对今后的研究方向做了展望。

通过以上的的工作，对 Android 平台的自动化测试系统进行了设计与实现，实现测试工作中的自动化测试方式，提供了有效的解决方案。

第 2 章 自动化测试系统的需求分析与总体方案

2.1 自动化测试系统的需求分析

在每一款新研发的 Android 设备上市之前,都需要进行大量重复的测试,如电源消耗测试,系统性能测试,图形基准测试,应用性能测试等等。尽量使设备在上市前将硬件和软件两方面都可能出现 BUG 的风险减少到最小。而在实际的测试的过程中,往往会出现很多未知的不稳定的状况,例如测试用例在执行的过程中设备会突然发生重启、ADB 工具突然挂掉等情况。导致测试用例执行不稳定不完全。本论文旨在于开发针对 Android 平台设备的自动化测试系统,使自动测试能够更加稳定高效的运行,提高测试用例的稳定性和可移植性、减少测试用例的运行的时间、提高测试过程中对被测 Android 设备有效 Bug 的检出,以及对被测 Android 设备发生异常时进行及时的处理等。Android 自动化测试系统作为外部测试框架和测试用例之间的中间层,主要是为了提高测试用例操作的共通性,以及测试用例运行的稳定性而设计的。

2.1.1 自动化测试系统的功能需求

首先根据对 Android 设备进行自动化测试的操作流程进行梳理可知,自动化测试系统应在正常启动后,加载相应的测试用例,然后将测试数据自动送入相应的测试脚本,并做好测试的准备工作,如安装好测试所需的 APK 文件或导入测试所需的数据文件等。然后需要根据测试脚本的编写自动执行测试,在脚本运行过程中监控运行状态,如果发生异常的话需要能够响应,做出相应的判断,并记录异常信息。在测试结束时按照格式输出测试结果,然后将测试结果数据返回给测试框架。

整个测试的流程分析如图 2-1 所示。

由上述测试流程中可知系统应具有如下功能:

(1) 自动执行测试用例

测试的执行过程是按照生成的测试用例自动执行测试脚本进行测试工作。执行过程为模拟手工操作在手机上自动进行软件响应的操作,按照测试脚本自动点击 Android 设备软件上的控件如按钮等,实现模拟用户真实操作

软件流程，从而代替手工测试工作。

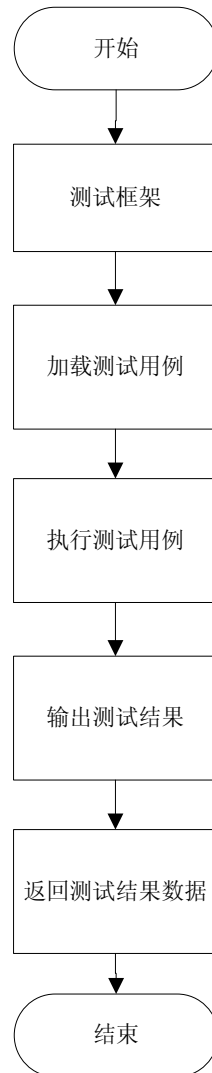


图 2-1 整体测试流程图

(2) 监控测试运行状态

根据以往在对 Android 设备进行测试时可发现,由于 ADB 工具自身不稳定,经常会出现死锁,使测试停止运行。另一个是由于很多 Android 设备都是处在正在开发中的状态,所以经常会出现很多未知的不稳定的状况,例如在运行的过程中会突然发生重启等情况。所以根据分析可得出,我们的系统在执行测试用例的过程中,需要对 ADB 工具和待测设备分别进行监控,在 ADB 发生死锁时,可以重新启动 ADB Server。记录每个测试用例的具体执行操作和相对应测试结果。将测试记录结果存储在指定文件中,其中记录操作过程是为方便在测试不通过时进行错误定位。并且在发生错误时可以在故障点继续执行测试用例。并监控被测设备在测试过程中是否出现了 Crash, 如果出

现，需要抓取 Crash 信息。

(3) 格式化输出测试结果

本系统是整个自动化测试框架的测试控制和执行部分，并不需要生成测试报告，但需要按照设计好的固定格式自动返回测试结果，将一般的运行结果和测试用例的测试结果完全分割开。一般的运行结果是通过 Python 自带的 Logging 来抓取，使用一个统一的 LogServer 管理，测试用例的测试结果放到标准输出中。使两者就互不干扰。这部分将读取在测试校验中记录的校验结果文件，将文件进行统一过滤处理生成最后格式统一的测试结果，然后将结果返回给测试框架。

2.1.2 自动化测试系统的非功能需求

满足上述功能的同时，还需要满足一定的非功能需求。随着产品新版本的操作逻辑和基本功能等的变更，自动化测试为了与其保持同步就需要不断的维护和扩展，这就使得自动化测试成为一个长期的过程。为此在设计自动化测试系统时考虑自动化测试在将来的可维护性与移植性是至关重要的，所以需要常用的操作封装起来，以便于测试人员编写测试脚本时可以直接调用，避免大量的重复劳动。与此同时保证完整的进行自动化测试的也是同样重要的。这就要求自动化测试即能够覆盖到绝大多数的功能，并能够对系统性能进行测试。这就需要做到布局自动化测试执行脚本的结构和对测试用例的分类要有的合理性。通过使用注释可使自动化测试具有更好的可读性。通过模块化设计使得模具重用性，更便于更新。

在非功能方面需要具有如下特性：

(1) 稳定性

自动化测试系统本身应是具有稳定性的。由于进行自动化测试时可能会有长时间运行系统的时候，为此测试系统就要具有一定的稳定性，否则测试将无法正常工作。这部分将要求系统连续运行 48 小时以上。

(2) 高效性

以往的测试用例用了很多 sleep 操作来等待某个操作执行完毕，很难把握使用 sleep 等待的时间，并且会浪费执行时间。使测试效率变低。这就需要自动化测试系统可以监控设备的运行状况，在出现异常时，能够及时恢复现场，保证测试用例可以继续执行，以减少再次重新运行所造成的时间浪费。提高测试时与设备的交互，及时抓取设备的运行状态，并根据设备的运行状态判断测试下一步的流程。

(3) 可移植性

使测试用例需要应用在其他测试框架下或运行在 Linux 或 Cygwin 时，可以方便的移植，不需要重新编写测试脚本，而可以直接移植，或仅仅对中间层部分进行很小的改动。

(4) 功能点覆盖

对于自动化测试系统对于功能点覆盖的要求也很高，尽可能的覆盖全部测试功能点。这部分的覆盖标准，主要考虑的是事件覆盖和事件交互覆盖两种方式。通过对测试用例进行合理的分类，并在设计测试用例时抽取恰当的测试观点，提高错误的检出率。

2.2 系统的实施方案

2.2.1 自动化测试系统整体逻辑架构

根据对自动化测试系统进行的需求分析，并结合实际测试的情况，我们可以得出自动化测试系统运行是的一个整体逻辑架构。

自动化测试系统的逻辑架构图如图 2-2 所示。

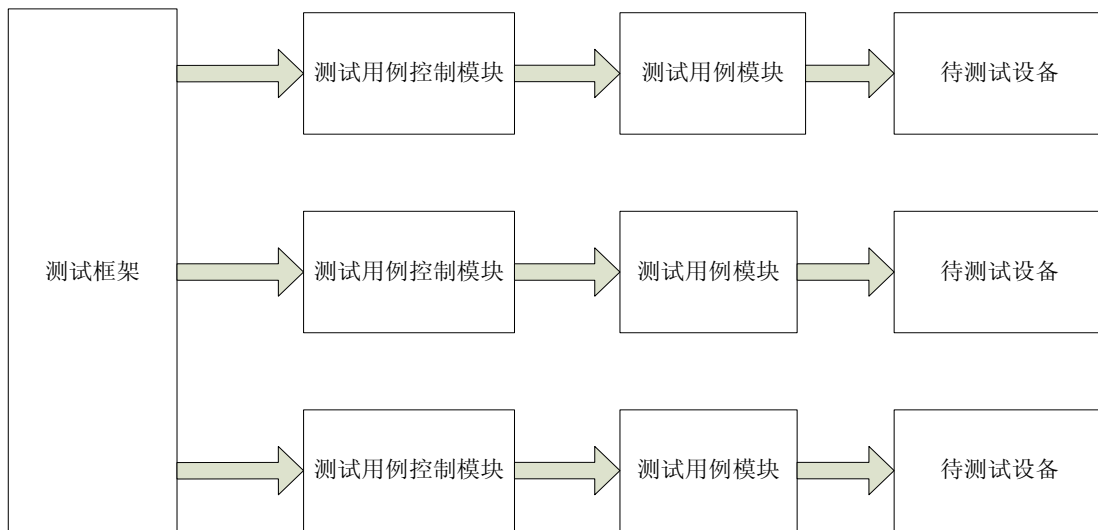


图 2-2 自动化测试系统逻辑架构

可以看出，在执行自动化测试的过程中，一般主要由以下四部分组成：

(1) 测试框架

调用本自动化测试系统的外部测试框架，用来管理和组织分配测试用例，接收测试结果生成测试报告。

(2) 测试用例控制模块

该部分为本次课题的设计重点，它是自动化测试系统的核心，作为一个测试用例控制部分，在它的外部测试框架和测试用例之间搭建了一个中间层，用来驱动测试用例，监控测试用例运行状态，并可以监控设备的运行情况，防止发生 ADB 的死锁，防止测试用例进入死循环后无法退出。并能够将测试用例共同的部分抽取出，提高测试时与设备的交互。

(3) 测试用例模块

具体进行测试工作的 Python 脚本。根据具体的测试要求进行编写。例如通过调用第三方工具来评估设备状态。如可以测试硬件性能的 GLBenchmark、Quadrant 等第三方跑分软件。然后抓取测试结果。测试用例可以根据具体需要编写。

(4) 待测试设备

搭载 Android 系统的被测设备。包括平板电脑和智能手机两大类。

2.2.2 测试用例控制模块

测试用例控制模块为本自动化测试系统的设计重点，它作为整个测试工作流程的控制部分，在测试框架和测试用例之间搭建了一个中间层，用来驱动测试用例，监控测试用例运行状态，防止 ADB 工具发生死锁，防止测试用例进入死循环后无法退出，并可以监控被测设备的运行情况，提高测试时与设备的交互。在被测设备发生意外重启、系统 crash 等问题的时候测试用例控制模块可以抓取设备的错误信息。方便开发人员判断测试过程中出现的问题。而且测试用例控制模块能够将测试用例共同的部分抽取出来，方便测试用例的维护，减少开发人员的工作量。而且可以过滤测试结果，将测试结果数据以外部测试框架能够解析的规范化格式输出，方便生成测试结果报告。

测试用例控制模块的结构图如图 2-3 所示。

测试用例控制模块主要包括以下几个子模块：

1) 测试用例加载模块：外部测试框架调用自动化测试系统的入口，同时负责每个测试用例的执行流程。并负责提供测试用例所需要的数据，安装所需的测试工具。在进行测试之前将测试数据 push 到被测设备中，将所需的 apk 文件或测试工具安装好。在测试结束后，将这些内容从设备中删除。另外，本模块还包含了两个子功能：

