

武汉理工大学

(申请工学硕士学位论文)

Android 系统 UI 性能测试方法 的研究

培养单位 : 计算机科学与技术学院

学科专业 : 软件工程

研究生 : 苏敏

指导教师 : 张能立 副教授

2015 年 5 月

分类号_____

密 级 _____

UDC _____

学校代码 _____10497

武汉理工大学

学 位 论 文

题 目 _____Android 系统 UI 性能测试方法的研究_____

英 文 _____The Research of Android UI Property_____

题 目 _____Testing Methods_____

研究生姓名 _____苏敏_____

指导教师 姓 名 _____张能立_____ 职称 _____副教授_____ 学位 _____硕士_____

单位名称 _____计算机科学与技术学院_____ 邮编 _____430070_____

申请学位级别 _____硕 士_____ 学科专业名称 _____软件工程_____

论文提交日期 _____2015 年 4 月_____ 论文答辩日期 _____2015 年 5 月_____

学位授予单位 _____武汉理工大学_____ 学位授予日期 _____2015 年 6 月_____

答辩委员会主席 _____钟珞 教授_____ 评阅人 _____付国江 副教授_____

_____佘名高 副教授_____

2015 年 5 月

独 创 性 声 明

本人声明，所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得武汉理工大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：_____ 日 期：_____

关于论文使用授权的说明

本人完全了解武汉理工大学有关保留、使用学位论文的规定，即学校有权保留、送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

(保密的论文在解密后应遵守此规定)

签 名：_____ 导师签名：_____ 日 期：_____

摘 要

随着 Android 应用软件数量的快速增长,用户可选择的应用软件越来越多,对应用软件的要求也越来越高,除了满足基本的功能外,还要求应用软件的用户体验好。应用软件的体验效果包括 UI (User Interface) 的启动时间、流畅度、内存等。在 Android 应用开发过程中,这些性能指标是 Android 应用测试人员的重要关注点。

目前,在 Android 应用测试领域里,大部分的测试仍然停留在手工测试上。通过手动操作一款应用来验证是否满足功能需求。此方法忽略了应用的 UI 性能问题,并且测试花费时间比较长。本文主要针对 Android 系统 UI 性能测试方法进行研究,并以自动化的方式代替手工测试。完成了以下工作内容:

1) 分析影响 Android 系统 UI 性能的因素,并针对不同的因素提出对应的测试方法,包括:利用 LogCat 视图测试 UI 启动的时间,用命令行“`dumpsys meminfo`”与 Heap 视图对内存进行测试,通过 GT 工具测试 UI 的流畅度。

2) 对性能测试中的 UI 启动时延与流畅度测试方法提出改进方案,并对改进方案进行分析,得出改进后的优点。

3) 将研究的内容与改进后的测试方法应用到一个具体的实际项目中。首先,对测试初期工作进行介绍,包括:需求分析、需求评审、测试环境设计、测试用例设计等。接着,实现具体的测试方法。最后,选择与被测项目同一类型的一款产品进行测试结果的对比分析,提出应用软件自身需要优化与改进的地方。

4) 通过测试效率与测试结果的分析,验证改进后的启动时延与流畅度测试方法的效果。

本文提出了使用自动化脚本获取应用启动日志的方式来测试 UI 的启动耗时,并且在流畅度测试方面,通过 MonkeyRunner 工具录制脚本实现自动滑动 UI,采集 FPS 值的功能。最终的测试结果表明,使用自动化测试 UI 的启动耗时与流畅度,节约了测试人员一半的时间,并且提高了测试结果的精确度。

关键词: Android 系统 UI, 性能测试, 自动化测试, MonkeyRunner

Abstract

With the rapid growth of technology of Android, the number of applications that people could choose is becoming bigger. Correspondingly, the quality required is becoming higher. Besides basic functions, good User Experience (short for UE) is required too. UE includes UI (User Interface) boot time, fluency, memory etc. High quality controlling during the process of android application development is the duty of a software test engineer.

At present, most of the testing jobs of android are still finished manually. These methods of verifying application manually ignore the performance problem, and also time wasting. In this paper, we focus on the research of Android UI performance testing methods and propose an automated one. Works concluded as follows:

- 1) Analysizing factors that effect the Android UI performance, and propose different testing methods respectively. Like: Logcat View to test UI boot time, command line “dumpsys meminfo” with Heap View to test memory, GT tools to test the fluency of UI.

- 2) Propose the optimized method of testing UI boot time and fluency in performance, attached with the analysis of the method and advantage.

- 3) Implement the optimized method to a concrete project. At first, we cover the preliminary stage work include requirement analysis, requirement assessment, testing environment and testing case designation. Then we implement the concrete testing method. Finally, we compare and analyze the result with the same kind of application as the testing project, and then propose the points that need optimizing and improvement.

- 4) Analysing the testing efficiency and result, test engineer achieved a better result from the optimized method on UI boot time and fluency testing.

This paper proposes a serial of automated methods include using automated script to retrieve boot log for testing the UI boot time, using MonkeyRunner to recorded automated UI slide and collect the FPS value. The experiment indicates that using automated method saves half the time of test engineer and improves the

accuracy.

Key words: Android UI, Performance Testing, Automated Testing, MonkeyRunner

目 录

摘 要	I
Abstract.....	II
第 1 章 绪论	1
1.1 课题研究背景	1
1.2 国内外现状分析	2
1.3 课题研究目的及意义	3
1.4 论文的组织安排	4
第 2 章 Android 系统 UI 设计及测试.....	5
2.1 Android 系统 UI.....	5
2.1.1 UI 的层次结构	5
2.1.2 UI 的布局	6
2.2 Android 系统 UI 的测试	8
2.2.1 DDMS 工具.....	8
2.2.2 UIAutomator 工具	9
2.2.3 UI 测试内容	11
2.3 本章小结	14
第 3 章 UI 性能测试方法的研究	15
3.1 影响 UI 性能因素的分析	15
3.1.1 UI 的启动耗时	15
3.1.2 内存泄露	17
3.1.3 流畅度	19
3.2 UI 性能测试的方法	20
3.2.1 启动时延测试	20
3.2.2 内存测试	23
3.2.3 流畅度测试	26
3.3 启动时延测试方法的改进	30
3.3.1 改进的思路	30
3.3.2 改进后的优点	37
3.4 流畅度测试方法的改进	37

3.4.1 改进的思路	37
3.4.2 改进后的优点	39
3.5 改进后的自动化测试方法与现有方法的区别	40
3.6 本章小结	41
第 4 章 改进 UI 测试方法的使用与结果分析	42
4.1 项目背景介绍	42
4.2 需求分析与设计	43
4.2.1 测试需求分析	43
4.2.2 需求评审	45
4.2.3 测试环境的设计	45
4.2.4 测试用例的设计	47
4.3 测试方法的实现	51
4.3.1 功能测试方法的实现	51
4.3.2 启动时延测试方法的实现	53
4.3.3 内存测试方法的实现	55
4.3.4 流畅度测试方法的实现	57
4.4 测试结果的分析	58
4.4.1 启动时延测试结果的分析	59
4.4.2 内存测试结果的分析	60
4.4.3 流畅度测试结果的分析	60
4.5 改进方法的结果对比	61
4.5.1 启动时延测试方法的对比	61
4.5.2 流畅度测试方法的对比	63
4.6 本章小结	64
第 5 章 总结与展望	65
5.1 总结	65
5.2 展望	66
致谢	67
参考文献	68
攻读硕士期间的研究成果	71

第 1 章 绪论

1.1 课题研究背景

在移动互联网时代的飞速发展与软硬件技术的不断创新下，移动终端设备已成为市场的主流产品。如今，无论是在现代化的一线大都市，还是在发展中的二、三线中小型城市，小学生、成年人、老年人等不同年龄阶段的人群中基本每个人手上都有一部智能手机。用户对移动智能设备的迫切需求推动了各大厂商投入大量的资金、人力、技术等生产高端、中端、低端等移动智能手机。

目前，在移动市场上主流的操作系统有 Android 和 IOS 两大操作系统^[1]，其中 Android 的占比率为 83.10%，紧随其后的是 IOS，占总数的 12.70%，除此之外 Windows Phone 占比 3.00%，Blackberry 占比 0.80%。Android 操作系统深深吸引了广泛开发者的密切关注。许多擅长其他开发语言的程序员开始转型到 Android 操作系统的开发中，带动了 Android 手机操作系统在计算机应用领域中的广泛使用^[2]。

Android 操作系统能够获得如此神速的发展是因为 Android 操作系统有别于其它操作系统^[3]。Android 系统的开源性特点不仅降低了程序员的学习成本，也给厂商带来了巨大的商机。任何手机厂家可以把自家的软件服务跟 Android 系统紧密结合，开发出各种应用软件，为用户带来各种新鲜的体验。例如，三星、华为、HTC、小米等各大手机制造厂家将硬件配置的重视程度逐渐转移到 Android 系统与应用软件的开发中了。同时，各大小公司在 Android 系统上开发应用软件，使手机设备的界面更加丰富，功能变得更加强大。与之相比，曾经风靡全国的诺基亚公司因止步不前、墨守成规，不接纳 Android 操作系统，终将走向了没落。

Android 系统从 2008 年 10 月的 1.0 版本到如今 5.0 版本，功能越来越强大。操作系统的不断完善与移动终端智能设备在广大用户群体中的使用，激励着一批又一批的开发人员编写出各类移动应用软件^[4]，其中，包括社交聊天、娱乐游戏、影音视频、生活工具等。应用软件中有一类专门的手机系统主题应用，例如目前比较受用户喜爱的 360 手机桌面、Android 91 桌面、Go 桌面、小米桌面、RUI 手机桌面等等。Android 应用提供的服务大大提高了人们的生活品质，

给人们的生活带来了乐趣与便捷。

Android 系统用户界面 (UI, User Interface) 的变化,也是用户有目共睹的。从最初的简单启动器到如今功能齐全,界面丰富的各类应用主题,给人赏心悦目的视觉效果。经过最近几年的快速发展,各大生产智能设备的厂商都开始投入一部分人力到自身设备 UI 主题定制与开发中,比如华为、HTC、MIUI 等制造的手机都带有自己的原生桌面。同时,UI 的开发吸引了一批又一批创业者团队,他们在不断探索 Android 的过程中开发出了比较优秀的主题软件有:360 手机主题、魔秀主题、Android 91 桌面、RUI 手机桌面等等。

伴随着 Android 操作系统的快速发展^[5],移动应用软件越来越丰富,同一类型的软件,例如,手机桌面软件,可能都有成千上万种。在这个应用软件风靡市场的时代,一个软件想要站稳脚步,除了功能满足用户需求外,还要注重用户的体验效果,如界面美观、使用流畅、占用内存小等。在 Android 应用开发过程中,对移动应用软件的质量把关,是一名软件测试人员的职责所在^[6]。

1.2 国内外现状分析

为了保证 Android 应用软件的质量与提高测试工作的效率,近几年来,国内外陆续推出了一系列的开源测试工具。

首先,Android 应用软件的测试技术起源于国外,包括:

1) Google 公司自主研发了 Monkey、MonkeyRunner、UIAutomator^[7]等等 Android 应用软件测试工具。其中,Monkey 工具,用于对一款 Android 应用软件进行压力测试;MonkeyRunner 工具,对 Android 应用软件执行回归测试;UIAutomator 用于测试一款 UI 的布局等等。

2) FaceBook 公司也推出了一款开源测试工具 ATC^[8]。ATC (Augmented Traffic Control),主要用于模拟在各种网络环境下,测试 Android 应用软件在使用过程中是否会出现数据丢包、网络延迟、连接中断等等异常问题。

3) IBM 公司开发的 RTW8.5(IBM Rational Test Workbench 8.5 版本)工具主要通过 PC 端录制脚本实现自动化测试 Android 应用软件的功能。