(1)测试用例创建功能：生成一个包括所有必要结构的测试用例模板，防止测试人员在编写测试脚本时遗漏某些重要问题，方便开发人员编写新的测试脚本。

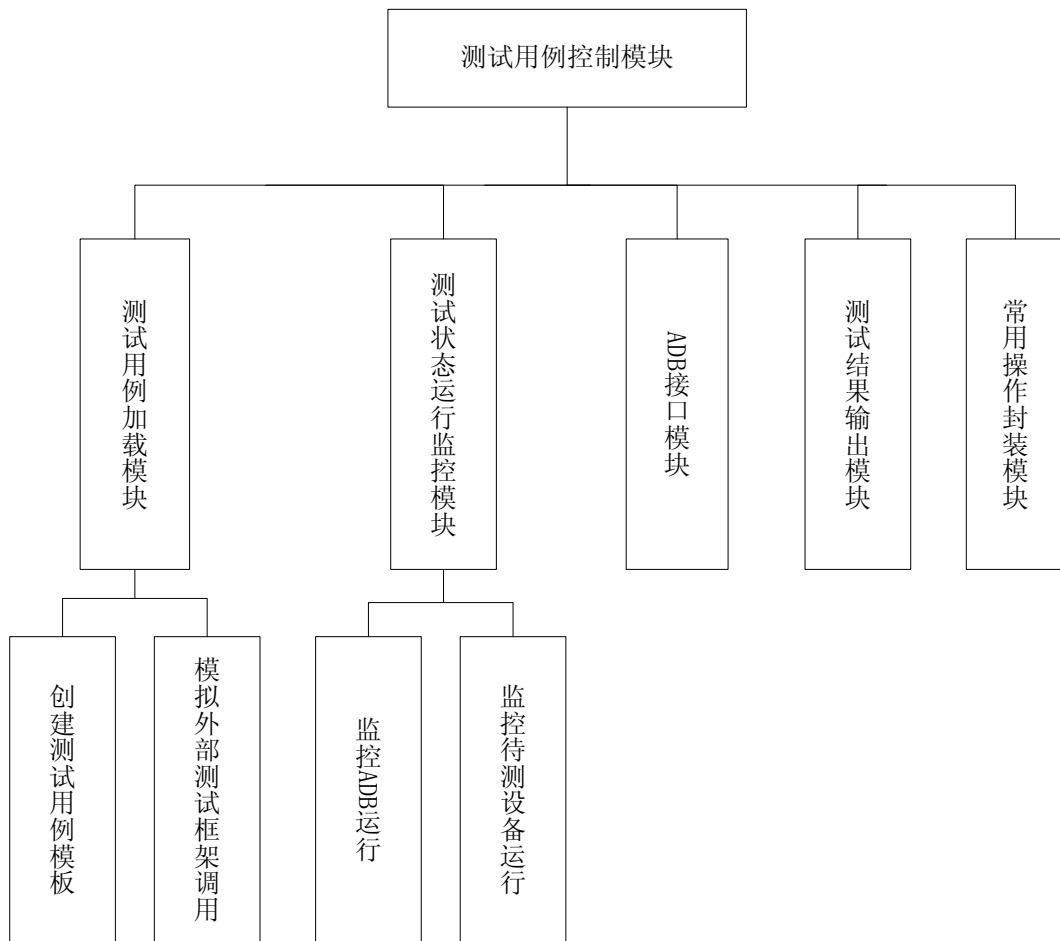


图 2-3 测试用例控制模块结构图

(2)模拟外部测试框架调用功能：可以模拟外部测试框架对测试用例控制模块的调用动作，进行测试活动。方便调试以及可以在本地进行测试工作，

2)测试状态运行监控模块：监视设备在测试用例运行的过程中是否出现了断开连接等问题，如果出现，抓取错误信息。监视 ADB 的运行状态，出现死锁的时候，重新启动 ADB Server，并将当前操作重新执行。

3)ADB 接口模块：对于 Android 设备操作的 ADB 接口的封装，保证测试用例对设备操作的接口的统一性。方便测试用例调用。

4)测试结果输出模块：统一封装各种测试结果的输出形式，保证每个分类的测试用例的输出结果的统一性。便于测试结果数据被测试框架解析，生成测试报告，写入数据库。并且使用 Python 已有的 Logging 模块搭建的 logclient，用来输出测试过程中的信息。便于开发人员分析系统运行情况。

5)常用操作封装模块：将一些常用的操作从提取出来并封装，供需要的测试用例直接调用。这样可以很大程度的减少测试用例编写和维护的成本。

2.2.3 测试用例控制模块的业务流程

本课题中的测试用例控制模块是对测试框架进行优化的重点，根据设计，它的业务流程如图 2-4 所示。

测试用例控制模块首先调用测试用例加载功能，此时并没有加载测试用例，而是在为加载测试用例做准备工作。然后控制模块会自动调用 Log 输出功能，开启 LogServer。此处的设计是为了防止一般的 Log 对测试结果进行干扰。加载完 Log 输出功能之后控制模块会自动调用设备状态监视功能，开启对设备的监控，监控设备可能出现的异常状态，根据设备的状态会判断之后的操作。到这里之后控制模块便会正式加载测试用例。在此处测试系统调用的全部为测试用例基类所定义的接口。之后控制模块调用测试数据控制功能向待测设备中 push 进测试数据。

测试用例控制模块之后会调用 ADB 接口，对设备按用例中脚本编写的流程进行操作。在操作的同时，控制模块会开启 ADB 监视功能，监视 ADB 状态。防止 ADB 出现死锁。与此同时控制模块会自动调用常用操作的封装模块，该模块可以直接进行设备操作，如解锁设备，重启测试设备等。在此处是测试用例具体实现的部分。通常测试用例里的操作都是由 ADB 命令进行控制，一个常用的操作实际上是很多个 ADB 接口调用的集合，即通过顺序调用不同的 ADB 接口来实现某个特定操作。常用操作封装模块可以大大的简化编写测试用例的复杂度。这也是实现自动化测试的要点所在。即通过封装好的 ADB 命令来模拟人的实际操作。

测试用例运行完成之后，调用测试结果输出模块，该模块统一封装各种测试结果的输出形式，这样可以保证每个分类的测试用例的输出结果的统一性，清晰的输出测试结果。并且一旦格式发生了变更，主要修改该模块内的格式就可以了，这样可以减少开发的负担。此时测试数据控制功能工作，将测试前 push 到设备中的数据删除掉。最后控制模块再次调用设备状态监视功能，check 设备是否出现 Crash 状况，如果出现，抓取 Crash 信息。最终，将处理完毕的测试结果返回给测试框架，以上便是自动化测试系统进行一次测试的完整控制流程。

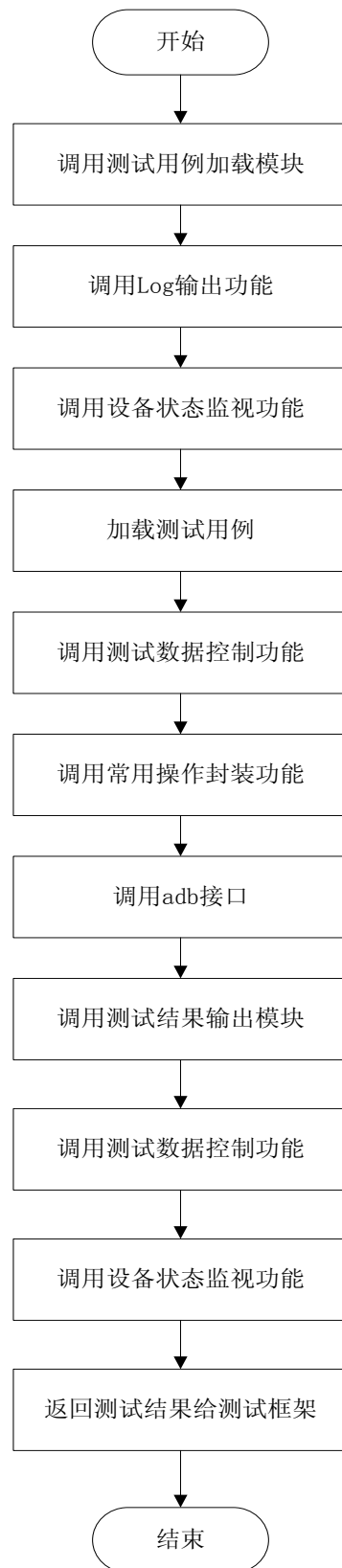


图 2-4 测试用例控制模块的完整业务流程图

2.3 预期达到的目标

下面将从功能性和非功能性两方面介绍项目预期达到的目标。

2.3.1 功能性目标

控制测试执行流程；监控设备的运行状况，出现异常时，及时恢复现场，保证测试用例可以继续执行，以减少再次重新运行一次所造成的时间浪费。使运行 Log 和测试用例的测试结果互不干扰。提高测试时与设备的交互，及时抓取设备状态，根据设备的状态，判断下一步的流程。使测试用例可以尽可能多的覆盖需要测试的部分。

2.3.2 非功能性目标

通过设计良好的测试用例，将测试用例分类，保证测试的全面性，实现自动化测试系统的各项指标。提高测试用例的高效性、可移植性和稳定性。

2.4 本章小结

本章对自动化测试系统进行了需求分析。分析了在实际测试应用中所涉及的具体测试工作需求。之后根据分析结果，将这些需求转化为针对于系统的需求，分别描述了自动化测试系统的功能需求和非功能需求。并设计和确定了系统的实施方案。这一章是理解后续章节的基础，以下几章将对自动化测试系统分别进行设计、实现和测试。

第 3 章 Android 平台的自动化测试系统的设计

3.1 系统的总体设计

根据第 2 章中的需求分析和实际测试执行流程可知自动化测试系统总体架构图如图 3-1 所示：

本测试系统架构中也使用了经典的三层结构，分别为数据层、测试逻辑层和操作界面层。

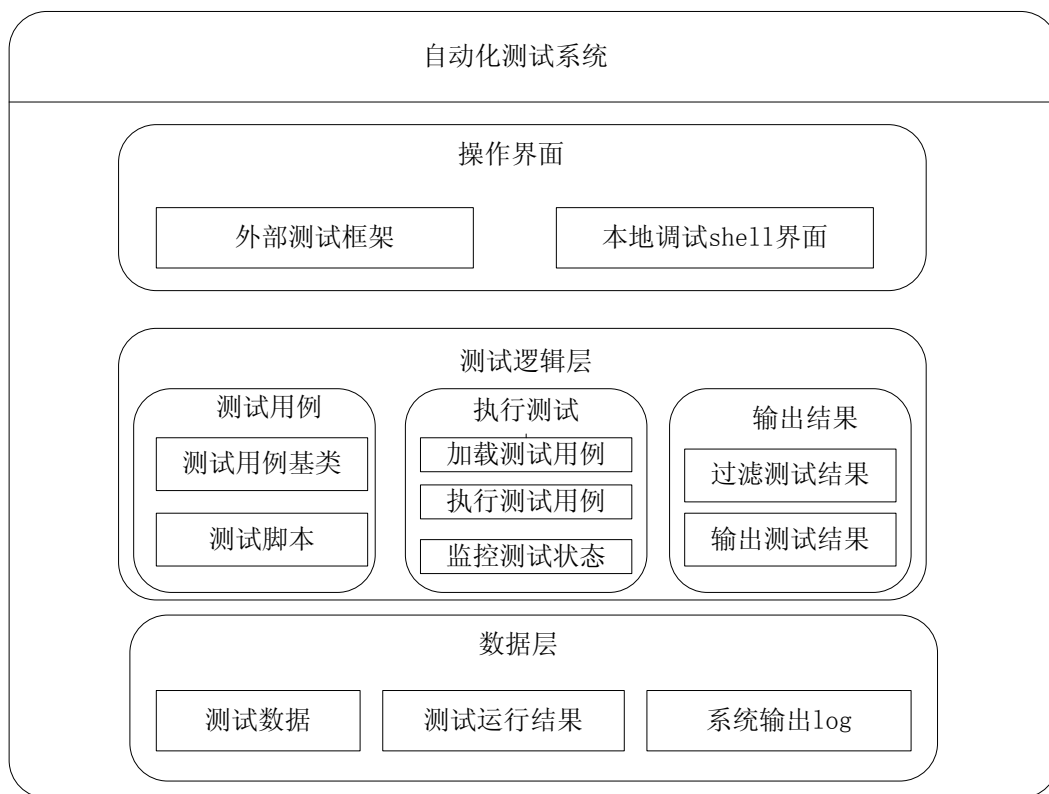


图 3-1 自动化测试系统逻辑框架

其中数据层包括系统的各个测试用例所需的数据、返回给外部测试框架用来生成测试报告的测试结果数据、供开发人员调查运行问题的系统运行 log 数据；测试逻辑层包括本系统的核心部分测试用例控制模块、测试用例模块；UI 层则是系统用户端界面展示，包括系统功能展示和测试结果展示。由于本系统在工作中是被本项目中的测试框架部分调用的，所以本系统的设计重点是实现自动化测试的逻辑控制，UI 设计部分不包含在本设计之内，具体的

操作通过测试框架部分的 UI 界面来显示，通过测试框架提供的 GUI 来选择具体的测试活动。为了方便调试，并实现使本自动化测试系统可以独立运行完成自动化测试工作，系统设计了一个模拟测试框架调用动作的调试功能，可以通过 shell 窗口来控制系统进行测试工作。

自动化测试系统的主要架构，由以下两部分组成：

(1) 测试用例控制模块

该部分为本次课题的设计重点，它作为一个测试用例控制部分，在测试框架和测试用例之间搭建了一个中间层，用来驱动测试用例，监控测试用例运行状态，并可以监控设备的运行情况，防止发生 ADB 的死锁，防止测试用例进入死循环后无法退出。并能够将测试用例共同的部分抽取出，提高测试时与设备的交互。将测试结果按标准格式输出，提供给测试框架，使其生成具体的测试报告。本文所研究的自动化测试系统是为本项目的另一个子项目的自动化测试框架提供稳定的测试服务所研发的。为了方便调试以及可以在本地进行测试工作，在测试用例控制模块中设计了一个模拟测试框架的功能，可以对测试用例控制模块进行调用，进行测试活动。

(2) 测试用例模块

该部分为具体进行测试工作的 Python 脚本。根据具体的测试要求进行编写。为了能够覆盖待测设备的待测点，根据待测试设备需要被测试的部分，主要将测试用例分为以下几类：应用性能测试用例、通用性测试用例、图形基准测试用例、平台函数测试用例、电源消耗测试用例、以及系统性能测试用例。该部分可以在之后的工作中根据用户需求不断扩充完善。

3.2 测试用例加载模块的设计

测试用例加载模块是整个测试用例控制模块的基础，它为外部测试框架提供了调用接口，并且负责加载测试用例，同时负责每个测试用例的整个执行流程。它首先对测试用例进行初始化操作。负责提供测试用例所需要的数据，安装所需的测试工具。一般都是会在进行测试之前将测试数据 push 到被测设备中，并将所需的程序的 apk 文件或调用的测试工具安装好。之后便会驱动测试脚本运行。并会在相应的阶段调用其它功能模块进行相应的动作。在测试结束后，测试用例加载模块会将之前安装进设备的程序和数据从设备中删除。

另外，本模块还包含了两个子功能：

(1)测试用例创建功能：生成一个包含编写测试脚本时必要结构的测试用

例模板，防止测试人员在编写测试脚本时遗漏某些必需步骤，方便开发人员编写新的测试脚本。

(2)模拟外部测试框架调用功能：可以模拟外部测试框架对自动化测试系统的调用动作，在本地直接进行测试工作。这个功能可以方便测试开发人员调试测试脚本，方便观察测试运行情况，并可以当需要时在本地进行测试工作。

测试用例加载模块主要由 `tc_loader`、`Installer` 两个类以及 `tc_main` 接口组成。具体类图如图 3-2 所示。

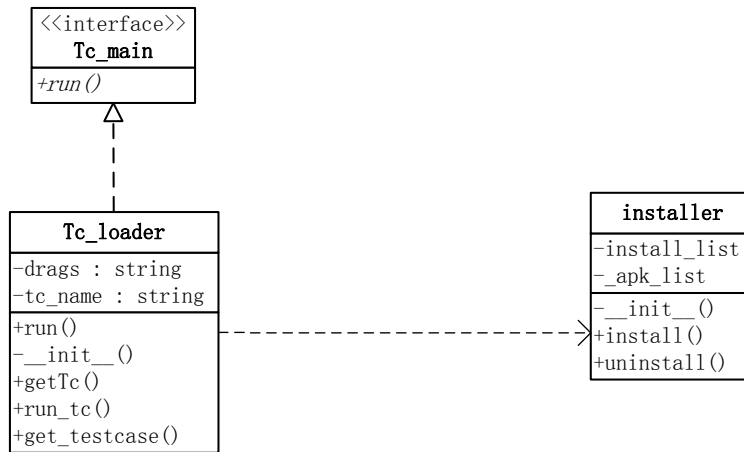


图 3-2 测试用例加载模块类图

类图中各个类以及类中各个方法的描述如表 3-1 所示。

表 3-1 测试用例加载模块相关类及方法描述

类	方法	作用
tc_main		提供了自动化测试系统被外部调用的接口
	run()	进入测试系统的事件函数
tc_loader		测试用例加载模块的核心类。控制整个测试流程
	__init__()	初始化函数
	getTc()	关联测试用例的事件函数
	run_tc()	执行测试用例的事件函数
	get_testcase()	加载测试用例的事件函数

表 3-1(续表)

类	方法	作用
installer		为执行测试用例做一系列准备工作
	<code>__init__()</code>	初始化函数
	<code>install()</code>	加载测试数据，安装测试所需的 apk 文件 与测试工具的事件函数
	<code>uninstall()</code>	测试结束后删除测试数据，删除已安装的 apk 文件与测试工具的事件函数

3.3 测试状态运行监控模块的设计

测试状态运行监控模块是提高测试用例执行稳定性的重要模块。它负责对测试用例执行过程中异常情况发生的监控，主要有两个功能。

第一个功能是监控 ADB 工具的运行状态是否正常，是否有 ADB 死锁等现象出现。由于对 Android 设备的自动化测试往往都是通过 ADB 命令序列来控制待测 Android 设备模拟人工执行动作来进行测试工作，所以当 ADB 工具发生类似于死锁的问题时往往会导致自动化测试进入死循环状态。一旦这种情况出现，测试状态运行监控模块能够及时恢复现场，保证测试用例可以继续执行，以减少在发生异常后重新执行一遍测试工作所造成的时间浪费。

第二个功能是监控设备运行状态，由于待测设备往往都是处在开发中的状态，所以经常会出现不稳定的状况，如在执行过程中设备重启等问题。这样会影响测试结果的准确性。为了解决这个问题，测试状态运行监控模块会对设备运行状态进行监控，监控模块会抓取相应的内核 log，并将监视结果反馈给开发人员，使测试人员能够一目了然的了解到在测试用例执行过程中待测设备是否正常运行。

测试状态运行监控模块主要由 `device_status_monitor`、`HandlerBase`、`Uptime_Handler`、`Boot_Count_Handler`、`Watchdog` 这五个类组成。具体类图如图 3-3 所示。

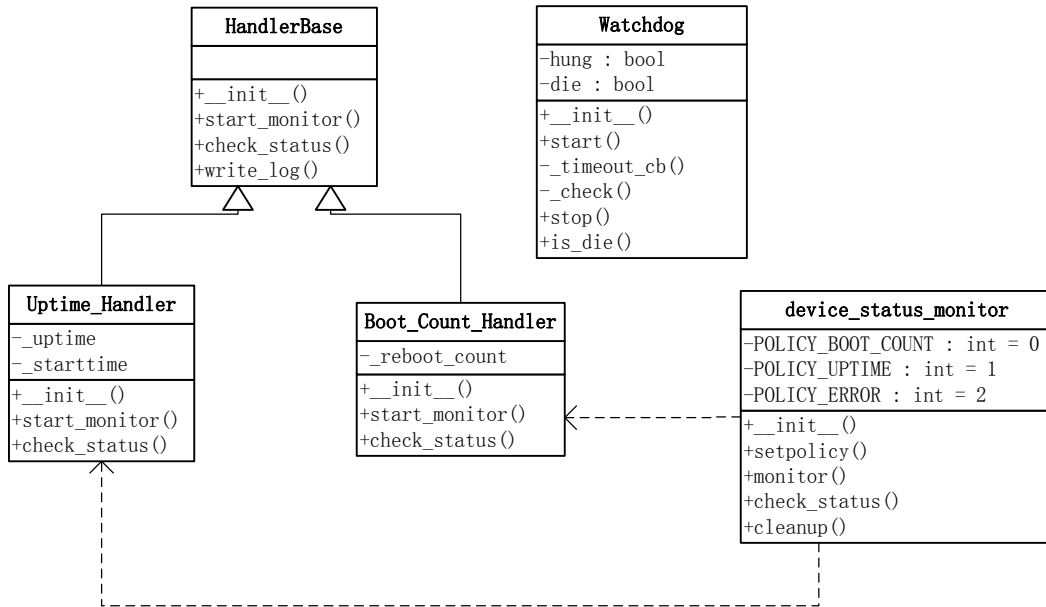


图 3-3 测试状态运行监控模块类图

类图中各个类以及类中各个方法的描述如表 3-2 所示。

表 3-2 测试状态运行监控模块相关类及方法描述

类	方法	作用
Device_status_monitor		监视设备运行状态的控制类
	__init__()	初始化函数
	setpolicy()	选择设备监视策略的事件函数
	monitor()	判断监视策略选择结果的事件函数
	check_status()	检查监视状态的事件函数
	cleanup()	清空监视结果的事件函数
HandlerBase		对待测设备运行状态进行监视的基类
	__init__()	初始化函数
	start_monitor()	启动监视待测设备运行状态的事件函数
	check_status()	检查监视状态的事件函数
	write_log()	输出设备监控结果的事件函数

表 3-2(续表)

类	方法	作用
Uptime_Handler		采用时间策略判断设备运行状态的类
	__init__()	初始化函数
	start_monitor()	启动监视待测设备运行状态的事件函数
	check_status()	检查监视状态的事件函数
Boot_Count_Handler		采用启动计数策略判断设备运行状态的类
	__init__()	初始化函数
	Start_monitor()	启动监视待测设备运行状态的事件函数
	check_status()	检查监视状态的事件函数
Watchdog		监控 ADB 运行状态的类
	__init__()	初始化函数
	start()	启动 ADB 运行状态监控的事件函数
	timeout_cb()	解决 ADB 运行超时状态的事件函数
	check()	监控 ADB 运行状态的事件函数
	stop()	终止对 ADB 命令监控运行状态的事件函数
	Is_die()	返回 ADB 运行异常 log 的事件函数

3.4 ADB 接口模块设计的设计

ADB 接口模块为测试用例提供了对于 Android 设备操作的 ADB 接口的封装，保证测试用例对设备操作的接口的统一性。这样测试用例在执行 ADB 操作时不需要考虑具体的 ADB 工具调用问题，可以直接对封装好的 ADB 接口进行调用，进行对待测设备的操作。

ADB 接口模块主要由 ADB 和 Executor 两个类组成。具体类图如图 3-4 所示。ADB 类提供了封装了各种 ADB 接口，Executor 类提供了对 ADB 工具进行操作的方法,并封装了 Python 的 subprocess 模块中的一些方法，使执行每条命令和对每条 ADB 命令进行监听时都是独立的一条进程。

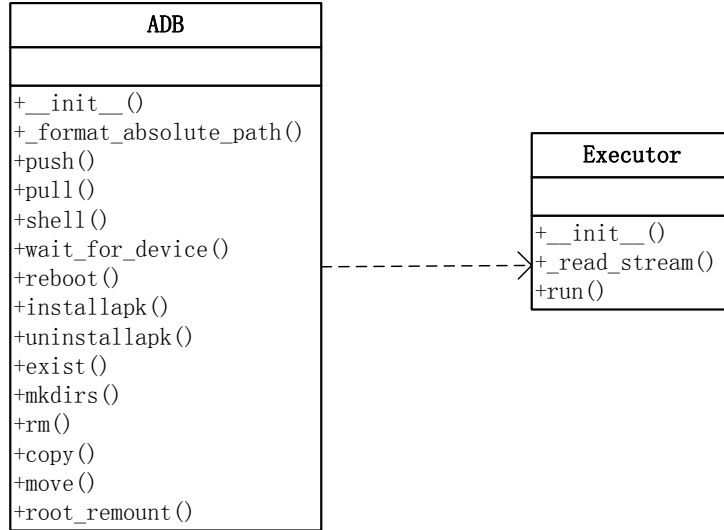


图 3-4 ADB 接口模块类图

类图中各个类以及类中各个方法的描述如表 3-3 所示。

表 3-3 adb 接口模块相关类及方法描述

类	方法	作用
ADB		ADB 各种操作的调用接口类
	<code>__init__()</code>	初始化函数
	<code>format_absolute_path()</code>	设置绝对路径的事件函数
	<code>push()</code>	向待测设备装载数据的事件函数
	<code>pull()</code>	从待测设备取数据的事件函数
	<code>shell()</code>	adb Shell 命令接口
	<code>wait_for_device()</code>	等待待测设备响应的事件函数
	<code>reboot ()</code>	重启待测设备的事件函数

表 3-3(续表)

类	方法	作用
	installapk()	在待测设备上安装 apk 文件的事件函数
	uninstallapk()	卸载待测设备中已安装的 apk 文件的事件函数
	exist()	检查待测设备中文件是否存在的事件函数
	makedirs()	新建文件夹的事件函数
	rm()	删除待测设备中数据的事件函数
	copy()	复制待测设备中数据的事件函数
	move()	移动待测设备中数据的事件函数
	root_remount()	使待测设备获得 root 权限的事件函数
	Executor	对 ADB 进行操作类
	__init__()	初始化函数
	read_stream()	读取数据流的事件函数
	run()	执行 ADB 命令的事件函数