其次,在国外 Android 测试技术的开源基础下,国内的 Android 应用软件测试技术也得到了突飞猛进的发展。目前,国内一些大公司相继自主研发了一系列的 Android 应用软件测试工具,比较出名的有:

1) 百度科技公司开发了一款云测试工具 MTC(Mobile Testing Center)。该

测试平台上聚集了成百上千种类型的设备，用户只需要将自己开发的 Android 应用软件上传到该平台上，就可以对应用软件执行相应的兼容性、性能、虚拟场景等方面的测试工作并且测试结果一般会在八个小时以内以报告的形式输出。总之，一个公司的设备资源比较有限，MTC 工具能够帮助测试人员测试一款 Android 应用软件，让该应用软件尽可能多的兼容市场上已经存在的设备。

2) 腾讯科技公司开发的一款 Android 应用软件性能测试工具 GT。GT 工具安装在用户的设备上就可以直接操作使用，无需与 PC 机相连接。该工具比较便捷，能够帮助测试人员进行内存、流畅度、流量等方面的测试工作。

3) 网易公司自主研发的 Emmagee 工具，与 GT 工具功能比较类似。该工具能够方便的对一个应用软件进行性能测试。譬如测试应用软件的内存、CPU、流量等。

1.3 课题研究目的及意义

随着 Android 应用功能越来越强大，Android 界面上布置的元素也逐渐增多，例如图片、文字、按钮、动画^[9]等等。在 Android 应用的开发过程中，如果对这些资源文件没有合理的使用，将会导致一些应用性能问题。例如，在图片的使用过程中，不进行合理的裁剪与压缩，将会占用设备大量的内存，甚至产生内存泄露的问题。其次，如果在一个界面上，不注意界面布局，过度的重复绘制一个元素，将会影响一个界面的流畅度。再次，在一个应用的启动过程中，设置大量的写文件或解压缩包操作等等，将会导致一个界面启动非常耗时。

本课题主要针对上面提出的三类性能方面的问题提出对应的测试手段，并对 UI 性能测试方法进行研究改进。研究的意义包含：

1) 一个应用软件在启动的过程中会出现长时间页面加载、闪屏、黑屏等情况，导致应用启动慢。通过应用启动时延的测试方法，将测试结果与同类产品的启动耗时数据进行对比与分析，从而发现自身需要优化的地方。

2) 虽然 Android 应用的内存问题，开发人员也在关注与优化，但是由于开发需求的增多，或者项目代码量的增多，从而忽略了一些常规内存问题。例如在程序代码中重复创建多个对象，对象使用后未及时销毁，使用的图片资源未经过处理等等，这些问题都会引发应用内存泄露。通过内存测试的方法可以及时发现或规避内存泄露的问题。

3) 目前，Android 应用的用户界面越来越丰富，界面滑动的流畅度已成为

Android 用户关注的焦点。为了提高用户的体验效果与避免用户在使用过程中出现卡顿（Jank）问题，对 UI 进行流畅度测试成为 Android 应用软件开发过程中必不可少的环节。

1.4 论文的组织安排

本文的组织结构，以及每一章节的基本内容如下：

第一章为绪论，分别对课题的研究背景、国内外现状、课题研究的目的与意义以及整个文章的组织结构进行介绍。

第二章为基础章节，首先介绍了与 Android 系统 UI 设计相关的基础知识，包括 Android 系统 UI 的层次结构、UI 布局。接着，对 UI 测试过程中经常使用到的测试工具 DDMS 与 UIAutomator 工具进行详细的讲解，并简单介绍了查看 UI 布局的 UIAutomatorViewer 工具。最后，论述了与 UI 测试相关的内容。

第三章为本文的核心章节，首先，对影响 UI 性能因素进行详细分析，包括 UI 的启动时延、内存、流畅度。接着，针对 UI 性能因素提出具体的性能测试手段，例如利用 DDMS 工具中的 LogCat 视图观察 UI 的启动时间；在 Heap 视图中通过 Allocated 值的变化测试应用内存是否稳定；运用 GT 工具或命令行的方式获取 FPS（Frame Per Second，指每秒传输的帧数）值，计算 UI 的流畅度等。再次，对性能测试中的启动时延与流畅度的测试方法提出改进方案。最后，列举改进后的性能测试方法与现有的 Android 系统 UI 性能测试的自动化方法的区别。

第四章将研究的内容与改进的测试方法应用到一个具体的实际项目中。主要内容包括：针对一款 Android 系统 UI 智能桌面进行测试初期工作，包括项目背景介绍、测试需求分析、需求评审、测试计划制定、设计测试用例与配置测试环境等。接着对 RUI 手机桌面执行测试，并对测试结果进行分析。最后，通过测试结果的对比，验证改进后的测试方法提高了测试人员的工作效率与测试结果的精确度。

第五章对整个研究中的工作及测试方法经验进行全面深入的总结，指出本文中的不足之处，并对后续研究工作进行了展望。

第 2 章 Android 系统 UI 设计及测试

2.1 Android 系统 UI

Android 系统 UI 一般是指手机屏幕上的内容，用户能够看到，并可以供用户操作，比如相机、浏览器、电话本、时钟、设置等等应用的集合体就是一个用户界面。狭义上理解，一个桌面启动器（Launcher）程序就是一个 Android 系统 UI^[10]。每个 Android 设备都自带一个系统桌面启动器，设备上所有的应用都是从这里启动的。

2.1.1 UI 的层次结构

在 Android 系统中，一个用户界面包括多个应用程序。应用程序由 Activity 组成，同时系统为每个 Activity 分配了一个默认的窗口界面（Window），开发人员可以在界面上摆放一些 UI 组件，如标签、按钮、文本框、单选框、多选框等等，这些控件存放在显示视图（ContentView）中。Activity 对应的窗口界面就好比一个大画板，画板上可以描绘各种元素，UI 组件就是画板上布置的内容。可见，窗口界面中的组件越丰富，布置的越美观，则越能吸引用户的眼球。

在 Android 系统中，构成 UI 的基本元素是 View 和 ViewGroup^[11]。View 是可以看见的，用于与用户交互的一些控件的集合体。ViewGroup 是一个容器类，也是 View 的一个扩展类，可以包含多个 View，也可以派生出 ViewGroup。Android 上的所有界面组件（控件、图片、文本框等）都建立在 View 与 ViewGroup 上。

如图 2-1 所示，是一个应用程序 UI 的层次结构图^[12]。

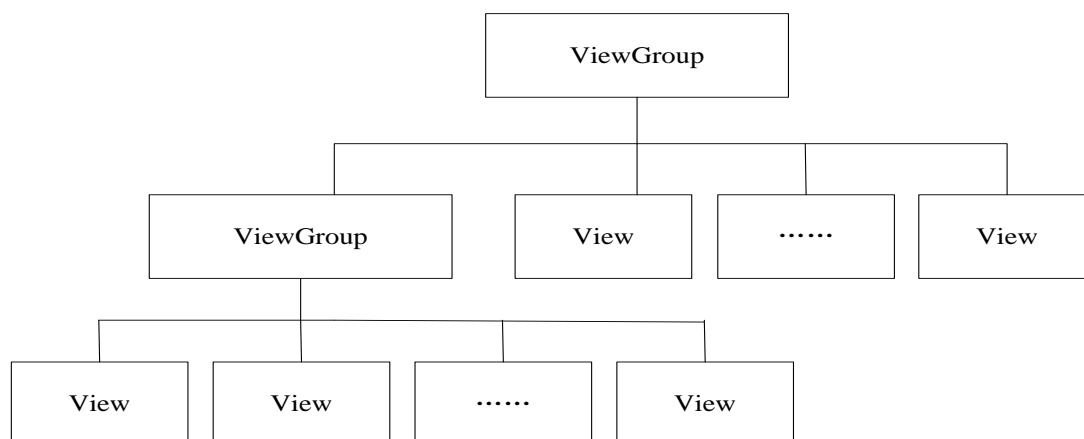


图 2-1 UI 层次结构图

一般来说，在 Android 平台上，创建 UI 布局通常有二种方式^[13]。一种是在一个 Android 工程下的 XML 文件中直接创建。在 XML 文件中创建布局 and UI 控件后，程序在运行的过程中就会加载 XML 文件，此时，就可以在 View 上面生成对应的界面布局，其中 XML 文件位于项目中的/res/layout 文件夹的目录下。

第二种则是在 Java 代码中直接添加布局元素实现界面布局。开发人员可以在应用程序中使用 View 和 ViewGroup 来创建对象^[14]，并指定设置的布局类型与参数。

在 XML 文件中实现布局层次比较清晰，结构比较鲜明，并且可以实现比较复杂的布局界面。在代码中直接添加布局显得更加灵活，但缺点是：使得代码比较多，混乱。

除了以上说的两种方法外，还有一种最简单、便捷的方式是通过 Android 下的 layout 面板，直接拖对应的控件至面板上，并设置控件属性值。该方式只能实现比较单一的布局，需要与 XML 文件生成布局的方式一起使用。

2.1.2 UI 的布局

Android 系统用户界面的布局情况是决定一个 UI 是否美观的基础^[15]。UI 布置的合理、层次鲜明，有利于用户的使用。因为移动设备的屏幕大小和 PC 屏幕差异比较大，所以需精心的选择布局进行装载控件与信息。在 Android 系统中提供了五个布局类，它们分别为框架布局 (FrameLayout)、线性布局 (LinearLayout)、绝对布局 (AbsoluteLayout)、相对布局 (RelativeLayout)、表格布局 (TableLayout)^[16]。以下分别对 UI 的常见布局进行介绍：

1) 框架布局 (FrameLayout) 是最简易的布局^[17]。手机的主界面就可以看成一个框架视图, 界面上的每个应用又是一个个子视图, 即所有的布局都是在框架视图中的再一次布局操作。框架布局好比一个书桌, 书桌中的抽屉、柜子又是一个个独立的布局。当建立一个框架视图时, 界面上显示是一个黑框, 可以在这个黑色窗体上加上一些元素, 例如, 图片作为背景色。

2) 线性布局 (LinearLayout) 是在 Android 应用的开发过程中使用频率最高的一种布局视图。最常见的是在刚刚创建的一个 Android 工程项目中, 将 LinearLayout 布局设置为默认的布局视图^[18]。线性布局分为横向排列和纵向排列。横向排列是指每一列只有一个组件, 从左到右依次摆放。则纵向排列刚好相反, 每一行只有一列, 依次由上而下排列。

3) 绝对布局 (AbsoluteLayout) 是为界面上需要放置的控件指定了一个固定的位置, 即横坐标 (X) 与纵坐标 (Y)。采用绝对位置维护屏幕上的控件位置会相当繁琐。此布局不具备灵活性, 无法满足 Android 设备多样化的特性, 因此建议在应用程序开发中尽量少用。

4) 相对布局 (RelativeLayout) 是指布局中的元素之间有个参考的关系, 需要通过与其他视图的关系来确定自身的位置^[19]。例如, 将一个文本框放在按钮的右边, 图片的下面放一个按钮。文本框、按钮、图片这三者之间就是一个相互参照的关系, 它们的摆放位置由另一个依赖组件的位置关系所决定。相对布局是将一些独立的控件或容器组合在一起的桥梁。熟练掌握相对布局, 可以设计出强大、丰富的 UI。

5) 表格布局 (TabLayout), 顾名思义, 该布局就像一张表格, UI 组件存放的位置就是表格中的单元格。在表格布局中, 可以设定表格的行高和列宽等属性。常见的应用程序中使用表格布局的有计算器, 拨号器等^[20]。

综上所述, 这五种布局类各有千秋, 开发人员在开发过程中应根据具体场景与需求合理运用布局, 布局之间也可以嵌套使用。在布局上添加各种 UI 组件, 才能开发出灵活、丰富、美观的用户界面。