3.5 常用操作封装模块的设计

在测试用例的编写过程中，有一些必须执行的操作非常繁琐，如重新启动设备到启动器界面，对屏幕解锁，连接 WIFI 网络，注册账户等等。这些需要一系列大量的 ADB 命令来完成。常用操作封装模块便是为此设计的。模块由 System、Wlan、Account、Contacts 四个类组成，具体类图如图 3-5 所示。System 类封装了常用的系统操作接口；Wlan 类封装了对 wlan 网络的常用操作接口；Account 类封装了对账户的常用操作接口；Contacts 类则封装了对联系人的增、删操作接口。这些操作可供测试用例直接调用。

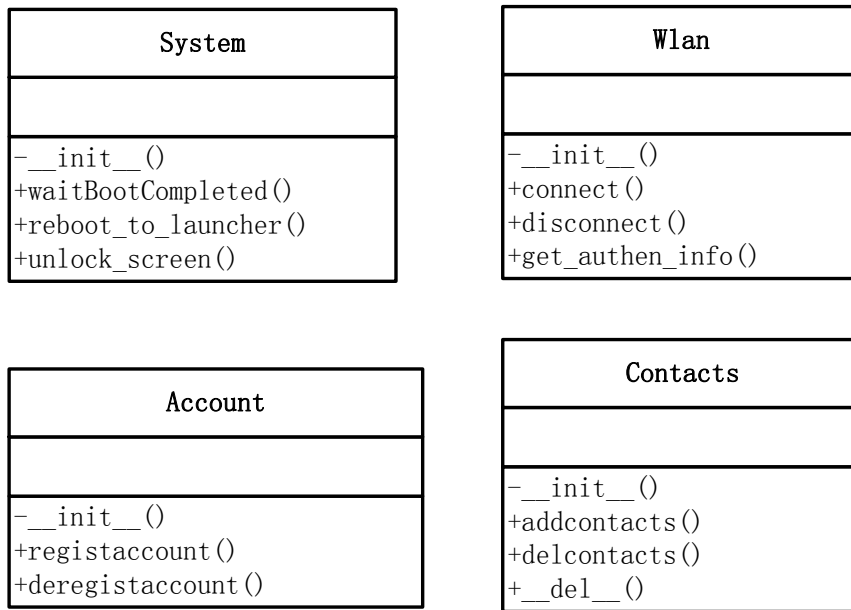


图 3-5 常用操作封装模块类图

类图中各个类以及类中各个方法的描述如表 3-4 所示。

表 3-4 常用操作封装模块相关类及方法描述

类	方法	作用
System		Android 系统操作常用操作类
	<code>__init__()</code>	初始化函数
	<code>waitBootCompleted()</code>	等待系统启动完全的事件函数
	<code>reboot_to_launcher()</code>	重新启动系统到启动器事件函数
	<code>unlock_screen()</code>	将屏幕解锁
Wlan		无线网络常用操作类
	<code>__init__()</code>	初始化函数
	<code>connect ()</code>	链接无线网络的事件函数
	<code>disconnect()</code>	断开无线网络的事件函数
	<code>get_authen_info()</code>	获取卡信息的事件函数
Account		用户账户常用操作类

表 3-4(续表)

类	方法	作用
Contacts	<code>__init__()</code>	初始化函数
	<code>registaccount()</code>	注册用户账户的事件函数
	<code>deregistaccount()</code>	注销用户账户的事件函数
		用户联系人常用操作类
	<code>__init__()</code>	初始化函数
	<code>addcontacts()</code>	添加联系人的事件函数
	<code>delcontacts()</code>	删除联系人的事件函数
	<code>__del__()</code>	清空联系人信息的事件函数

3.6 测试结果输出模块的设计

测试结果输出模块主要包含两个功能，即测试结果数据的规范化输出功能以及测试用例运行 log 数据的输出功能。

设计测试结果数据规范化输出功能是为了符合外部测试框架规定的 log 格式，以便生成测试报告。设计输出测试用例运行 log 数据是提供给开发人员，使开发人员能够了解到测试用例的整个执行状况。在设计此处时由于需要解决测试结果与执行 log 容易混在一起的问题，本模块将测试结果数据输出以 `print` 方法输出，而测试用例执行 log 日志是通过调用 Python 的 `logging` 模块内的方法进行输出。

但不论是用 `logger` 输出的，还是用 `print` 输出的，最后都会输出到系统的标准输出中。而且两者是混在一起的。外部测试框架获取测试结果是用正则表达式匹配的，在它匹配内容的时候，由于我们用 `logger` 输出的 log，都是以时间+模块名开头的，这些 log 由于不符合正则，所以 外部测试框架不会对这些 log 进行处理。而我们在 `fomatter` 中用 `print` 输出的 log，由于是按照外部测试框架规定的 log 格式输出的，所以，可以被正则表达式匹配，当成测试结果进行处理。具体类图如图 3-6 所示。

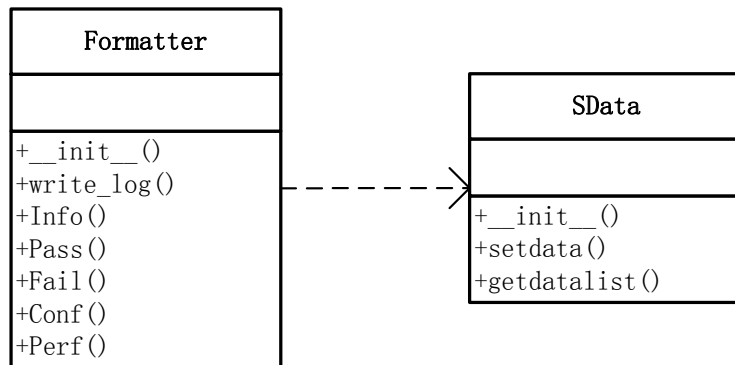


图 3-6 测试结果输出模块类图

类图中各个类以及类中各个方法的描述如表 3-5 所示。

表 3-5 测试结果输出模块相关类及方法描述

类	方法	作用
Formatter		Android 系统操作常用操作类
	__init__()	初始化函数
	write_log()	输出 log 的事件函数
	Info()	基本信息 log 格式输出的事件函数
	Pass()	执行成功 log 格式输出的事件函数
	Fail()	执行失败 log 格式输出的事件函数
	Conf()	提示跳过该测试的事件函数
SData	Perf ()	执行性能 log 格式的事件函数
	__init__()	初始化函数
	setdata()	写 log 的事件函数
	getdatalist()	读取 log 数据的事件函数

Formatter()类的 write_log()为输出 log 方法，其它的函数均为可供选择的一种 log 输出格式。这几种格式都是将 log 按照外部测试框架规定的规范进行处理，针对不同的运行结果，将 log 以外部测试框架能够解析格式输出，

用来直接生成测试报告。

规定测试结果的输出格式主要目的是为了在 web 上显示最终的测试结果，让人一看到 web 上的测试结果，就能看出自己想要了解的测试用例运行的总体情况，为了达到这一要求，系统定义了一个 log 输出的统一标准，外部测试框架对于 log 的解析也是根据这个标准来检索测试结果，根据这些结果，来画图或者生成图表。

根据测试结果的特点，系统对测试结果定义了以下几类标准输出格式：

(1) PASS / FAIL：标志 case 是否成功运行

例：TestCaseName 1 PASS : Test 1 passed

TestCaseName FAIL : Error occurred

(2) INFO：case 的运行信息，这些 log 会显示在测试报告中，而用 logger 输出的 log 由于没有添加 INFO 这个表示，不会显示在测试报告中 Test information, free formed.

(3) CONF：标志本条测试不具备测试条件，test 被 skip 掉了。

例：TestCaseName 1 CONF : Test 1 skipped

(4) PERF：主要输出性能测试的结果，即 performance 的缩写

例：Test1 1 PERF : Test1-CursorUp(s) 0.341

Test2 2 PERF : Music-CursorDown(s) avg=0.412, min=0.400, max=0.420, n=5

(5) SDATA /DATA：主要用于测试结果需要以表格形式显示的 case 的结果输出。

例：Test1 1 SDATA : Page: 1, Duration: 17 s, Battery: 100 %, Capacity: 190 mAh

Test2 2 SDATA : free_mem: 100 MB, rss: 34356 B, cpu_load: 30 %, booklet: Reader

Test1 1 DATA : Test1-CursorUp(s) 0.341

Test2 2 DATA : Music-CursorDown(s) [0.4, 0.5, 0.3, 0.35]

3.7 测试用例基类模块的设计

测试用例是自动化测试系统的数据基础，为了配合测试系统进行全面的测试工作，大量的自动化测试用例陆续被开发出来。将测试用例按待测设备需要被测试的功能及性能进行合理的分类显得尤为重要。图 3-7 为测试用例模块图。

根据合理的分配，测试用例模块主要分为应用性能测试用例、通用性测试用例、图形基准测试用例、平台函数测试用例、电源消耗测试用例、以及系统性能测试用例一共六类。每个类别中有一些测试脚本，这些测试脚本通过自动控制运行该方面的测试工具，以得到所需的测试结果数据。

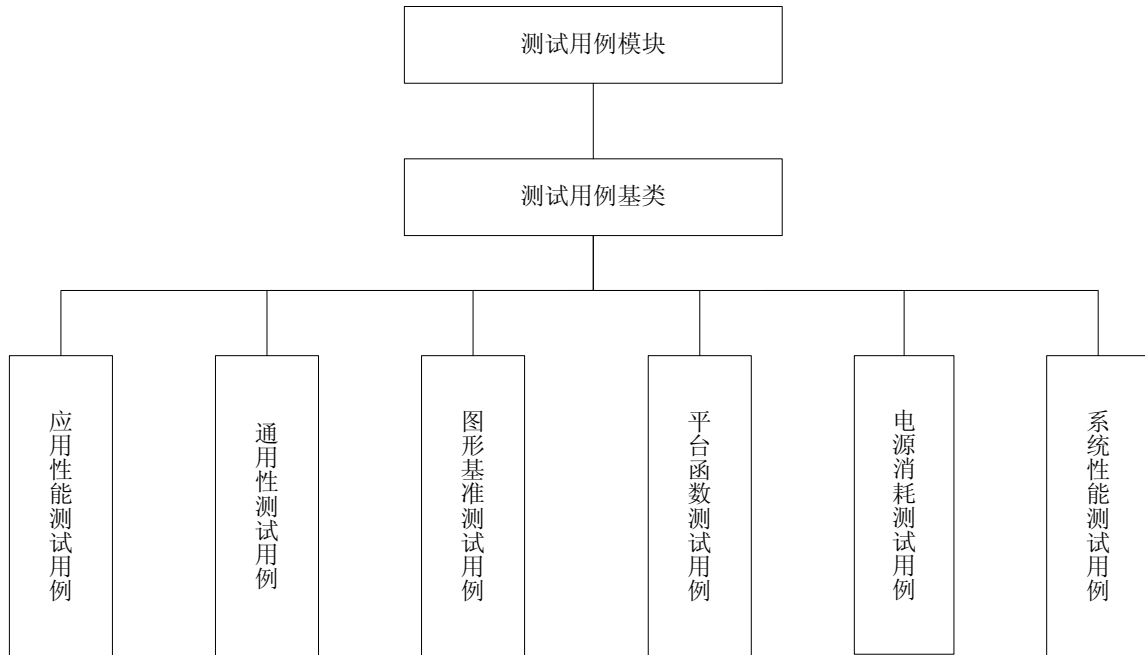


图 3-7 测试用例模块图

测试用例基类模块是所有测试用例的基类，它定义了测试用例各运行阶段的接口函数，由测试用例加载模块来调用。根据测试流程以及测试工作经常用到的接口，测试用例基类定义了如下接口函数。部分代码如下。

```

class Tc_base():
    def __init__(self, name):
        """
        Base initialization
        """
        self._CWD=""
        self._PREBUILTDIR=""
        self._HOOK_INSTALL = {}
        self._HOOK_INSTALL_APK = {}
        self._device= tc_device.Device()
        self._adb = tc_adb.ADB()
        self._formatter = tc_formatter.Formatter()
    
```

```
self._logcat = tc_logcat.Logcat()
```

```
def setup(self):
```

```
    """
```

```
    """
```

```
def run(self):
```

```
    """
```

```
    """
```

```
def cleanup(self):
```

```
    """
```

```
    """
```

3.8 本章小结

本章介绍了软件自动化测试系统的设计。根据第 2 章软件自动化测试的需求分析结果，对自动化测试系统进行了总体的设计。首先明确了整个自动化测试的架构以及自动化测试具体执行流程。接着对手机软件自动化测试进行了功能分解，分解成五大模块——测试用例加载模块、ADB 接口模块、测试结果输出模块、常用操作封装模块。最后分别对上述五大模块进行了概要设计、详细设计等。

第 4 章 Android 平台的自动化测试系统的实现

4.1 测试用例加载模块的实现

根据在第 3 章中简介的测试用例加载模块的详细设计，以下将介绍主要方法的实现。

测试用例加载模块的方法实现流程图如图 4-1 所示。

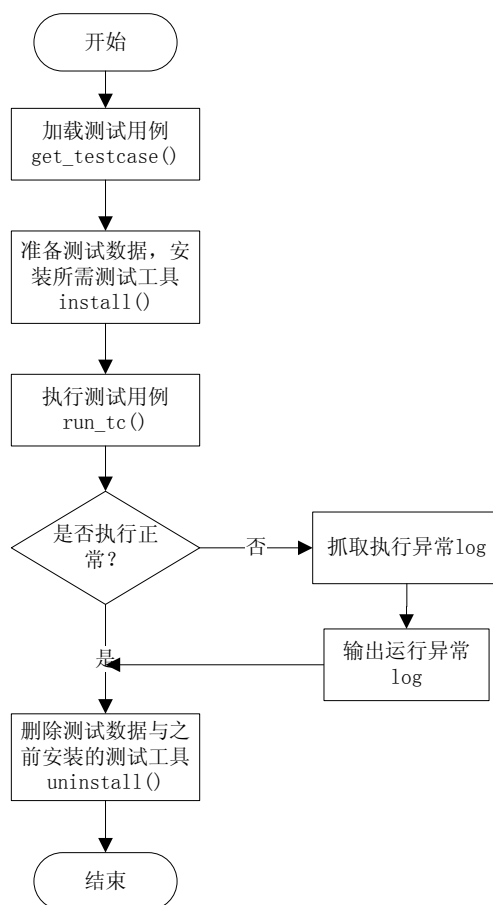


图 4-1 测试用例加载模块实现流程图

测试用例加载模块的具体实现过程如下：

(1)调用 `get_testcase()`加载测试用例。

(2)调用 `installer` 类的 `install()`方法为执行测试做准备工作,加载测试数据,并安装需要的 `apk` 文件与第三方测试工具。

(3)调用 run_tc()函数执行测试用例，依次执行测试用例的 setup()、run()、cleanup()三个基本步骤。在此过程中任何一个步骤出现异常时都会通过 logger.error()方法抓取运行异常信息。

(4)在测试用例执行完毕之后调用 installer 类中的 uninstall()方法将已安装的 apk 文件和测试工具删除，并清除无用的测试数据。

(5)返回运行异常测试结果。

4.1.1 模拟外部测试框架调用功能的实现

根据设计，测试用例加载模块可以模拟外部测试框架对测试用例控制模块的调用动作，进行测试活动。方便调试以及可以在本地进行测试工作。一下为该功能的实现方法。模拟外部测试框架调用以及测试用例模板创建功能的流程图如图 4-2 所示：

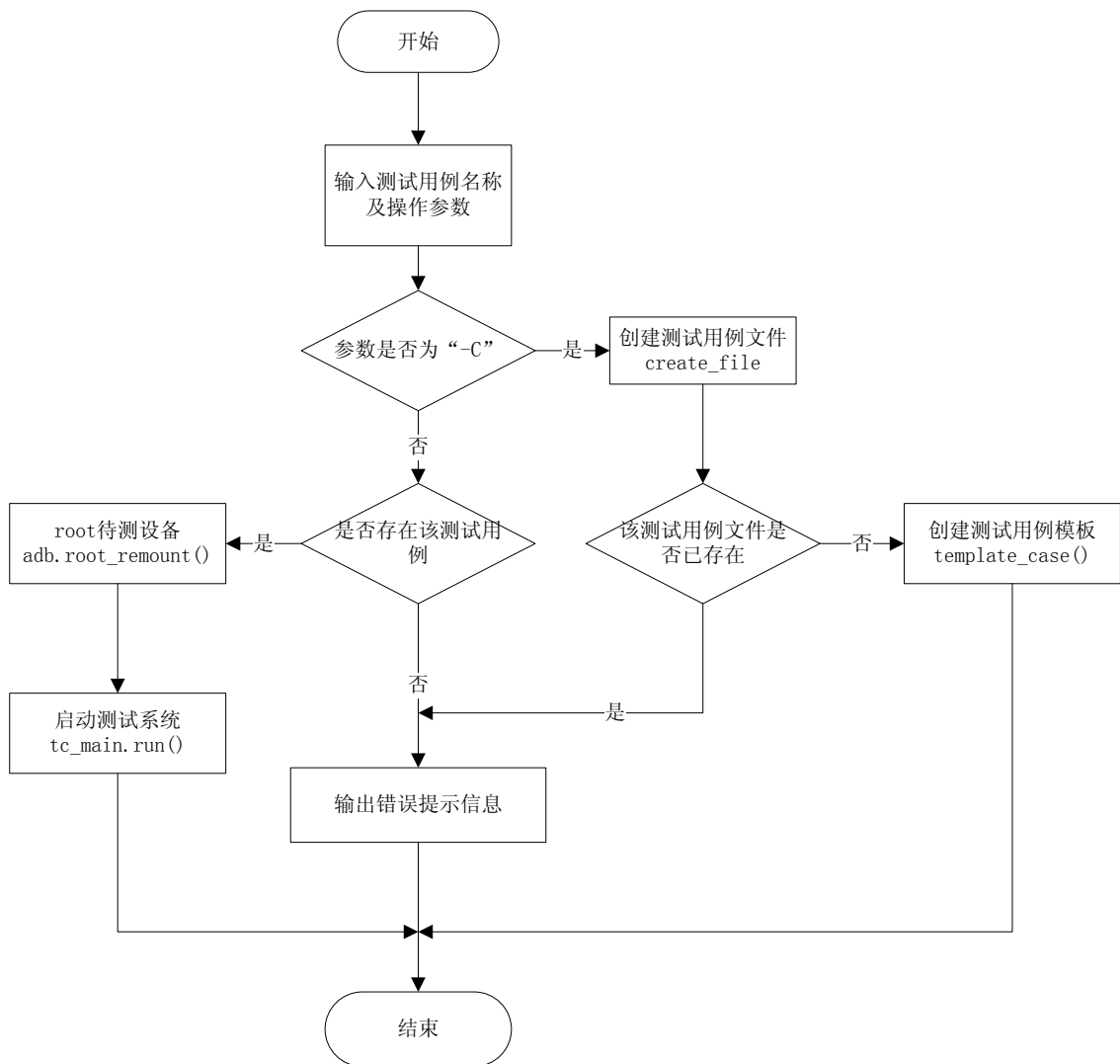


图 4-2 模拟调用功能实现流程图

模拟外部测试框架调用以及测试用例模板创建功能的具体实现过程如下：

(1)提示用户输入测试用例名称及功能参数。

(2)如果用户输入了测试用例名称，并没有参数，则判断是否存在该测试用例，若不存在该测试用例，则输出错误提示信息“Testcase do not exist!”。若存在该测试用例，则先调用 `adb.rootremount()`将待测设备 root，然后调用 `tc_main.run()`启动自动化测试系统。

4.1.2 创建测试用例模板功能的实现

在模拟外部测试框架调用功能的实现中，如果用户输入了测试用例的名称和参数“-C”，则调用 `create_file()`方法尝试在指定目录下创建该文件，若该文件已存在，则输出错误提示信息“[%s](指相应的测试用例名称) is already exist!”。若不存在该文件，则创建文件，并调用 `template_case()`方法创建一个测试用例模板。

4.2 测试状态运行监控模块的实现

测试状态运行监控模块是提高测试用例执行稳定性的重要模块。它负责对测试用例执行过程中异常情况发生的监控，根据第 3 章的详细设计，它包括 ADB 工具运行监控和待测 Android 设备监控两部分。

4.2.1 ADB 工具运行监控功能的实现

ADB 运行状态监控功能是通过 Watchdog 这个类实现的。ADB 运行状态监控的实现逻辑图如图 4-3 所示。

具体实现步骤如下：

1、当 cmd 命令传入时，`Executor.run()`首先判断传入的命令是否是 ADB 命令。如果不是 ADB 命令则开启一条线程执行该命令。

2、当确定传入的命令为 ADB 命令后，`Executor.run()`调用 `Watchdog.start()`，同时 `Executor.run()`开启一条线程执行 ADB 命令。

3、`Watchdog.start()`调用 `Watchdog.check()`方法也开启一条线程，每隔 2 秒执行一次“adb shell pwd”命令来判断 ADB 工具是否有响应。

4、如果 ADB 工具没有发生响应超时异常，`Executor.run()`在执行完当前 ADB 命令之后，便会调用 `Watchdog.stop()`杀死该条 ADB 命令相应的监控进

程。

5、如果 Watchdog.check()监测到 ADB 工具响应超时,便调用_timeout_cb()方法杀死 ADB 命令执行线程,并通过 usbreset 工具重新启动 ADB Server,然后重新从步骤 2 执行此条命令。

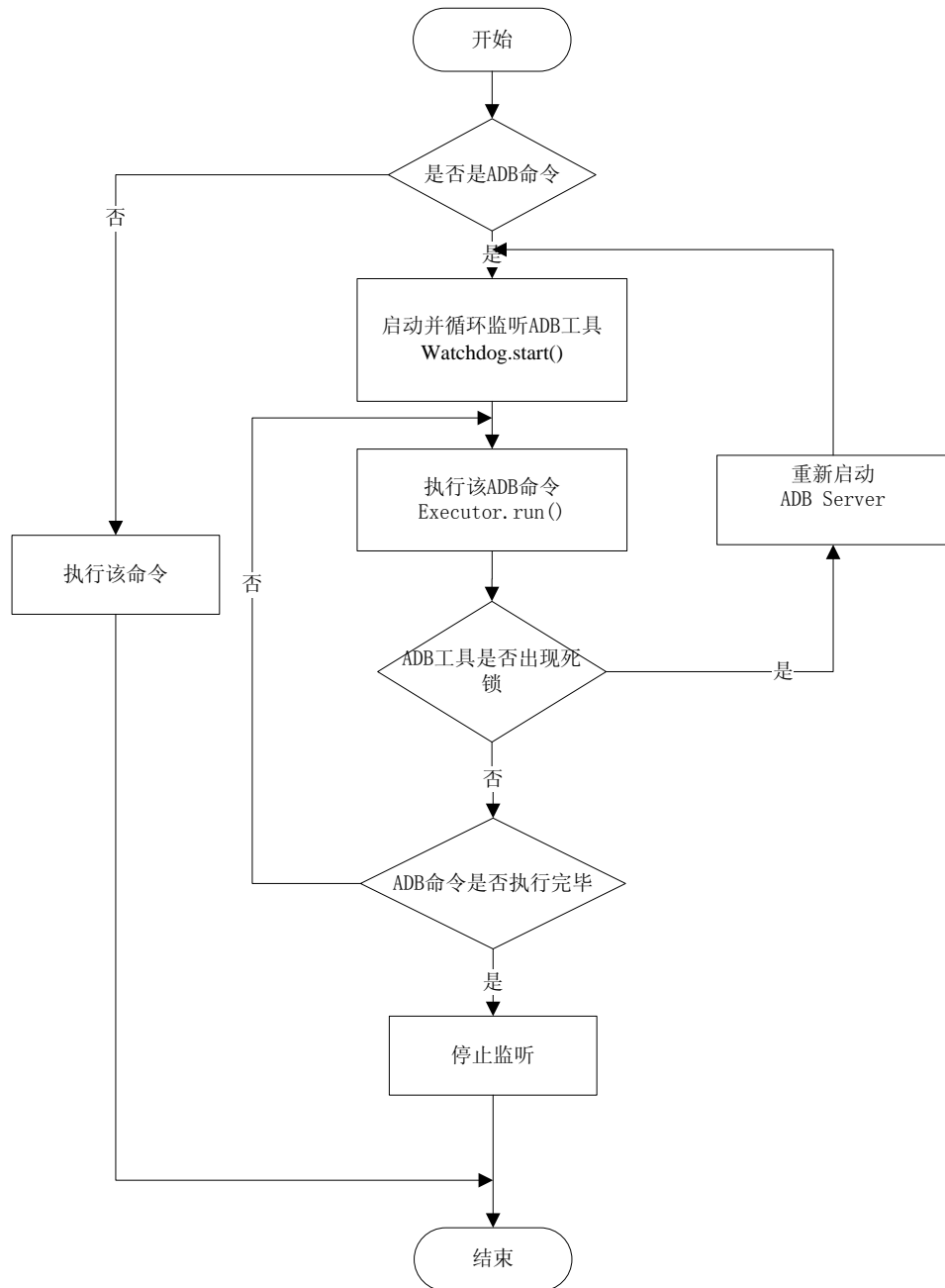


图 4-3 ADB 运行状态监控实现逻辑图

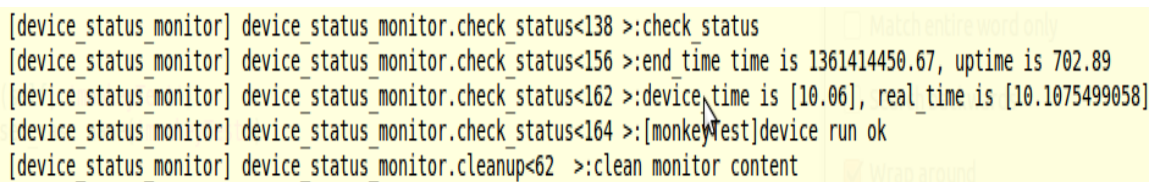
4.2.2 待测设备运行监控功能的实现

由于需要判断待测设备在测试用例执行过程当中是否正常运行，有无出现例如意外重启等异常。本模块设计了两种策略来解决这个问题。由自动化测试系统的工作流程可知，系统在启动后会调用 `Device_status_monitor` 类的 `monitor()` 方法，`monitor()` 方法会判断具体的测试脚本中所选择的策略是时间策略或计数策略来选择不同的方法来监控待测设备。