Android 系统中主要的界面控件有: 单选框 (RadioButton)、多选框 (CheckBox)、列表显示 (ListView)、下拉列表框 (Spinner)、拖动条 (SeekBar)、进度条 (ProgressBar)、图片组件 (ImageView) 等。

2.2 Android 系统 UI 的测试

Android 系统 UI 的测试属于 Android 应用测试中的一个类别。同时，拥有 Android 应用中经常使用到的一些测试方法，例如，一个 UI 的安装卸载、兼容性适配性、业务功能与性能等方面的测试^[21]。

在 Android 测试的过程中经常用到的工具有 Robotium、Monkey、CTS、DDMS、MonkeyRunner、UIAutomator^[22]等等。本文主要介绍了 DDMS 工具与 UIAutomator 工具。最后，简要描述用于查看 UI 元素的 UIAutomatorViewer 工具。

2.2.1 DDMS 工具

在执行整个 UI 性能的测试中，用得最多的工具非 DDMS 莫属了。DDMS (Dalvik Debug Monitor Service) 是 Android 开发环境中的 Dalvik 虚拟机调试监控服务^[23]。在 Android 系统环境下，每个应用程序都拥有自己独立的进程，这些进程运行在一个属于自己的虚拟机上。同时，虚拟机为每一个进程提供了唯一的端口号，用于区分和连接调试器进行调试。

DDMS 工具保存在 SDK 下的 tools 文件中，双击该目录下的“ddms.bat”可以启动 DDMS 工具，或是在命令行中输入“ddms”亦可开启该工具。DDMS 工具可以用于查看手机上所开启的线程信息^[24]。

首先在设备型号的窗口中选择要查看的进程，按下“update threads”图标，此时，右边的“Threads”窗口就会显示所选进程包含的所有线程信息，并且当选择其中一个线程后，详细信息就会在下面的窗口中看到。

如图 2-2 所示，在进程栏中选择“com.uprui.phone.launcher”进程。然后，点击更新进程按钮，对应的右边窗口弹出进程中包含的所有线程信息。例如，双击进程中的一个线程“RuiSchedulThread-1”，下面对应的窗口就展示出了线程的详细信息。具体操作步骤在图 2-2 中分别标注为①、②、③、④。根据线程详情信息可以看出当前线程正在等待 MessageQueue 中的消息，这些线程信息对于调试线程竞争的情况很有帮助。

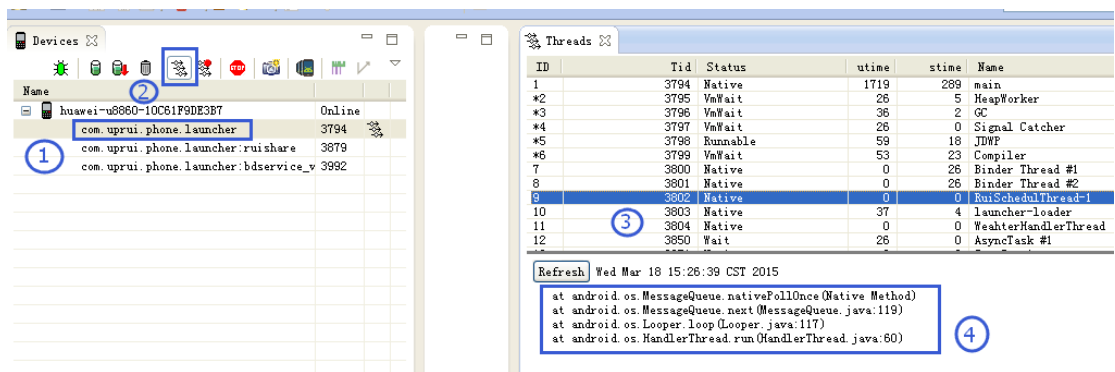


图 2-2 DDMS 线程信息图

除了以上描述的跟踪线程信息外，DDMS 工具还具有如下几个常用功能：

- 1) LogCat 视图属于 Android 应用的日志系统，用于显示系统组件和应用程序在不同情况下的输出日志信息。
- 2) Emulator Control 用于对正在运行的设备进行一些交互操作，例如，打电话、发送文本消息。
- 3) Allocation Tracker 视图显示某一时段里内存的资源分配情况，一般可以用于测试内存泄露的问题。
- 4) File Explorer 视图实现的功能是将文件从手机端与 PC 端进行互传，也可以对手机中的文件进行删除、复制、移除等操作。
- 5) Heap 视图计算当前进程在设备中所占内存大小。例如，观察视图中“Allocated”值的变化，测试应用软件是否存在内存泄露。

2.2.2 UIAutomator 工具

UIAutomator 是 Google 官方提供的 Android 系统 4.1 以上版本自带的一个 UI 测试框架^[25]。用来做 UI 测试，它可以获取到界面上所有的 View 元素和控件，也可以用来做功能性测试，自动点击界面上的控件元素，也就是相当于测试人员在手工操作一款 UI。

测试人员可以在不需要知道 APP 的源代码的情况下就用 UIAutomator 对程序进行 UI 测试，包括点击 UI 界面上的控件按钮、文本框中内容的输入与清除、拖动控件等操作。UIAutomator 测试是权限范围最大的，可以对整个系统做 UI 测试，而不仅仅是局限于单个 APP。

UIAutomator 环境的配置比较简单，需要安装 Android 4.1 以上的 Android

SDK。首先，在命令行中输入“android list”，就会把所有的 sdk 打印到控制台上。接着，选择 API 大于 16 以上的那个 android target 并记住这个 target 前面的 ID 号。

最后，新建 UIAutomator 工程。UIAutomator 是一个 java 工程，所以新建工程时跟新建 java 工程是一样的。在新建的工程中需要导入两个 Android SDK 的 jar 包，即 android.jar 和 uiautomator.jar。同时还需要导入一个 junit 库。当工程建好后，就可以在 src 目录下面新建 class 文件了，并将其基类（superclass）设置为 UiAutomatorTestCase。此时，表示一个 UIAutomator 已经建立完成。

UIAutomator 常用的 API(Application Programming Interface)包括 UiDevice、UiObject、UiSelector、UiScrollable、UiCollection^[26]。以下分别介绍：

- 1) 在测试中可以通过 UiDevice 对象去对设备进行控制，如点亮屏幕、锁屏、旋转方向、截图、点击 home 键、back 键、menu 键等。UiDevice 对象可以通过 getUiDevice()方法来获取。
- 2) UiObject 可以对整个 UI 里面的某个元素或控件进行操作。也就是说，可以通过 UiObject 对象来对控件模拟用户的操作。在 UiObject 构造函数里传入一个 UiSelector 对象，通过该对象来查找所需要的元素或控件。
- 3) 可以通过 UiSelector 对象去定位 UI 元素。如果发现多个满足条件的控件则会返回第一个控件，还可以使用 childSelector()函数来嵌套 UiSelector 对象。
- 4) 如果操作的控件需要滚动就可以使用 UiScrollable。UiScrollable 可以实现向前或向后滚动、快速滚动、滚动到某个对象等等。
- 5) UiCollection 代表所有控件的集合，可以用来获取所有控件的个数，获取子元素对象。

UIAutomatorViewer 是一款与 UIAutomator 外形比较相似的工具，可以用它来对 UI 组件进行分析^[27]。在命令行中输入 uiautomatorviewer 就能启动这个工具。点击右上角的 Device Screenshot 就可以获取当前的 UI 组件。

如图 2-3 所示，在窗口中，主要根据左边的 UI 分析右边对应的界面布局情况。

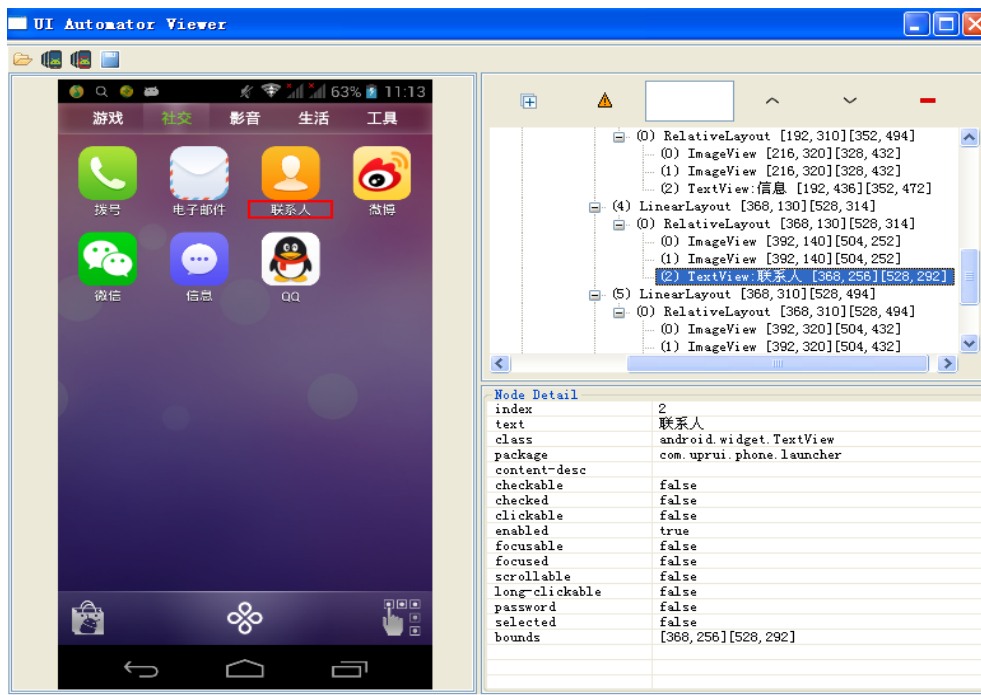


图 2-3 UIAutomatorViewer 界面

从界面中可以看出，当前选中的是“联系人”区域，右边呈现出了“联系人”的布局情况。它是 `RelativeLayout` 下的一个 `TextView` 属性，其相对布局上面又嵌套一个 `LinearLayout`。同时，可以通过下面的“Model Detail”窗口查看选中元素的一些常用信息，包括元素所在父类中的位置，即索引（index），应用的包名与类名，控件的可见范围的坐标值（bounds），对该组件是否可以执行点击操作，是否可以滑动等等。

2.2.3 UI 测试内容

（1）UI 业务功能测试

业务功能测试主要是验证 UI 的功能是否完备，是否符合需求文档所要求的功能点^[28]。例如，项目需求文档上有一个功能点是需要在 UI 上呈现一个天气插件，并保证天气插件显示的天气情况是准确的。这时 UI 的业务功能就包括验证天气插件是否在界面上显示、是否可以移动天气插件、天气插件在有网络的情况下是否能够正确定位当地的天气等等。

（2）兼容性测试

兼容性测试主要包括：Android 系统版本的兼容^[29]。目前，Android 系统

版本有 2.3、4.0、4.4 等等，在测试计划中，需要单独安排时间用于 Android 不同系统的兼容性测试。其次，是手机不同厂家系统的支持。不同厂家会有不同的 Android 系统，例如：小米、华为、锤子对市面上主流手机的支持。再次，手机不同分辨率与尺寸的支持。例如，3.5 寸的 UI 到 5.0 寸屏幕上显示会有区别。最后，网络的兼容性。例如，在 2G、3G、4G、WIFI、弱网络和断网时 UI 是否显示正常。与此同时，UI 应用安装卸载是否兼容在手机端直接安装，或是 PC 连接手机后通过第三方应用软件进行安装等等。

（3）UI 性能测试

在 Android 系统应用的开发过程中，性能测试是保证用户体验效果的有效途径^[30]。UI 的性能测试主要是测试 UI 的启动时间，UI 与 UI 之间切换是否出现卡顿、UI 占用的内存是否会突然暴增等等。这些也是本文研究的重点内容。除此之外，性能测试还包括 UI 耗电量与流量测试、占用手机设备的 CPU 大小、Crash 率测试等等。

（4）稳定性测试

稳定性测试又包含持续测试、压力测试、Monkey 测试。持续测试是指对一个 UI 连续进行操作，不受外界条件的影响。例如，在网络良好的情况下，持续点击 UI 上的一个控件按钮，观察是否会出现异常的情况。在网络比较弱的环境下，执行同样的操作，观察 UI 变化。此时，网络突然中断，仍然持续操作，是否有异常情况发生。压力测试，与持续测试比较类似，一般是指对一个功能点执行很多次数的操作，确定 UI 是否能够抵抗住。比较典型的例子就是，在测试一款手机游戏时，要不停的执行按键操作，观察游戏是否出现奔溃或失灵等异常现象。Monkey 在 UI 测试中属于压力测试中的一种。Monkey 是 Android 自带的一款测试工具^[31]，通过 adb 命令执行一些简单的滚动、点击、输入等操作，它是一种无目的的操作，这些操作事件都是随机的。

（5）界面易用性测试

界面易用性测试主要是指 UI 布局符合 Android 交互规范、用户的习惯、使用快捷方便等特性^[32]。例如，UI 界面上的一个对话框有两个按钮，分别是取消与确定。一般而言，用户习惯点击右边的按钮表示确认的含义，左边按钮表示取消。同样的，易用性还表示可以用性强、操作简单、完成任务使用时间短等等。因为使用 Android 应用的用户群体比较广泛，一个应用设计的越简单、功能入口越容易发现，则用户操作起来就更加方便、快捷。

（6）接口性测试

一般而言，一个 Android 应用包括客户端（Client）与服务器端（Service）两个方面^[33]。接口性测试主要是验证 Client 端的数据更新与 Service 端是否一致，即 UI 界面上是否能够根据服务器端实时更新数据。最简单的例子：微信朋友圈中，好友动态信息的更新。如果用户的某位好友发布了一条动态消息，朋友圈会立即提示有好友动态消息。接口性测试发现的异常现象有：Client 端更新时断开了；Client 端更新时 Service 端挂了等等。

（7）交互性测试

交互性测试主要是指手机端与 PC 之间，应用软件与系统软件之间的交互^[34]。通过 USB 线将手机与 PC 端相连接后，是否能够在 PC 端上直接通过应用宝、手机助手等软件将待安装的手机应用包安装至设备端。在执行一个 UI 时，如果中间突然打进一个电话，当前 UI 与电话应用的界面是否能够正确进行交替等等。

（8）回归测试

在一个应用完成了所有功能、性能、兼容、交互性等测试之后，就预示着第一轮测试基本快结束了。待开发人员将测试人员提交的 Bug 解决之后，就要进行一轮轮的回归验证测试^[35]。回归测试是按照已经设计好的测试用例将所有功能过一遍，它的目的是验证 Bug 的解决情况以及修复的 Bug 是否会影响 UI 相关模块的其他功能。在 UI 回归测试中，经常利用 MonkeyRunner 工具录制脚本，进行回归测试^[36]。

（9）灰度测试

灰度测试是在 UI 产品发布之前进行的一项有效测试活动。灰度测试是指选择一部分忠实的用户，试用已经开发好的软件产品。通过灰度测试可以及早的收集到用户的反馈意见，从而不断完善应用的功能^[37]。同时，由于 Android 设备型号比较多，在测试升级安装时，无法覆盖到所有的设备，通过灰度测试可以弥补资源不足的困境。根据选择用户的范围又将灰度测试分为小灰度与大灰度测试。小灰度测试一般是指选择公司内部员工，如开发人员、运维、产品经理、推广人员，使用已开发好的一款产品，用以及时发现产品中的缺陷。大灰度，不仅仅在公司内部试用产品，而且同时市面上选择一部分忠实的用户使用即将发布的应用软件。

（10）竞品测试

竞品测试主要是指选择同一类型产品进行功能方面与性能方面的对比，从而发现自身产品的不足与需要改进的地方。例如，在测试 RUI 手机桌面应用软

件时，就会选择 360 手机桌面、Android 91 桌面作为自身的比较对象，从而发现需要进一步优化的地方。通过竞品测试可以使一个软件产品不断完善与优化，达到极致的效果。

2.3 本章小结

本章主要介绍了 Android 系统 UI 设计的基础知识与 UI 测试的基本内容。其中与 UI 开发相关的知识点，包括 UI 的层次结构、UI 的布局。接着介绍了 UI 测试过程中经常用到的 Android 测试工具，如 DDMS、UIAutomator 工具等等。最后，对 UI 测试方法进行论述。

第 3 章 UI 性能测试方法的研究

3.1 影响 UI 性能因素的分析

一个 UI 的性能除了与设备本身的硬件配置、使用的网络环境有关外，还受到应用本身的影响。比如，一个用户界面打开非常慢，应用占用内存很大甚至出现内存泄露等问题，都属于应用本身的性能问题，这些性能问题严重影响了用户的体验效率。以下分别从启动时间、内存、流畅度等方面对 UI 的性能进行分析。

3.1.1 UI 的启动耗时

UI 的启动时间往往是用户的第一印象，用户初步接触一个应用时，第一动作是点开应用，观察 UI 的启动。当一个 UI 启动非常慢，还会出现黑屏、闪屏，则这种情况严重影响了用户继续使用应用的心情。一个界面启动的速度决定了该应用软件的使用周期。

一般情况，一个 UI 的启动时间分为两大类：一类是响应时延，即打开窗口的时间，在 web 页面就从浏览器发请求到 html 开始渲染的时间；另一类是数据加载时间，包括加载本地数据和网络数据，本地数据可能存在加解密和解压缩等过程。无论是客户端还是在 web 页面，在测试新需求时，应该关注是否新增窗口、数据容量最大多少、数据加载是同步还是异步等。

因此，一个 UI 启动速度慢由两个原因引起。

第一个原因是外界原因导致，例如移动网络的慢，用户当前的网络状况不好，如网络阻塞，或者用户使用的是 2G 网络。而 2G 网络的信号强度远远小于 3G、4G，甚至 WIFI 网络。在这种弱网络的情况下，UI 的启动、数据加载、刷新等等都会出现卡顿现象。

如图 3-1 所示，显示了一个界面正在缓冲的过程。在网络比较差的情况下，打开一个 UI 界面，会出现界面加载时间长的问題。



图 3-1 数据缓冲黑屏图

第二个原因，是应用自身原因导致 UI 启动缓慢。比如，一个 UI 在启动的过程中进行大量的写文件、重复读文件、解压缩包等操作时，都会影响到 UI 的启动速度。因为在写文件的操作过程中，是将整块擦除之后再进行写，所以比较慢。另外，在擦除一个文件块时，需要相对较长的时间；或者是如果写操作试图修改一个包含已有数据的文件，那么这个块中所有的有用数据页都会拷贝到一个新的页面后，再对该页面进行写操作^[38]。因此，在一个应用的启动过程中应该尽量执行一些数据的加载，数据的读取操作，避免过多的写文件操作。

本文主要针对应用自身原因导致的 UI 启动耗时长进行测试。也就是说，在没有开启网络环境的情况下，启动一个应用，等待 UI 界面弹出时观察对应的启动时间。

UI 启动是一个 Activity 的跳转原理。在一个 Activity 的界面跳到另外一个 Activity 时，需要一个触发。最简单的例子是用户在 ActivityA 界面上点击了某一按钮，启动 ActivityB，其中 ActivityA 与 ActivityB 均表示两个不同的界面。

如图 3-2 所示，就是从 ActivityA 界面跳到 ActivityB 界面的原理图。从图中可以看出，当 ActivityB 界面上没有控件加载时，用户的感知时延跟测试的数据基本一样。从 ActivityA 跳到 ActivityB，首先在 ActivityA 调用一个 `startActivity(intent)` 方法，接着通过 `onCreate()` 创建 ActivityB，但一直调用到 `onResume()`，ActivityB 界面不会立即呈现，只有将 View 添加到 Window 之后，ActivityB 所在的界面才显示出来。这个跳转过程所花费的时间，就是 UI 的启动耗时。

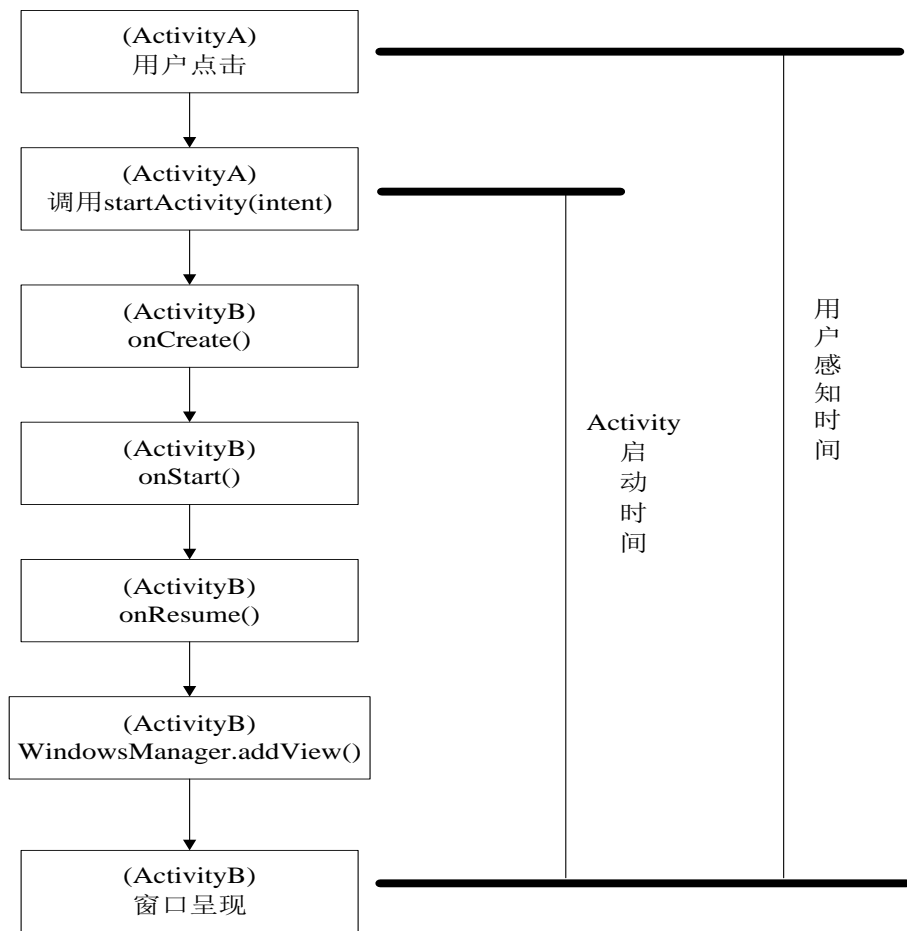
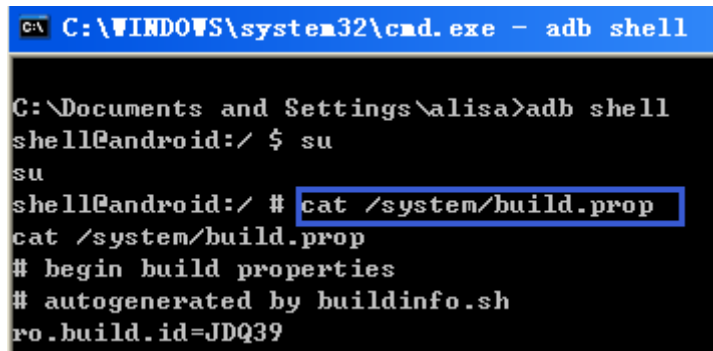