时间策略的原理是在系统启动时监视模块首先读取当时的 PC 系统时间 $T1$ ，与此同时通过“`cat /proc/uptime`”命令获得该路径下待测设备的运行时间 $t1$ 。在测试结束时监视模块再次读取当时的系统时间 $T2$ ，同时也获取待测设备的运行时间 $t2$ 。比较 $(T2-T1)$ 与 $(t2-t1)$ 的值的的大小。如果设备发生异常时，系统运行时间计数会暂停数秒。所以当两个值相差比较大时，可以判定设备在测试用例执行过程中发生了异常重启等问题。根据实验数据得知，一般情况下两个值的差值如果小于 2 秒可以认为设备运行正常。

有个别设备在记录系统运行时间时误差过大，这样很难判断设备是否发生了异常重启。为了解决这个问题设计了系统启动计数策略。计数策略的原理是在系统启动时和测试结束后通过“`cat /proc/boot_count`”获取该路径下待测设备的启动次数。在测试执行过程中，如果系统需要正常的重启都是调用“`adb reboot`”命令来执行。在重启之后会重新读取启动次数。所以只有当异常重启发生时才会出现启动数值不同的情况。通过比较这两个数值是否相等即可确定系统在执行过程中是否发生意外重启。这两种策略可以根据不同需要由测试脚本通过调用 `setpolicy()` 方法自由选择。

如图 4-4 为测试状态运行模块执行测试用例后生成的输出结果截图。



```
[device_status_monitor] device_status_monitor.check_status<138 >:check status
[device_status_monitor] device_status_monitor.check_status<156 >:end time time is 1361414450.67, uptime is 702.89
[device_status_monitor] device_status_monitor.check_status<162 >:device time is [10.06], real_time is [10.1075499058]
[device_status_monitor] device_status_monitor.check_status<164 >:[monkeyTest]device run ok
[device_status_monitor] device_status_monitor.cleanup<62 >:clean monitor content
```

图 4-4 待测设备监测结果截图

4.3 ADB 接口模块的实现

ADB 接口主要是通过 `Executor` 类中的 `run()` 方法来实现的。ADB 操作被封装起来可供测试用例直接调用。它的功能实现逻辑图如图 4-5 所示。

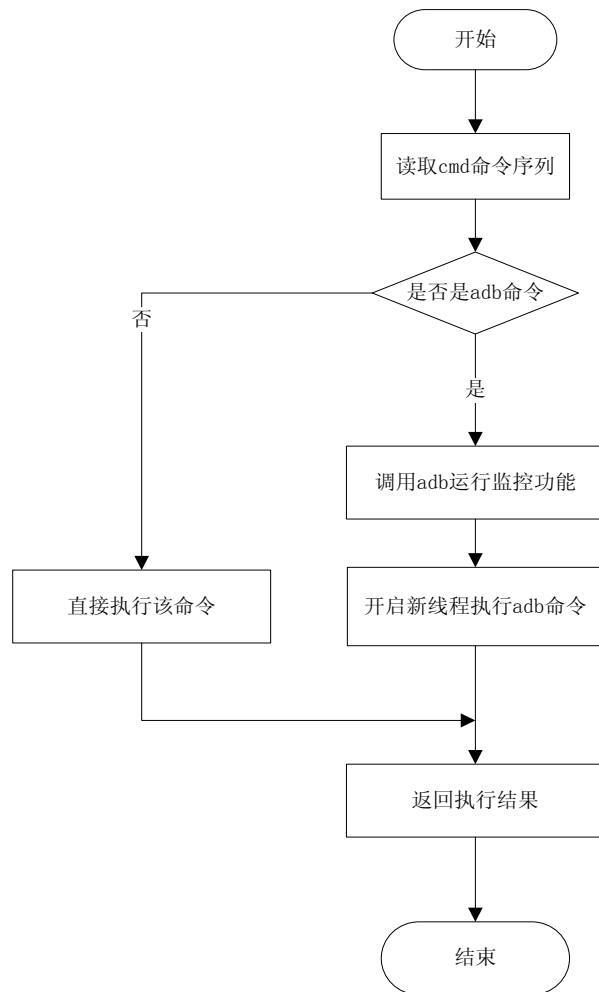


图 4-5 ADB 接口模块运行逻辑图

ADB 命令执行功能的具体实现过程如下：

- 1) Executor.run()读取一条 cmd 命令。
- 2) 判断获得的命令是否为 ADB 命令，如果不是 ADB 命令的话直接执行该条命令。
- 3) 如果读入的命令为 ADB 命令，调用 Watchdog.start()启动 ADB 运行监控功能。并新创建一个进程对 ADB 运行状态进行监听。在 ADB 命令执行完毕后关闭该进程
- 4) Executor.run()开启一条进程程执行 ADB 命令。执行完毕后关闭该进程。
- 5) 返回运行结果。

4.4 常用操作封装模块的实现

常用操作封装模块的实现是以按照模拟人的点击动作，将一系列的 ADB 命令序列以固定顺序封装起来完成。以注册账户操作举例，该 ADB 操作动作序列如下所示：

```
logger.debug('deregist account')
    if self._device.is_product() == False:
        logger.debug('The device is not product.')
        return
    self._adb.shell("am start -n com. settings/.SettingsHome")
    time.sleep(10)
    # input center keyevent
    self._adb.shell("input keyevent 23")
    time.sleep(15)
    # input center keyevent
    self._adb.shell("input keyevent 23")
    time.sleep(5)
    self._adb.shell("input keyevent 23")
    time.sleep(5)
    time.sleep(60)
    self._adb.shell("am force-stop com.amazon.kindle.otter.settings")
```

该命令通过 Android 中的 ADB 驱动命令自动在设备上模拟点击注册账户所需要点击的坐标位置，然后在操作结束后关闭该进程。

4.5 测试结果输出模块的实现

测试用例运行 log 日志的输出功能是通过调用 Python 的 logging.getLogger()方法，并调用 logging.formatter()来整理系统 log 输出格式。当 Formatter()类的 write_log()方法工作时，优先测试结果输出。

该方法实现的主要代码如下图 4-6 所示：

```

def write_log(self, type, message):
    logger.debug("Format Result:" + self.Head_Format %(self.casename, self.instance, type, message))
    print self.Head_Format%(self.casename, self.instance, type, message)
    return RES_SUCCESS

def Info(self, instance, msg):
    '''
    Test information, free formed.
    '''
    self.instance = str(instance)
    return self.write_log(INFO, msg)

def Pass(self, instance, msg):
    '''
    Test passed.
    Ex. TestCaseName 1 PASS : Test 1 passed
    '''
    self.instance = str(instance)
    return self.write_log(PASS, msg)

def Fail(self, instance, msg):
    '''
    Test passed.
    Ex. TestCaseName FAIL : Error occurred
    '''
    self.instance = str(instance)
    return self.write_log(FAIL, msg)

def Perf(self, instance, component, test, unit, values):
    '''
    Performance result, for reporting final performance measurements.
    Format: TestCaseName instance PERF : component-test(unit) values
    Ex.
    Test1 1 PERF : Test1-CursorUp(s) 0.341
    Test2 2 PERF : Music-CursorDown(s) avg=0.412, min=0.400, max=0.420, n=5
    '''
    self.instance = str(instance)
    msg = '%s-%s(%s)'%(component, test, unit)
    if isinstance(values, dict):
        m = ""
        for key, value in values.items():
            m += '%s=%s, '%(key, value)
        msg += m[:-2]
    else:
        msg += str(values)

    return self.write_log(PERF, msg)

```

图 4-6 测试结果输出的部分实现代码

4.6 本章小结

本章介绍了手机软件自动化测试的实现方案，根据第 3 章手机软件自动化测试系统的设计的结果，对手机软件自动化测试系统进行了实现。分别描述了五大模块——测试用例加载模块、ADB 接口模块、测试结果输出模块、常用操作封装模块的实现方案，即这五大模块中主要方法的程序流程。

第 5 章 自动化测试系统的测试

软件测试是在程序发布前通过反复运行应用程序来显现出程序中存在的 Bug 为目的过程，测试的主要目的是尽早发现系统设计中的缺陷、系统实现与需求偏离的地方，及时找到代码实现中存在的问题和程序内部 bug，确保系统尽可能的功能完整、运行正确稳定并且具有一定并发性和可靠性。软件工程则是通过将软件的开发流程规范化，提高软件的质量，从而满足用户对软件的需求的技术。因此在通过进行软件测试发现了程序的错误之后还必须分析错误原因并解决。软件测试的结果则是分析软件可靠性的重要依据。

本小节将从功能测试和性能测试两个方面给出一些测试用例和实际测试结果，以便于我们对系统的设计和开发成果做出有效的评估。

本课题的测试工作要实现以下目标：

- 1) 确保系统完成了在系统总体需求说明中的所有功能
- 2) 确保各个功能点满足性能需求
- 3) 确保系统的可扩展和已操作
- 4) 整个过程是准确的

5.1 测试方法

该系统功能是实现 Android 平台设备的自动化测试，由于本系统的大多数功能都是在后台运行，并不是可以具体实际体现的功能，所以为此系统功能模块的测试为在 Android 设备上验证被测试脚本的自动化测试是否可正常执行，如可正常执行即意味着系统功能可正常运行如自动执行测试用例，监控测试运行状态过程等。主要通过人工方式验证是否成功执行了测试用例。封装的接口是否能够正常工作。