图 3-2 Activity 跳转原理图

3.1.2 内存泄露

一个应用所占内存过大，是引发 UI 性能问题的第二个原因。如果一个 UI 的内存优化的比较好，占用内存少，则用户的手机不会出现突然卡死或界面崩溃的异常现象。目前，市面上，手机种类繁多、内存大小各不一样，因此，开发人员在设计一个 UI 应用程序的时候，理应考虑到低配置、低内存的手机。

Android 系统为设备上的每个应用程序都设定了一个最高的可用内存阈值，当一个应用的内存超过这个阈值就会产生 OOM（Out Of Memory）错误，即内存泄露问题。其中，不同的设备为应用程序进程设定的阈值不同，可以通过在命令行中，输入“cat /system/build.prop”获取应用内存阈值。具体命令行，如图 3-3 所示。

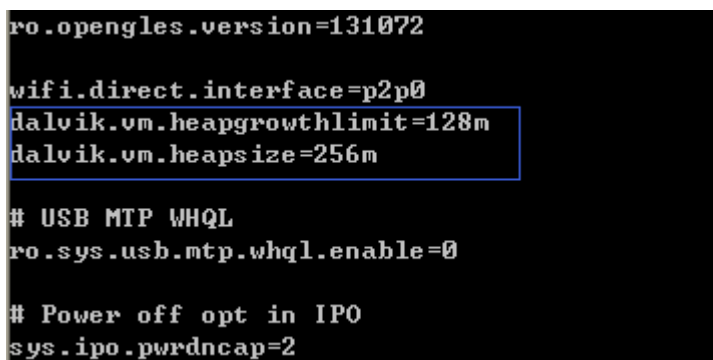


```
C:\WINDOWS\system32\cmd.exe - adb shell
C:\Documents and Settings\alisa>adb shell
shell@android:/ $ su
su
shell@android:/ # cat /system/build.prop
cat /system/build.prop
# begin build properties
# autogenerated by buildinfo.sh
ro.build.id=JDQ39
```

图 3-3 获取内存阈值

其中，adb 命令表示 PC 已登录到 Android 手机中的 Linux 环境下，可以通过输入一些 Linux 命令对手机执行对应的操作，例如，最常用的“adb devices”用于查看手机连接 PC 的状态。su 代表测试人员对手机具有超级使用权限，能够对手机里面自带的一些文件进行插入、移除、复制等行为。cat 用于一次将整个文件所包含的内容显示出来。“cat /system/build.prop”表示将系统下的 build.prop 文件内容全部显示出来。

如图 3-4 所示，在默认情况下应用程序的内存阈值由 heapgrowthlimit 取值大小所决定。而在 Android 系统 3.0 以后允许通过在 manifest.xml 文件中添加“android:largeHeap=true”字段使 heapsize 生效。heapgrowthlimit 与 heapsize 的关系就好比学生在食堂打米饭，一般人的饭量不会超过三碗，而个别壮汉吃到了五碗。这里的三碗米饭就是 heapgrowthlimit 的意思，五碗米饭就是 heapsize 代表的含义。heapsize 的取值总是大于的 heapgrowthlimit 的值，某些特殊的应用可以超过最大阈值，但最大限度只能到达 heapsize 的值。



```
ro.opengles.version=131072
wifi.direct.interface=p2p0
dalvik.vm.heapgrowthlimit=128m
dalvik.vm.heapsize=256m
# USB MTP WHQL
ro.sys.usb.mtp.whql.enable=0
# Power off opt in IPO
sys.ipo.pwrndncap=2
```

图 3-4 heapgrowthlimit 与 heapsize 取值

在 Android 应用的开发过程中，如果不合理地使用内存，会导致内存泄露的问题，例如，对一张图片重复储存，其次，代码中保留了不再使用的对象引用，引起这些对象占用的内存无法及时被 GC 线程回收，最终导致程序不断地向虚拟机申请新的内存空间而造成 OOM 的风险。其中，GC（Garbage Collection）是指 Java 的垃圾回收机制，其核心思想是指：一个已创建的对象正在使用，则不被回收。如果一个对象被创建后未使用，称为垃圾对象，将会被回收机制进行回收^[39]。

引起 Java 层内存问题的另一个重要原因是代码中没有合理地复用原有数据，而是频繁申请新内存。如连续对一个对象创建很多次，但没有使用。这种依靠 GC 机制进行内存回收的方式并不可靠，并且频繁的进行内存申请会导致触发 GC 操作过多增加系统开销，会影响到应用的流畅度，尤其是在低内存场景下会产生 UI 界面无响应的情况。

3.1.3 流畅度

衡量 UI 性能的第三个关键因素是流畅度。UI 流畅度总体反映了一个 UI 界面在使用过程中是否流畅。理论上，UI 越流畅，则对应的 FPS 越高。对于应用在终端设备上运行时的 FPS 进行采集分析，可以用来检测应用实际运行的流畅度，帮助开发改进应用不流畅的问题，提升产品的用户体验。

Android 中显示系统的绘制依赖于 SurfaceFlinger 服务^[40]。一般而言，Android 应用程序的显示过程是指应用程序调用 SurfaceFlinger 服务，再把经过测量、布局和绘制后的 Surface 渲染到显示屏幕上。这里的 SurfaceFlinger 可以解释为手机上的一个大屏幕，用于 Android 系统服务管理 Android 系统的帧缓冲区。其中，Surface 表示 Android 应用程序中的每个窗口对应的一个画布（Canvas），通俗的说，Surface 为一个 Activity 所在的显示窗口。

Android 应用 UI 的显示过程包括了两个部分和两个机制。其中，两个部分是指应用侧绘制和系统侧渲染，两个机制是进程间的通信和显示刷新机制，应用测和系统侧的渲染通过进程间通信来完成。如图 3-5 所示，展示了 UI 的显示过程。

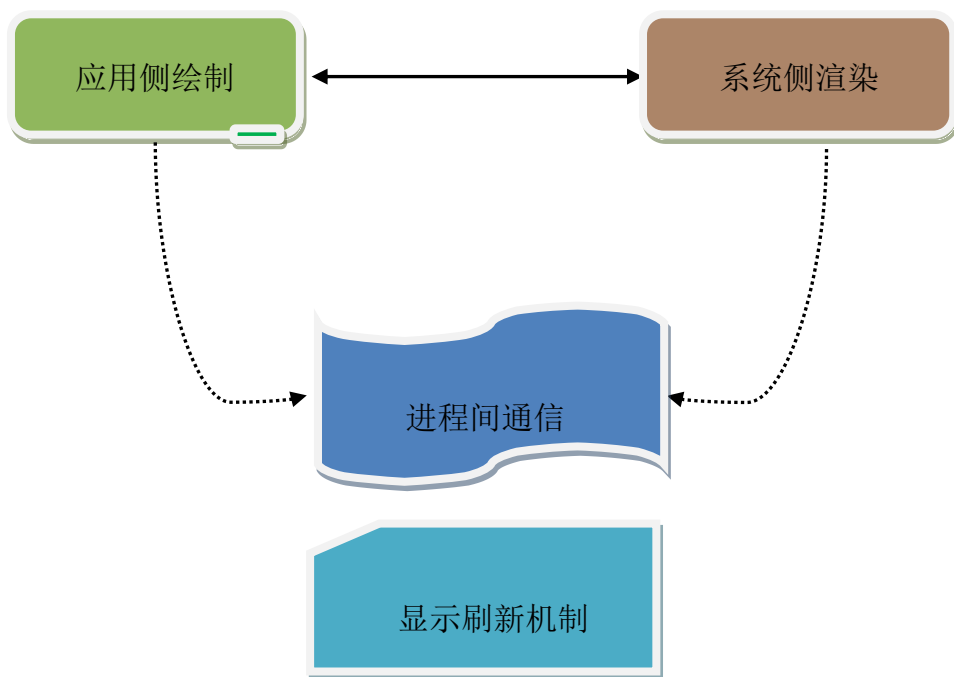


图 3-5 UI 显示中的组成部分

在 UI 的显示过程中，UI 应用侧绘制是指一个 Android 应用程序窗口里面包含了很多 UI 元素，如按钮、图片、文字，这些 UI 元素以树型结构来组织，即它们之间存在着父子关系。因此，在绘制一个 Android 应用程序窗口的 UI 之前，首先要确定子 UI 位于父 UI 的位置与大小。

总的来说，UI 应用上的渲染过程涵盖了测量（measure）、布局(layout)和绘制(draw)三个阶段^[41]。测量表示对一个 View 宽度进行确定，其中，每 View 上的控件的实际的宽与高是由它的父视图与本身决定；布局这一阶段是确定视图的位置；绘制表示在窗口中绘制所有界面元素。

3.2 UI 性能测试的方法

3.2.1 启动时延测试

测试一个 UI 的启动时延是指计算一个 UI 应用从第一次安装后，点击进入应用，待界面显示出来所花费的时间。也可以直接对一个 UI 应用清除数据后，点击启动，测试耗时。其中，对 UI 应用进行清除数据的方式分为两种。

第一种，是在“设置”菜单中，选择应用，找到被测应用后，进行清除数

据操作。这时清除了所有的缓存信息，就相当于执行完了一次卸载到重新安装的过程。

第二种方法是通过命令框去实现清除应用数据的操作。具体的命令是：“adb shell pm clear 被测应用包名”，其中，pm 是 package manager 的简称，表示对设备上的应用软件进行管理，如清数据、安装应用、卸载等等。

将一个应用恢复到初始化状态后，接着进行具体的启动时延测试。一般而言，启动时延测试方法分为眼看、秒表法和 DDMS 工具中的 LogCat 视图分析法。

传统秒表测试时延的方法是指测试人员一只手拿着一个秒表，一只手拿着手机，然后点击设备上需要测试的 UI 应用，观察并等待 UI 界面完全显示出来时，按下手中的秒表器。此时，计算的时间就是应用的第一次启动耗时。

秒表测试法的优点就是操作比较简单，只需要有个可用的秒表器就可以展开时延测试工作了。因此，对测试人员的要求也比较低，只需要会用秒表器就可以了。其缺点就是：在测试过程，依赖人为主观因素，同时要求测试人员在执行过程中，高度集中，否则在 UI 界面弹出时，忘记了按秒表就是一次失误的操作。

UI 时延测试中的另一种方法是利用第二章的 DDMS 工具展开测试。该测试方法主要是通过 DDMS 中的 LogCat 视图去观察 UI 应用启动后进入 Activity 的时长。具体操作方式：

- 1) 首先，确保设备已连接到 PC 机上，可以通过 adb devices 命令行验证。
- 2) 接着，清除被测应用中的数据并将 PC 机上的 DDMS 工具打开。
- 3) 再次，在 DDMS 窗口中的 LogCat 视图里面，按“Clear Log”清空当前的日志并用上“Scroll Lock”键进行日志锁定。
- 4) 最后，在手机界面上点击被测 UI 应用，眼睛盯着手机界面与 LogCat 视图，等待 UI 出现时，按下鼠标。此时，锁定的日志就是界面 Activity 弹出的时间。

UI 启动时延测试方法的具体流程，如图 3-6 所示。

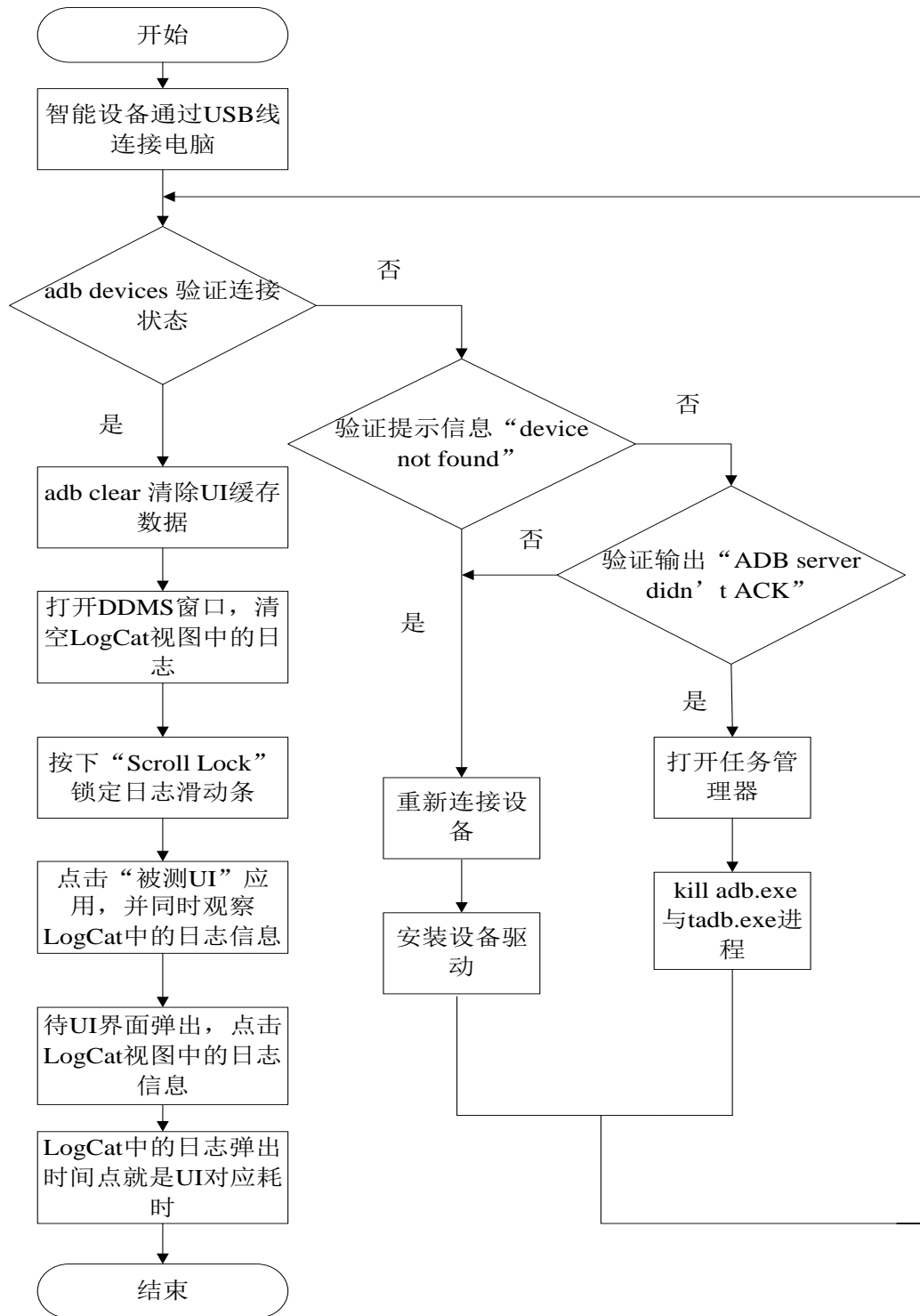


图 3-6 UI 启动耗时测试流程图

从测试启动耗时的流程图可以发现，设备通过 USB 数据线连接至 PC 机时会出现三种情况。第一种，命令行中显示设备的型号，表示设备连接成功，

可以继续执行后面的操作。第二种，弹出“device not found”信息，表示设备未找，需要将数据线拔掉，重新执行连接操作。第三种，“ADB Server didn't ACK”表示当前设备被其他端口占了，可以通过杀掉 adb.exe 或 tadb.exe 后，重新启动设备连接。

通过 DDMS 工具中的 LogCat 视图锁定日志信息后，分析启动时延的测试方法，其优点是获取时延值比较方便，并可以拓展到一些自动化工具的开发中，但其缺点是操作步骤比较繁琐，与秒表器测试方法类似，要求测试人员在执行测试时，手机与 PC 两者兼顾。

除了以上两种时延测试方法外，Android 应用测试领域还存在另外两种测试方法。

第一种是利用高速摄像机拍摄 UI 的启动过程，并利用数据挖掘的方式分析时延。该方法要求成本比较高，并要求测试人员具备一定的数据分析与挖掘方面的能力。

第二种是录屏分帧法，即用摄像头拍摄被测手机屏幕，用自动化工具或是手动操作被测手机应用，结束拍摄，并把视频拷贝到 PC 上，再运用视频解析工具分析画面中的帧数。本课题主要是利用前两种方式测试时延。

对测试 UI 启动时延方法进行归类，如表 3-1 所示。

表 3-1 UI 启动时延测试方法

	时延测试方法	优点	缺点
1	秒表计时	便捷，成本低	主观因素强，精度低
2	DDMS 工具中的 LogCat 视图分析	取值方便，可用于自动化	操作步骤繁琐，实现复杂
3	高速摄像机拍摄，数据挖掘分析时延	测试范围广，精度较高	不便捷，分析时间成本高，实现复杂
4	录屏分帧	精度很高	不便捷，分析时间成本高

3.2.2 内存测试

内存测试的主要目的是为了检验一个 UI 界面是否出现内存泄露的问题。当一个 UI 应用软件的内存值突然暴增时，则该应用会出现 Crash 或 ANR 异常。应用 Crash 是指一个 UI 界面在运行过程中退出主进程，终止运行。

ANR (Application Not Responding) 是指应用程序在运行过程中弹出的一个“应用程序无响应”的对话框, 该对话框有两个选项, 一个是“等待”, 一个是“强制关闭”, 用户可以根据自己的需求选择对应的操作。大多数用户对这个对话框都比较敏感, 因此, 应该尽量避免弹出此对话框。

一般情况下, 测试 UI 内存有两种方法。一种是通过 DDMS 工具中的 Heap 视图计算内存值, 另一种是 `dumpsys meminfo` 命令行的方式。

其中 Heap 视图, 如图 3-7 所示, 在 Heap 视图中查看内存的步骤:

- 1) 启动 DDMS 工具, 并在“Window”菜单下的“Show View”中, 选择“Heap”视图。
- 2) 用 USB 数据线将设备与 PC 相连, 并在手机中的开发者选项中, 选择 USB 调试模式。
- 3) 连接上后, DDMS 窗口里面会显示设备相关信息, 包含设备型号, Android 系统, 打开的应用进程等。
- 4) 在线程窗口中选择被测试的某类应用。
- 5) 挂上监听 heap 的标志 (即点击绿色的小桶)。
- 6) 点击右边的“Cause GC”, 进行刷新内存值操作。

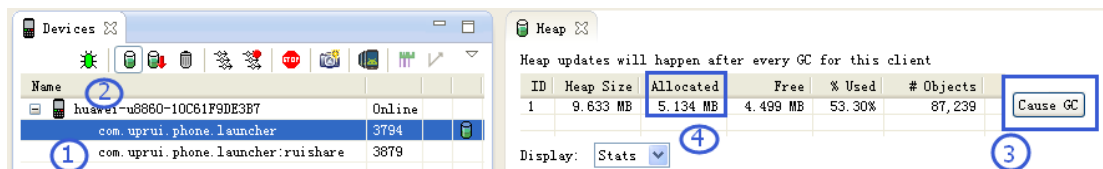


图 3-7 Heap 视图

图 3-7 中标注的①、②、③、④代表了查看应用内存值的四个步骤, 即列举了如何查看一个具体应用的内存值。其中, Allocated 代表 UI 已持有的内存值, 如果它超过了进程内存阈值, 或者突然剧增, 而且值不降下来, 就可能会产生 OOM。查看进程最大内存值的方式, 打开 CMD, 进入 adb shell, 输入“`cat /system/build.prop`”。其中, `heapgrowthlimit` 就是最大内存阈值。知道了进程阈值, 就可以很好的感知 Allocated 在程序运行时刻的危险值。通常情况下都要求产品运行在一个相对轻量的内存级别下, 可能会有一个百分比, 一般是小于 75% 进程内存阈值, 所以需要时刻关注此值变化。再次点击绿色小桶符号, 即解除 heap 监控了。需要注意的是, 在挂上监控 heap 标志会影响 UI 的流畅度, 所以,

不建议流畅度和内存测试同一时间进行，或用使用同一台设备测试。

第二种测试 UI 内存的方式是通过命令行“`dumpsys meminfo 被测应用包名`”的方式来查看 UI 应用的 meminfo 信息。使用命令行的方式测试 UI 内存的步骤，如下：

- 1) 首先，在命令行窗口中输入“adb shell”。
- 2) 进入 Android 手机中的 Linux 系统环境下，接着键入查看内存的“`dumpsys meminfo`”命令行，回车。
- 3) 最后，对 meminfo 窗口进行分析。可以看出 `dalvik allocated = 5238/1024 = 5.11MB`，与 DDMS 工具中 Heap 视图下的 Allocated 值 5.092 非常相近。
- 4) 与 heap 视图类似，当命令行中 `dalvik allocated` 的值超出了最大阈值就会出现内存泄露（OOM）的问题。

如图 3-8 所示，通过命令行“`dumpsys meminfo 被测应用包名`”，查看 UI 的内存值信息。

```

dumpsys meminfo com.uprui.phone.launcher
Applications Memory Usage (kB):
Uptime: 71571642 Realtime: 182316859

** MEMINFO in pid 25726 [com.uprui.phone.launcher] **
      native  dalvik  other  total
size:   23628   10183   N/A   33811
allocated: 19674   5238   N/A   24912
free:    2421   4945   N/A    7366
<Pss>:   7205   1954   18253  27412
<shared dirty>: 620   1852   5232    7704
<priv dirty>:  7184   1092   17684  25960

```

图 3-8 dumpsys meminfo 窗口

从 `dumpsys meminfo` 窗口中可以看出，`size` 表示历史分配的最大值，`allocated` 指当前占用的内存值，`free` 是剩余可用的内存。`Pss` 值，相当于用户在手机设置菜单中看到的进程占用内存大小的值。`shared dirty` 表示应用进程间共享的用于写操作复制的内存，`priv dirty` 是进程中不能被直接覆盖的内存，该进程不与其它进程共享。当进程结束，这部分内存马上被释放。`native` 指应用进程的堆内存，对应的内存映射名字叫“Heap”。

Dalvik 是 DVM (Dalvik Virtual Machine) 位于操作系统提供的虚拟内存管理机制之上, 通过设备文件内存映射后, 得到的内存空间 (安卓系统匿名共享内存机制 `ashmem`)。然而, 另外一套实现内存分配机制, 是把这块空间分配给 java 程序使用, 动态生成 java 对象。这一块空间是可以被多个 DVM 共享使用的。Other 指其他的一些指标, 比如说安卓属性系统服务和安卓进程间通信 binder 机制所用的设备文件内存映射。total 表示所有总和。

测试 UI 内存泄露的方法除了以上两种外, 还包括通过 MAT 工具解析抓取内存快照的 dump 文件。这种方法经常用于开发人员调试自己编写的 Android 应用, 观察内存变化情况。dump 文件是通过 DDMS 中的 Heap 按钮捕获到的内存文件。另外一种是利用业界已经存在一些已经开发好了的内存测试工具进行测试, 比如 APT、Emmagee 等等。

3.2.3 流畅度测试

帧率 (Frame rate), 是用来计算一个 UI 上每一秒可以显示多少帧数, 其测量单位为每秒显示的帧数。FPS 表示图形处理器的缓存每秒钟能否刷新的次数, 确切的解释为“每秒钟填充图像的帧数”^[42]。一般来说, 每秒显示的帧率越高, 则表示越流畅。在 Android 应用的测试过程中, 将 FPS 作为手机页面流畅度的衡量指标。