5.1.1 测试脚本的选取和执行

根据选择，将 Qundant test 脚本选作本系统的测试用例。

首先通过 `./simulator.py -p test quandant` 命令启动模拟外部测试框架调用功能执行该测试用例，测试脚本执行比较长，部分关键运行截图如图 5-1 和 5-2 所示：

```
kangkij@P-D2-kangkij:~/work/base/8960/amazon/aosta/scripts/katf_lib$ ./simulator.py -p test Quadrant
simulator root device
2013-05-30 17:29:58,381:381 [device_status_monitor] device_status_monitor.start_monitor<117 >:start_monitor
2013-05-30 17:29:58,382:382 [device_status_monitor] device_status_monitor.start_monitor<119 >:start_monitor casename is [Quadrant]
2013-05-30 17:29:59,384:384 [device_status_monitor] device_status_monitor.start_monitor<133 >:start time is 1369906198.38, uptime is 279.45
2013-05-30 17:29:59,403:403 [Quadrant] Quadrant._init_<38 >:test times is 3
2013-05-30 17:29:59,403:403 [tc_loader] tc_loader.run_tc<63 >:HOOK INSTALL = {}
2013-05-30 17:29:59,403:403 [tc_loader] tc_loader.run_tc<64 >:HOOK INSTALL APK = {'thirdparty/Quadrant.apk': 'com.aurorasoftworks.quadrant.ui.professional'}
2013-05-30 17:29:59,403:403 [tc_loader] tc_loader.run_tc<70 >:----- BEGIN RUNNING <Quadrant> -----
2013-05-30 17:29:59,403:403 [installer] installer.install<24 >:install binary
2013-05-30 17:30:02,407:407 [util.tc_adb] tc_adb.installapk<124 >:4333 KB/s (944622 bytes in 0.212s)

2013-05-30 17:30:02,408:408 [util.tc_adb] tc_adb.installapk<124 >: pkg: /data/local/tmp/Quadrant.apk

2013-05-30 17:30:02,408:408 [Quadrant] Quadrant.setup<43 >:setup
2013-05-30 17:30:07,422:422 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:30:08,551:551 [Quadrant] Quadrant._prepare<70 >:QuadrantProfessionalActivity start ok
2013-05-30 17:30:08,552:552 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:30:08,552:552 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13052
2013-05-30 17:30:17,562:562 [Quadrant] Quadrant.run<51 >:run [0]
2013-05-30 17:30:19,565:565 [util.tc_adb] tc_adb.say<219 >:CMD Result: begin execute cmd:adb logcat -c
2013-05-30 17:30:19,621:621 [util.tc_adb] tc_adb.say<219 >:CMD Result: end execute cmd:adb logcat -c->0
2013-05-30 17:30:22,570:570 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:30:23,577:577 [Quadrant] Quadrant._startquadranttest<98 >:BenchmarkExecutionActivity start ok
2013-05-30 17:30:23,578:578 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:30:23,578:578 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13094
2013-05-30 17:30:25,580:580 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:07,665:665 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:07,666:666 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13096
2013-05-30 17:31:09,667:667 [Quadrant] Quadrant.run<51 >:run [1]
2013-05-30 17:31:12,671:671 [util.tc_adb] tc_adb.say<219 >:CMD Result: begin execute cmd:adb logcat -c
2013-05-30 17:31:12,720:720 [util.tc_adb] tc_adb.say<219 >:CMD Result: end execute cmd:adb logcat -c->0
2013-05-30 17:31:15,674:674 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:16,681:681 [Quadrant] Quadrant._startquadranttest<98 >:BenchmarkExecutionActivity start ok
2013-05-30 17:31:16,682:682 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:16,682:682 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13115
2013-05-30 17:31:18,683:683 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:59,769:769 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:59,769:769 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13117
2013-05-30 17:32:01,771:771 [Quadrant] Quadrant.run<51 >:run [2]
2013-05-30 17:32:04,775:775 [util.tc_adb] tc_adb.say<219 >:CMD Result: begin execute cmd:adb logcat -c
2013-05-30 17:32:04,820:820 [util.tc_adb] tc_adb.say<219 >:CMD Result: end execute cmd:adb logcat -c->0
2013-05-30 17:32:07,779:779 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
```

图 5-1 测试用例执行界面

```
2013-05-30 17:30:23,578:578 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:30:23,578:578 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13094
2013-05-30 17:30:25,580:580 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:07,665:665 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:07,666:666 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13096
2013-05-30 17:31:09,667:667 [Quadrant] Quadrant.run<51 >:run [1]
2013-05-30 17:31:12,671:671 [util.tc_adb] tc_adb.say<219 >:CMD Result: begin execute cmd:adb logcat -c
2013-05-30 17:31:12,720:720 [util.tc_adb] tc_adb.say<219 >:CMD Result: end execute cmd:adb logcat -c->0
2013-05-30 17:31:15,674:674 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:16,681:681 [Quadrant] Quadrant._startquadranttest<98 >:BenchmarkExecutionActivity start ok
2013-05-30 17:31:16,682:682 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:16,682:682 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13115
2013-05-30 17:31:18,683:683 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:31:59,769:769 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:31:59,769:769 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13117
2013-05-30 17:32:01,771:771 [Quadrant] Quadrant.run<51 >:run [2]
2013-05-30 17:32:04,775:775 [util.tc_adb] tc_adb.say<219 >:CMD Result: begin execute cmd:adb logcat -c
2013-05-30 17:32:04,820:820 [util.tc_adb] tc_adb.say<219 >:CMD Result: end execute cmd:adb logcat -c->0
2013-05-30 17:32:07,779:779 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:32:08,786:786 [Quadrant] Quadrant._startquadranttest<98 >:BenchmarkExecutionActivity start ok
2013-05-30 17:32:08,787:787 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:32:08,787:787 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13139
2013-05-30 17:32:10,789:789 [util.tc_logcat] tc_logcat.start<54 >:logcat start cmd: adb logcat
2013-05-30 17:32:51,866:866 [util.tc_logcat] tc_logcat.stop<67 >:logcat stop cmd: adb logcat
2013-05-30 17:32:51,867:867 [util.tc_logcat] tc_logcat.stop<69 >:command pid: 13141
Quadrant 0 PERF : Total Score-Total Score(points) 1455
Quadrant 1 PERF : CPU Score-CPU Score(points) 3037
Quadrant 2 PERF : IO Score-IO Score(points) 1648
Quadrant 3 PERF : 2D fractal-2D fractal(fps) 5.18
Quadrant 4 PERF : 3D Corridor-3D Corridor(fps) 22.75
Quadrant 5 PERF : 3D Planet-3D Planet(fps) 26.98
Quadrant 6 PERF : 3D DNA-3D DNA(fps) 22.05
Quadrant 7 PASS : QuadrantUI run successfully
2013-05-30 17:32:53,869:869 [installer] installer.uninstall<44 >:uninstall binary
2013-05-30 17:32:55,871:871 [installer] installer.uninstall<55 >:uninstall package :com.aurorasoftworks.quadrant.ui.professional
2013-05-30 17:32:55,872:872 [device_status_monitor] device_status_monitor.check_status<138 >:check status
2013-05-30 17:32:56,873:873 [device_status_monitor] device_status_monitor.check_status<156 >:end time time is 1369906376.87, uptime is 456.94
2013-05-30 17:32:56,874:874 [device_status_monitor] device_status_monitor.check_status<162 >:device time is [177.49], real_time is [178.491279125]
2013-05-30 17:32:56,874:874 [device_status_monitor] device_status_monitor.check_status<164 >:[Quadrant]device run ok
2013-05-30 17:32:56,874:874 [device_status_monitor] device_status_monitor.cleanup<62 >:clean monitor content
2013-05-30 17:32:56,874:874 [tc_loader] tc_loader.run_tc<139 >:----- END RUNNING <Quadrant>:177471(ms)-----
The return value of Quadrant is: 0
```

图 5-2 测试用例执行通过界面

从运行结果中可以看出整个测试过程顺利执行，将在下面的内容中分析测试执行结果，并解释自动化测试系统的每个功能是如何体现出的。

5.1.2 测试运行的结果分析

根据之前测试脚本的运行结果，我们可以分析出自动化测试系统的运行状况。

(1) 从运行 log 可以看到，测试用例在运行后由测试用例加载模块控制执行过程，调用 ADB 接口模块的 `tc_adb.Installapk` 方法将 apk 文件安装进待测设备。

(2) 系统在正常启动后，首先调用 `logging_client.start()` 开启抓取系统运行 log 的功能，加载完 log 输出模块之后控制模块会自动调用 `run_tc()` 加载测试用例。`run_tc()` 会完成加载测试用例，并将测试用例进行初始化，自动将测试所需要的数据 push 进设备，并调用 `installer()` 完成必需 apk 文件和所需测试工具的安装。

(3) 之后系统调用 `device_status_monitor.monitor()` 开启对待测设备的监控，之后便调用 `tc_run()` 执行 `qundant` 工具执行测试动作，在操作的同时，控制模块会调用 `Watchdog.start()` 开启 ADB 监视功能，监视 ADB 运行状态。

(4) 测试用例运行完成之后，调用 `tc_formatter()` 输出测试用例的测试结果，运行结果如图 5-3 所示。

```
Quadrant 0 PERF : Total_Score-Total_Score(points) 1455
Quadrant 1 PERF : CPU_Score-CPU_Score(points) 3037
Quadrant 2 PERF : IO_Score-IO_Score(points) 1648
Quadrant 3 PERF : 2D_fractal-2D_fractal(fps) 5.18
Quadrant 4 PERF : 3D_Corridor-3D_Corridor(fps) 22.75
Quadrant 5 PERF : 3D_Planet-3D_Planet(fps) 26.98
Quadrant 6 PERF : 3D_DNA-3D_DNA(fps) 22.05
Quadrant 7 PASS : QuadrantUI run successfully
```

图 5-3 测试用例执行通过界面

(5) 测试用例加载模块在测试用例执行完毕之后调用 `installer` 类中的 `uninstall()` 方法将已安装的 apk 文件和测试工具删除，并清除无用的测试数据。

通过以上测试执行流程的分析，我们可以发现，整个自动化测试正常执行，监控功能以及测试结果输出功能也运行完美。在测试执行过程中也完成了对部分 ADB 接口以及常用操作封装模块的调用，以上模块全部运行正常。

5.2 其它功能测试

系统功能测试的过程中，即采用了黑盒测试的方法，又采用了白盒测试的方法。黑盒测试的方法一般是通过将程序的运行结果与预期做比较，观察程序运行是否能够达到之前预期的功能需求。这种方法是把测试对象看作是一个黑

盒子，使测试者站在用户的视角，在不分析程序的运行原理，更不必了解其内部的逻辑结构的情况下，只需要执行软件，比较其运行结果即可。白盒测试则是对软件的整个程序流程进行跟踪。白盒测试把测试对象看作一个打开的盒子，盒子的内部对软件测试者可见，允许软件测试者根据程序的逻辑关系对程序的逻辑路径进行测试。

5.2.1 ADB 接口模块的测试

ADB 接口模块主要是对各种 ADB 命令的封装，因为它们是接口的形式存在的，所以需要每个接口进行单元测试。测试用例如表 5-1 所示：

表 5-1 测试用例表

接口	动作	期望结果
push()	向待测设备装载数据	成功向待测设备装载数据
pull()	从待测设备取数据	成功从待测设备取数据
shell()	adb Shell 命令接口	成功调用 adb Shell 命令
wait_for_device()	等待待测设备响应	成功等待待测设备响应
reboot ()	重启待测设备	成功重启待测设备
installapk()	在待测设备上安装 apk 文件	成功在待测设备上安装 apk 文件
uninstallapk()	卸载待测设备中已安装的 apk 文件	成功卸载待测设备中已安装的 apk 文件
exist()	检查待测设备中文件是否存在	成功检查待测设备中文件是否存在
makedirs()	新建文件夹	成功在待测设备上新建文件夹
rm()	删除待测设备中数据	成功删除待测设备中数据
copy()	复制待测设备中数据	成功复制待测设备中数据
move()	移动待测设备中数据	成功移动待测设备中数据
root_remount()	使待测设备获得 root 权限	成功使待测设备获得 root 权限

经过对每个 ADB 接口进行单元测试，此部分验证全部通过，所有接口有效好用，能达到预期目标，与期望结果完全一致。

5.2.2 常用操作封装模块测试

常用操作封装模块与 ADB 接口模块类似，主要是对各种常用操作的命令序列的封装，它们也是接口的形式存在的，所以需要每个接口进行单元测试。测试用例如表 5-2 所示：

表 5-2 测试用例表

接口	动作	期望结果
waitBootCompleted()	等待系统启动完全	成功等待系统启动完全
reboot_to_launcher()	重新启动系统到启动器	成功重新启动系统到启动器
unlock_screen()	将屏幕解锁	成功将屏幕解锁
connect ()	链接无线网络	成功链接无线网络
disconnect()	断开无线网络	成功断开无线网络
get_authen_info()	获取网卡信息	成功获取网卡信息
registaccount()	注册用户账户	成功注册用户账户
deregistaccount()	注销用户账户	成功注销用户账户
addcontacts()	添加联系人	成功添加联系人
delcontacts()	删除联系人	成功删除联系人
__del__()	清空联系人信息	成功清空联系人信息

经过对每个常用操作接口进行单元测试，此部分验证全部通过，所有接口有效好用，能达到预期目标，与期望结果完全一致。

5.3 非功能测试

1) 性能测试

性能测试是一种更适合于软件性能需求的测试。并非所有的软件系统都有明确的性能规格说明，但每个系统都有隐含的性能需求。性能缺陷通常表明软件存在设计缺陷，引起系统的性能降级。性能测试的目标是识别性能瓶颈、进行性能比较和评价，性能测试可用来确定处理程序不存在明显的降级，但一般不用于软件优化。