在 UI 流畅度测试过程中, 人眼是最直观的, 界面卡不卡一目了然, 但是人工去操作就会存在一些弊端。首先, 对测试结果无法量化: 一个 UI 在界面与界面之间变换时, 无法估量界面出现卡顿的程度。其次, 无说服力: 测试人员将报告发出来给开发人员, 开发人员觉得不卡, 但测试人员认为很卡。再次, 工作量大: 一两个软件应用版本还好, 但是当有很多个版本需要去测, 比如多个正式版本和验证版本, 那么测试人员估计要一直滑到手软。

UI 流畅度测试的目标是: 在用户各种实际使用的场景下, 采集用户操作 UI 界面时(主要是滑动)的 FPS 值, 然后通过 FPS 值来衡量应用的流畅度。常用的流畅度测试方法有: GPU 过度绘制检查方法和通过 adb 命令中的 `dumpsys SurfaceFlinger` 方法获取 Jank 值两种方法。除此之外, 就是利用已经开发成熟的流畅度测试工具进行测试。

(1) GPU 过度绘制检查方法

GPU 过度绘制, 可以推断出一个 UI 的布局情况, 一个 UI 界面嵌套的布局

越多，绘制的越多。无色，表示没有过度绘制，每个像素绘制了 1 次。蓝色，表示每个像素绘制了 2 次。大片的蓝色是可以接受的。如果整个窗口是蓝色的，可以尝试优化减少一次绘制。绿色，表示每个像素绘制了 3 次。如果绿色覆盖的面积不大，也是可以接受的，但应该尽量对绿色区域进行优化。浅红，表示每个像素绘制了 4 次。小范围的浅红可以接受。暗红，表示每个像素绘制了 5 次或者更多。

UI 上重绘次数越多，意味着运算越多，帧渲染时间越长，进而引起界面卡顿，不流畅。深红色区域严重影响了 UI 帧渲染时间长的性能，需要优化，避免深红色区域的出现。下面列出了详细的步骤：

- 1) 系统版本要求：需要 Android4.1 以上版本。
- 2) 在手机的“设置”菜单中，选择“开发者选项”，打开“显示 GPU 过度绘制”与“强制进行 GPU 渲染”开关。
- 3) 在设置时，如果被测 UI 应用已经打开，需先杀死 UI 对应的进程。
- 4) 然后通过界面的颜色判断界面重绘的严重程度。

如图 3-9 所示，打开 GPU 过度绘制开关按钮。



图 3-9 “显示 GPU 过度绘制”的设置窗口

(2) 通过 `dumpsys SurfaceFlinger` 获取 Jank 值

与 FPS 相对的一个词是 Jank。Jank 是衡量一个界面在滑动过程是否出现丢帧、卡顿的标准。丢帧的原因是在绘制过程中，CPU 突然被最高优先级的进程调用，执行其他操作，错过了绘制时间，就导致了丢帧的情况。

Android 业界内，设定了一个界面流畅度的标准，当 $FPS \geq 60$ ， $Jank \leq 5$ 时，UI 比较流畅；当 $FPS \geq 30$ ， $Jank \leq 7$ 时，UI 的流畅度可以接受。

SurfaceFlinger 服务是 Android 的系统服务，负责管理 Android 系统的显示帧缓冲信息，Android 应用程序通过调用 SurfaceFlinger 服务将 Surface 渲染到显示屏上。具体操作如下：

- 1) 针对 Android4.1 以上版本。
- 2) 输入“adb shell”，进入 PC 与手机的交互模式。
- 3) 调用 `dumpsys SurfaceFlinger --latency-clear`，清空帧缓冲信息。
- 4) 调用 `dumpsys SurfaceFlinger -latency<layer-name/activity-name>`，列出指定 UI 层最近 128 帧的渲染，显示时间戳信息。结果显示，如图 3-10 所示。

```
16728003
13198521517787 13198554530710 13198555539480
13198538156095 13198571238326 13198572389403
13198554924018 13198587954326 13198589175172
13198571685249 13198604662480 13198606038787
13198588422326 13198621368095 13198622453480
13198605034556 13198638050864 13198641483634
13198621796249 13198654741633 13198656092480
13198639629787 13198671454018 13198674470326
13210526195019 13210555310791 13210557091557
13210571082942 13210586862173 13210589295403
13210590907096 13210620463865 13210621405711
13210615136173 13210637261865 13210639031711
13210627268096 13210653654711 13210655536250
13210642201942 13210670345942 13210671247019
13210656544711 13210687034634 13210690336404
13210672298019 13210704045865 13210705056557
13210692303327 13210720467250 13210723048865
13210706907865 13210737443327 13210740860865
13210735977173 13210753896173 13210755853250
13210745870250 13210770582635 13210771632173
13210756431019 13210787296711 13210790874788
13210791653788 13210804023404 13210806030096
```

图 3-10 128 帧渲染的时间戳

从帧渲染的时间戳截图可以看出，`dumpsys` 命令返回的是以纳秒为单位的时间戳。第一行是屏幕的硬件特性刷新周期，当前为 16.72ms。即为 VSYNC（垂直同步刷新）的时间间隔。接下来的每一行，一共 128 行，为每一帧的 3 个不同时间点的时间戳，分别记为 A、B、C 列。其中：

A 列表示应用程序开始绘制这一帧的时间戳；

B 列是 SurfaceFlinger 开始进行垂直同步刷新该帧到屏幕上的时间戳；

C 列是指 SurfaceFlinger 完成垂直同步刷新该帧到屏幕的时间戳。

通过在 B 列与 C 列的基础上，比较相邻两次 SurfaceFlinger 开始进行垂直同步刷新该帧到屏幕的时间戳的时间差，若时间差大于 100ms 则，Jank 数加 1，统计出 Jank 数。

（3）GT 工具测试流畅度

GT 工具是腾讯公司的一款测试工具。该工具开发完成后首先在公司内部使用，接着在与公司合作的一些项目中使用，最后，在该工具的成熟阶段开始对外界用户开放使用。通过收集各大市场上对 GT 工具的下载量与用户评价，可以看出 GT 工具是一款值得用户信赖的性能测试工具。

GT 是测试行业比较强大的一款性能测试工具，主要针对一款具体的 UI 做各种性能指标的测试。这里主要用它来采集 UI 滑动时的 FPS，测试 UI 流畅度。具体过程如下：

- 1) 在应用软件市场中下载 GT 工具并安装到测试手机上。
- 2) 安装完成后，启动 GT。
- 3) 打开 GT 里的插件标签，选择被测 UI 对应的进程名。
- 4) 在“参数”栏中，选择“编辑”，将 FPS 参数拖动到“已关注的参数”栏中。
- 5) 开始测试，点击红色的按钮，开始记录流畅度数值。
- 6) 手工滑动被测 UI 界面。
- 7) 待测试完成后，点击红色按钮，停止记录流畅度数值。
- 8) 将悬浮窗上的录制开关关闭。
- 9) 可以将最终的测试数据保存到 sdcard 上。从手机中导出来测试数据，分析具体的 FPS 值。

三种流畅度测试方法归类，如表 3-2 所示。

表 3-2 UI 流畅度测试方法

	方法	优点	缺点
1	GPU 过度绘制检查方法	便捷，操作简单，根据界面颜色进行判断	部分手机没有 GPU 过度绘制功能
2	Dumpsys SurfaceFlinger 获取 FPS 值	取值方便，可用于自动化	对取到的值计算过程比较复杂，人工滑动 UI
3	GT 工具测试流畅度	便携，测试范围广，分析成本低	人工滑动 UI, 测试的 FPS 值波动比较大

3.3 启动时延测试方法的改进

3.3.1 改进的思路

常规的时延测试方法是打开 DDMS 工具，同时观察手机与 LogCat 窗口，当点击应用后，等待 UI 界面弹出时，观察 LogCat 视图中输出的日志信息。此操作相对比较复杂，要同时兼顾手机端与 PC 中的 LogCat 视图。如果在 UI 界面弹出时，没有锁定弹出的日志信息，在折回去找弹出时刻的日志就好比大海捞针。因为手机端的后台进程在不停的运转，LogCat 视图中也随之不断输出日志信息。

一个 Activity 启动时间是从“Start”开始到“Displayed”显示的过程，并通过 Intent 机制来实现。Start 表示即将开启一个 Activity，待该 Activity 完全出现时，所消耗的时间就是 UI 的启动时间。在 Displayed 这条日志后面紧跟着一个时间值，该值也表示一个 Activity 显示时间，但这个显示时间与整个启动过程有个响应延迟值。响应延迟是因为在启动过程中，应用的主进程执行了其他操作，导致的时间差。如表 3-3 所示，是 Intel 公司规定的响应延迟时间，一般响应延迟不超过 500ms。

表 3-3 Intel 公司时延延迟标准^[43]

名称	最佳	好	可接受
响应延迟	≤100ms	≤200ms	≤500ms
图形动画	≥120fps	≥60fps	≥30fps
视频回放	≥60fps	≥30fps	≥20fps

计算一个 UI 时延的方法有两种：一种是忽略响应延迟，直接计算 Displayed 后面的显示时间，即为显示耗时。另一种是包含响应延迟，计算 Start Intent 出现的时间到 Displayed 的时间差。后一种计算方法与用户感观是比较一致的时间。

改进思路：将两种方法的实现改为不依赖于 DDMS 工具。即在测试时延过程中，可以不打开 DDMS 窗口，通过命令行的方式抓取被测 UI 的主进程日志。

(1) 通过命令行的方式实现。

抓取 log 的命令行：adb logcat -v time -b main -b system -b events >D:\alisa\log.txt，这条命令行的意思是抓取主线程的日志信息并保存到 D 盘的 alisa 文件下的 log 文本文件中。可以在命令行中输入“adb logcat --help”，查看对应参数含义，如图 3-11 所示。

```
C:\Documents and Settings\alisa>adb logcat --help
Usage: logcat [options] [filterspecs]
options include:
  -s          Set default filter to silent.
              Like specifying filterspec '*:s'
  -f <filename> Log to file. Default to stdout
  -r [<kbytes>] Rotate log every kbytes. (16 if unspecified). Requires -f
  -n <count>   Sets max number of rotated logs to <count>, default 4
  -v <format>   Sets the log print format, where <format> is one of:

              brief process tag thread raw time threadtime long

  -c          clear <flush> the entire log and exit
  -d          dump the log and then exit <don't block>
  -t <count>   print only the most recent <count> lines <implies -d>
  -g          get the size of the log's ring buffer and exit
  -b <buffer>  Request alternate ring buffer, 'main', 'system', 'radio'
              or 'events'. Multiple -b parameters are allowed and the
              results are interleaved. The default is -b main -b system.
  -B          output the log in binary
filterspecs are a series of
  <tag>[:priority]
where <tag> is a log component tag (or * for all) and priority is:
  U    Verbose
  D    Debug
```

图 3-11 查看 adb logcat 帮助文档图

其中，命令行中的，“-v time”表示打印出每条 log 对应的时间，参数“-b”表示加载一个日志缓冲区，默认是 main，后面可以添加与系统相关、与事件相关的参数信息。

接着在保存的日志文件中，按快捷键“Ctrl+F”查找 Display 被测应用包名与 Activity 的信息。该日志后面的时间属于没有延迟的耗时，如果计算包含延迟的时间，就需要查找 Start Intent 这条 log 输出的时间点，将两个时间点相减