针对本系统的性能测试主要是对测试系统稳定性、高效性及准确性的测试，系用来验证系统的稳定性，高效性等非功能需求的实现程度。表 5-3 给出了

本系统性能测试的主要测试点和测试结果。

表 5-3 系统性能测试汇总

测试点	测试方法	测试结果
稳定性	在测试用例运行过程中人为将设备断开重连	可以从断点重新运行并记录错误信息
高效性	将 ADB 工具人为关闭	可以重新启动 ADB Server，大大节省测试时间
准确性	进行多次测试动作	所有功能均能正确完成

2) 可扩展性验证

由实际工作验证，由于本系统对测试用例结构进行合理设计并对 ADB 命令与多种常用操作进行封装，大大减少了测试开发人员编写测试脚本工作量，并使测试用例的通用性和可扩展性大大增加。

5.4 测试结果分析

由功能测试以及非功能测试结果可见，自动化测试系统到达了第 2 章中的基本功能需求和非功能需求，自动化测试可以驱动测试用例运行，监控待测设备及 ADB 的运行状况，当 ADB 出现运行异常时，重启 ADB Server 并及时恢复现场，保证测试用例可以继续执行，减少再次重新运行一次测试用例所造成的时间浪费。使运行 Log 和测试用例的测试结果互不干扰。自动生成测试用例模板，方便开发人员编写新的测试用例。整体的自动化测试是实际可用并可稳定运行，实现了很大程度上的节省人力代替手工操作。

5.5 本章小结

本章主要介绍了自动化测试系统的功能测试。描述了自动化测试系统进行功能测试的方法，根据第 2 章的需求分析设计测试用例，分别对自动执行测试用例、监控测试运行状态、格式化输出测试结果、提供自动化测试系统的本地调用、生成测试脚本模板和对 ADB 命令进行封装进行了功能测试，并对自动化测试系统进行了非功能测试。最后对测试结果进行了总结。

结 论

本文主要论述 Android 平台的自动化测试系统的设计与实现的全过程。从对系统的需求分析、总体设计、详细设计到系统的实现与测试各个阶段进行了详尽的分析。整个系统的开发严格按照需求来进行，完成了如下模块的开发：测试用例加载模块、测试运行状态监控模块、测试结果输出模块、ADB 接口模块和常用操作封装模块，并得到了以下研究成果：

(1) 调查实际中的自动化测试方案，并查阅大量文献资料，分析了实际应用和前沿科研中的自动化测试系统。将较成熟和可适用的自动化测试模型引入对设备的自动化测试中。

(2) 从功能上来看完成了预计目标。实现了对测试用例的驱动，使测试用例能够自动执行，完成了进行自动化测试的过程中发生意外情况的控制。并解决了输出结果混乱问题，使测试结果能够与系统运行 log 区分出来，格式化输出。

(3) 从性能上来看与预期相吻合。自动化测试系统的测试稳定性较高，能够极大地提高测试框架的测试效率，同时减轻了测试人员的劳动强度。使用本系统后，能够实现轻松的对测试用例进行扩充和维护，极大的改善了之前测试框架存在的问题。

(4) 由于测试是运行在 Android 设备上的，在研究了多种系统实现的可行方案后，最终选择了通过 Android SDK 中的 ADB 工具来实现。并可以通过运行状态监控模块解决了 ADB 工具运行不稳定的问题。

(5) 将 ADB 命令序列和常用的操作封装起来，供测试开发人员编写测试用例时调用，极大的改善了测试用例的编写，减少了工作量并加强了测试用例的可复用性。

经过测试，证明了课题的研究满足了需求目标 and 设计目标，并且具有良好的效果。本系统已经在公司内部使用，大大提高了测试工作的效率，很大程度上节省手工劳动同时保证产品测试要求，完全满足了系统的需求。

本系统虽然满足了系统开发需求中的基本要求，但仍然存在着一一定的问题有待去解决：由于设备开发版本太多，并且规范不是十分统一，所以完全实现智能准确的自动读取设备的状态信息这部分工作还存在技术难度。而且测试用例需要不断的去扩充，尽可能全面的覆盖 Android 设备所有待测点。

参考文献

- [1] 李惠,丁革建.智能手机操作系统概述[J]. 电脑与电信. 2009(03).
- [2] 李芙蓉.当前智能手机操作系统及其比较分析[J]. 甘肃科技纵横. 2008(05) .
- [3] Baresi L, Pezze M. An introduction to software testing[J]. Electronic Notes in Theoretical Computer Science, 2006, 148(1): 89~111.
- [4] Myers G J, Sandler C, Badgett T. The art of software testing[M]. Wiley, 2011.
- [5] Poston R M. Automating specification-based software testing[M]. IEEE Computer Society Press, 1997.
- [6] Mugridge R, Cunningham W. Fit for developing software: framework for integrated tests[M]. Prentice Hall, 2005.
- [7] Kit E. Integrated, Effective Test Design and Automation-An integrated approach to test design and automation will help your team create more maintainable and cost-effective test suites, and help you get[J]. Software Development, 1999, 7(2): 27~41.
- [8] 古乐, 史九林. 软件测试技术概论 [M] .北京: 清华大学出版社, 2004.
- [9] Li Feng, heng Zhuang. Action-driven automation test framework for Graphical User Interface software testing. Autotestcon. 2007 IEEE 21~23.
- [10] 马秦晋, 刘世英, 徐小辉, 惠煜. 软件测试的重要性及现状分析[C]. 第六届全国计算机应用联合学术.2002: 56~58.
- [11] Atif M. Memon. An event-flow model of GUI-based applications for testing. Software Testing. Verification and Reliability, 2007, 17(3): 137~157.
- [12] Qing Xie, Atif M. Memon. Studying the Characteristics of a “Good” GUI Test Suite. in: Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering. 2006:159~168.
- [13] Atif M. Memon, Mary Lou Soffa. Regression testing of GUIs. in: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering. New York. 2003:118~127.
- [14] Juichi Takahashi, Yoshiaki Kakuda. Effective Automated Testing. A Solution of Graphical Object Verification. ATS'02.2002.
- [15] 竺静. 软件测试用例在软件测试中的重要性[J]. 舰船标准化工程师, 2004 (6): 40~42.
- [16] Phillip Bemhard. A Reduced Test Suite for Protocol Conformance Testing.

- ACM Transaction Software Engineering and Methodologies, 1994, 3(3): 201~220.
- [17] Kai Yuan Cai, Lei Zhao, Feng Wang. A Dynamic Partitioning Approach for GUI Testing. in: Proceedings of the 30th Annual International Computer Software and Applications Conference. 2006: 223~228.
- [18] 张仕成. 基于 Google Android 平台的应用程序开发与研究[J]. 电脑知识与技术, 2009, 5(28): 7959~7962.
- [19] 杨怡君, 黄大庆. Android 手机自动化性能测试工具的研究与开发[J]. 计算机应用, 2012 .
- [20] 王茜. Android 嵌入式系统架构及内核浅析[J]. 电脑开发与应用. 2011(04):99~108.
- [21] 姚昱旻, 刘卫国. Android 的架构与应用开发研究[J]. 计算机系统应用. 2008(11) .
- [22] 陈昱, 江兰帆. 基于 Google Android 平台的移动开发研究[J]. 福建电脑. 2008(11).
- [23] 姚昱旻. 基于 Android 的移动学习终端平台的开发与研究[D]. 中南大学 2008.
- [24] 陈璟, 陈平华, 李文亮. Android 内核分析[J]. 现代计算机(专业版). 2009(11) .
- [25] 公磊, 周聪. 基于 Android 的移动终端应用程序开发与研究[J]. 计算机与现代化. 2008(08) .
- [26] Chris Haseman. Android Essentials. . 2008 .
- [27] Dr. Markus Schmall, Jochen Hiller. Android for Java Developers. <http://www.java-forum-stuttgart.de/jfs/2008/folien/A7.pdf> . 2008 .
- [28] Jerome(J.F.) DiMarzio. AndroidTM A Programmer's Guide. . 2008.
- [29] Meier R. Professional Android 4 application development[M]. Wrox, 2012.
- [30] Open Handset Alliance . Android Activity .<http://code.google.com/intl/zh-CN/android/reference/android/app/Activity.html> . 2008.
- [31] Withall S. Software requirement patterns[M]. Microsoft Press, 2010.
- [32] 张舜尧. 手机自动化测试系统设计浅析[J]. 电脑知识与技术 (学术交流), 2007, 4(20).
- [33] 倪铭, 黄萍. 基于脚本的构件测试自动化框架[J]. Computer Engineering, 2010, 36(6).
- [34] 卢建军, 苏宁. 浅谈手机软件测试的流程与策略[J]. 制造业自动化, 2010 (012): 21~23.

- [35] 杨根兴, 宗宇伟. 软件测试不确定性研究及解决途径[J]. 计算机工程, 2004, 4.
- [36] 谭浩, 关昕, 马力. 性能测试的原理及其自动化工具的实现[J]. 计算机工程与设计, 2006, 27(19): 3660~3662.
- [37] Tsun Chow. Testing Software Design Modeled by Finite-State Machines. IEEE Trans. on Software Engineering, 1978, 4(3): 178~187.
- [38] Qing Xie, Atif M. Memon. Designing and comparing automated test oracles for GUI-based software applications. ACM Transactions on Software Engineering and Methodology, 2007, 16(1):748~752.
- [39] 路斯瑶, 胡飞. 基于事件流图的用户界面测试优化[J]. 测控技术, 2010
- [40] 陆永忠, 余幸花, 聂松林, 裴小兵, 汪春. 基于事件流图的 GUI 自动化测试系统的开发[J]. 计算机工程与科学, 2008.
- [41] Sun Y, Jones E L. Specification-Driven Automated Testing. Proceeding of the 42nd Annual Southeast Regional Conference, 2004.
- [42] 黄小勇. 一种手机自动化测试系统和测试方法[P]. 华为技术有限公司, 2007.
- [43] Atif M. Memon, Ishan Banerjee, Adithya Nagarajan. GUI rip-ping: Reverse Engineering of Graphical User Interfaces for Testing. in: Proceedings of The 10th Working Conference on Reverse Engineering, 2003:347~362.
- [44] 杨承川, 姚砺. 基于 GUI 的自动化测试框架的研究与改进[J]. 计算机与现代化, 2008: 38~41.
- [45] Mark Last, Menahem Friedman, Abraham Kandel. The Data Mining Approach to Automated Software Testing. in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 2003: 388~396.
- [46] Martin C. Brown 著, 康博译. Python 技术参考大全[M]. 清华大学出版社, 2002.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《Android 平台的自动化测试系统的实现与优化》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果。且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。。

作者签名：张兆鹏 日期：2013年7月4日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

(1) 学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；(2) 学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；(3) 研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：张兆鹏 日期：2013年7月4日

导师签名：[Signature] 日期：2013年7月6日

致 谢

至此论文完成之际，首先要感谢我的校内指导教师黄虎杰教授，本文的撰写工作是在黄老师的耐心帮助下完成的，他在我的课题研究中始终对我进行着精心的指导。无论从论文的选题、内容安排和结构设计直到论文的最终定稿等每个环节黄老师都给予了我非常大的帮助。黄老师那对学术的执着和忘我的工作热情深深的鼓舞了我，使我在这一年获益匪浅。

还要感谢实习基地指导教师康克军高级工程师在毕业设计中给予的帮助和指导，在实习阶段，他使我能够解决一个又一个困难，顺利的完成毕业设计的相关内容。

感谢哈尔滨工业大学软件学院的实习安排，这次实习大大提高了自己对理论知识的应用能力，锻炼了各方面的能力，为自己下一步的工作做好了坚实的基础。

感谢东软集团(大连)股份有限公司给我这次实习机会，感谢公司员工在实习过程中给本人的帮助，本人的点滴收获离不开前辈们的谆谆教诲。

感谢在毕业设计中给予我帮助的老师 and 同学。感谢所有在大学四年和研究生两年的学习生涯中的授课老师。

感谢爱人任芳琳对我的支持和鼓励，她在我学习过程中给予我很大的帮助，并在生活中给我无微不至的照顾。

感谢父母在背后所提供的默默支持，使我能够安心认真的实习，从而完成本次毕业设计。

最后，向百忙之中审阅本文的老师们表示衷心的感谢！

个人简历

学习经历:

2004 年 9 月考入黑龙江工程学院电气工程及其自动化专业, 2008 年 7 月本科毕业并获得工学学士学位。

2011 年 8 月至 2013 年 6 月, 在哈尔滨工业大学软件学院软件工程专业攻读软件工程硕士学位。

工作经历:

2012 年 7 月至 2013 年 5 月, 在东软集团(大连)股份有限公司嵌入式事业部实习。