即为启动时间了。

如图 3-12 所示，展示了命令行抓取 UI 日志信息与分析启动耗时的具体流程。

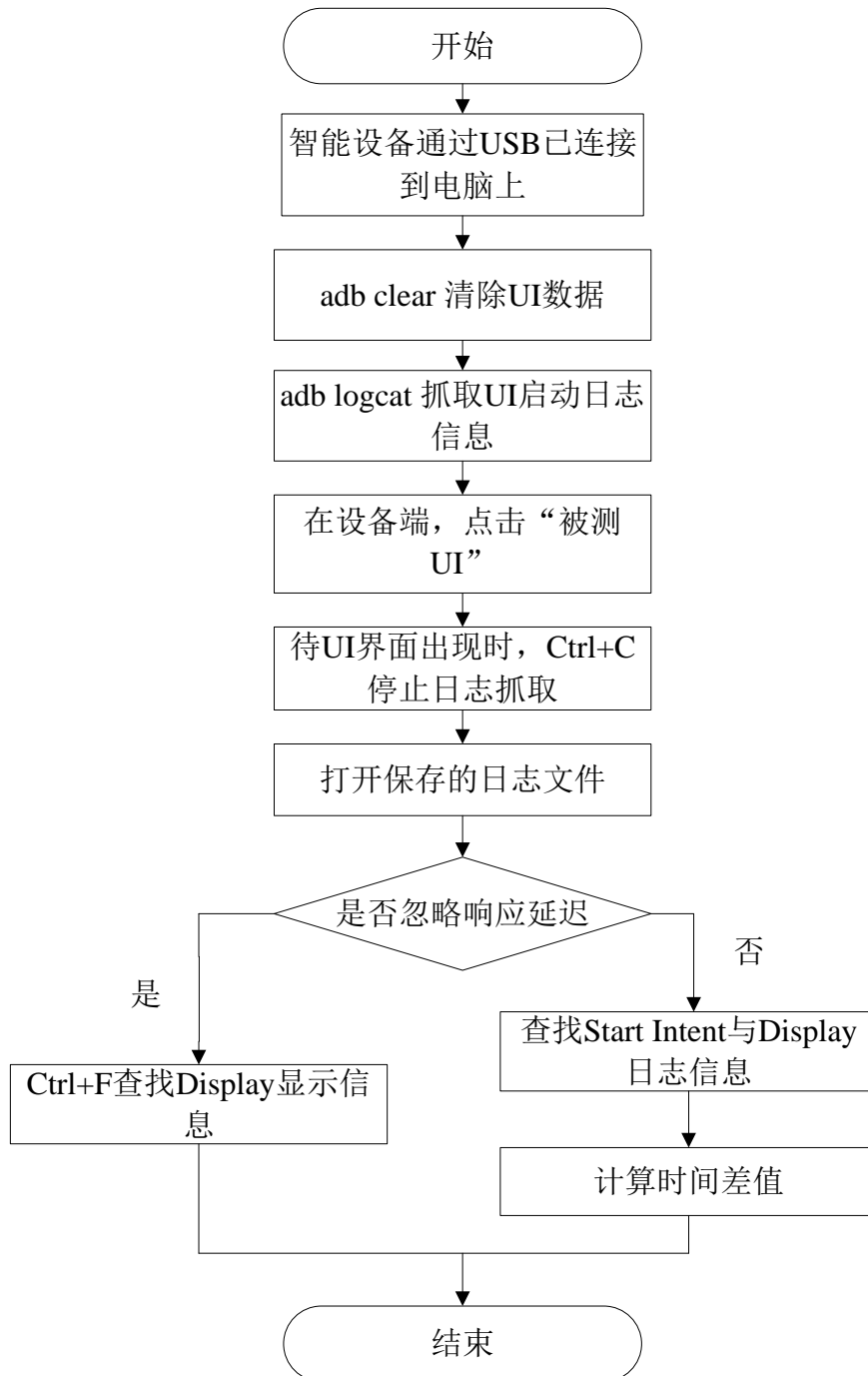


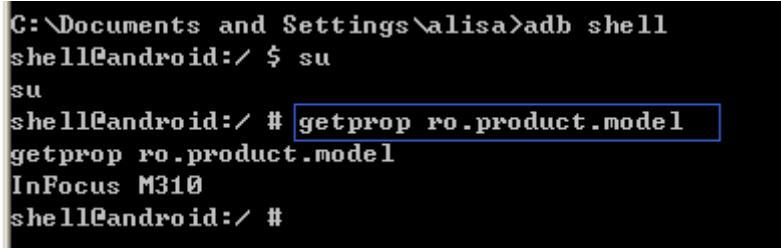
图 3-12 adb 命令测试启动时延的步骤

(2) 将命令行提升到脚本代码中实现。有利于提高测试的工作效率,减少重复手动操作。

首先,定义一个基类,用于初始化设备运行环境,包括:申请测试设备、安装被测 UI 应用、设置测试用例执行的优先级与测试用例 log 的保存路径。其中,申请测试设备包括判断当前设备的连接状态,获取设备型号、设备 id 等等。其中,查看设备型号的代码如下:

```
def get_device_module(self):
    module = self.run_shell_cmd('getprop ro.product.model')
    brand = self.run_shell_cmd('getprop ro.product.brand')
    if module.find(brand) >= 0:
        return module
    return '%s%s' % (brand, module)
```

获取设备型号与设备名称的命令是经过封装后的 adb 命令,执行这类命令可以直接操作设备,与 cmd 窗口输入的功能类似,如图 3-13 所示。



```
C:\Documents and Settings\alisa>adb shell
shell@android:/ $ su
su
shell@android:/ # getprop ro.product.model
getprop ro.product.model
InFocus M310
shell@android:/ #
```

图 3-13 获取设备型号的命令

接着,定义一个 Logcat.py 文件,用以实现日志的监控与捕获。具体流程:首先,启动一个 Logcat 监控器来监控整个应用 UI 启动过程输出的 log 信息,接着,等待 UI 启动完成后,结束 Logcat 的监控,对所有监控到的日志信息进行分解保存到日志文件中。启动 Logcat 日志监控与在命令行抓取日志信息输入的命令一样。其代码如下:

```
def start(self):
    self.device.adb.start_logcat(self.process_list, '-b main -b system -b events')
    time.sleep(1)
    self.running = True
```

在 start()方法中,调用 start_logcat()方法,并且传递 process_list 与“-b main -b system -b events”两个参数。其中,process_list 表示监控的进程列表。当列

表为空时，监控所有进程。此时，如果将 `process_list` 设置为 `system_server` 值，代表的意思是保存所有的 Android 系统的核心服务进程日志，例如 `ActivityManager` 日志等。`time.sleep(1)`方法，表示当前设备停止一秒后执行下一个操作。

启动日志监控时，设备的运行属性 `running` 赋值为 `True`。如果需要结束 Logcat 日志监控时，将其属性值改为 `False`。接着，利用正则表达式实现字符串的匹配功能^[44]。为了方便与输入的日志进行正则匹配，将监控到的日志信息分解为各个字段并将分解的 logcat 信息保存。日志的保存方式与文本保存类似，包括在指定目录下创建一个以“.log”为后缀的文件，打开新建的文件，将日志信息写入进去。

如图 3-14 所示，描述了整个 logcat 实现的功能。

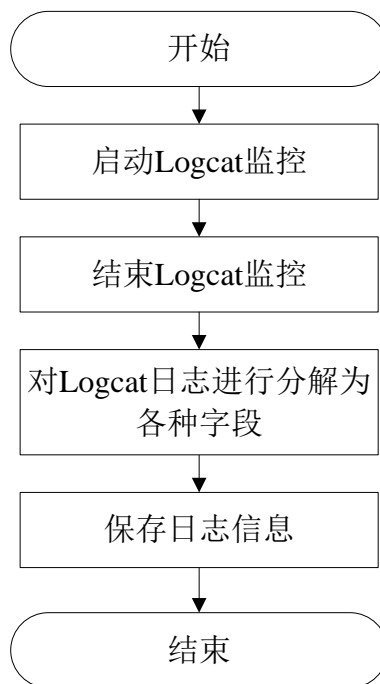


图 3-14 logcat 文件实现的功能

按照固定格式对日志信息进行分解，分解后的日志信息方便脚本用例调用，其具体实现代码如下：

```

class LogcatSplit(object):
    REGX_PROG = re.compile(r'^\[ (?P<process_name>.+?)\] \[ (?P<date_time>.+?)\] (?P<level>.+?) / (?P<tag>.+?) \((?P<tid>\d+?)\) : (?P<content>.*?)$')
    def __init__(self, log_line):
        """将 Logcat 日志分解成各种字段，供脚本使用"""
        self.log_line = log_line
        obj_fields = self.REGX_PROG.match(self.log_line)
        if obj_fields is None:
            logger.error(u'log line not match, plz correct REGX_PROG')
            logger.error(self.log_line)
        else:
            self.fields = obj_fields.groupdict()

```

其中，REGX_PROG 为正则表达式，表示日志信息以该正则表达式的形式输出。输出的结果包含主进程、时期、日志级别、进程的编号和日志内容。例如：“[system_server] [2015-03-16 13:19:58.911] I/ActivityManager(548): Displayed com.uprui.phone.launcher/com.rui.phone.launcher.Launcher:”就是一条标准的分解后的日志信息。

从这条信息可以看到 process_name = “system_server”，date_time = “2015-03-16 13:19:58.911”，level = “I”，tag = “ActivityManager...”，tid = “548”，冒号后面的内容都属于 content 的内容了。

一般来说日志级别（Level）包括 V、D、I、W 与 E。V（Verbose）表示输出的日志信息是最完整的。D（Debug）即调试信息。I（Info）指一些设备的一些通告信息。W（Warn）警告信息。E（Error）错误信息，也是开发人员与测试人员在应用软件的开发过程中最需要关注的信息。错误信息是提示导致 UI 崩溃的信号信息。

最后，完成了以上两步，就可以编写脚本 Case 调用基类，实现设备的连接操作，测试环境的初始化工作等等。接着，启动被测 UI 的 Activity，并通过 Logcat.py 中的方法捕获整个启动过程中的所有日志。最后，传递两个参数，即 Start 开始启动时的日志信息与界面显示出来时的 Displayed 日志信息，求出时间差。即为启动的耗时。

如表 3-4 所示，列出了脚本中的一些常用属性。

表 3-4 脚本中的常用属性

	属性	值	说明
1	owner	owner = 'alisa'	标识编写测试脚本的作者
2	priority	priority = 'High'	设置脚本的优先级
3	timeout	timeout = 2	设置超时时间为 2 分钟
4	config	config=OrderedDict()	引入一个键值对的有序集合体，用于存放日志信息
5	config['begin']	config['begin'] = ur'Start.*?com.rui.phone.lau ncher.Launcher'	将启动时的日志赋值给 config['begin']
6	config['end ']	config['end'] = ur'Displayed.*?com.rui.pho ne.launcher.Launcher'	界面显示时输出的日志赋值 给 config['end']
7	logcatmon	logcatmon = Logcat(self._run_device.dev ice_id, ['system_server', 'com.uprui.phone.launcher'], **regx_config)	调用 Logcat 方法，并传输三 个参数，一个是设备的 ID 号， 一个是日志进程名和包名
8	clear	self._run_device.clear_data('com.uprui.phone.launcher')	调用 clear_data()方法，用于 清除被测 UI 数据
9	start()	self.start()	脚本开始执行阶段
10	start_activity	self.device.adb.start_activity ('%s/%s' % ('com.uprui.phon e.launcher', 'com.rui.phone.launcher.Lau ncher'))	启动被测 UI 应用
11	wait_for_log	self.logcatmon.wait_for_log (ur'Displayed com.uprui.phone.launcher/c om.rui.phone.launcher.Laun cher', timeout=60)	等待最后一条日志出现，后 结束日志的抓取。调用 stop() 方法与 save()方法
12	duration	log_end() – log_begin()	计算 config['end']与 config['begin']的时间差

3.3.2 改进后的优点

(1) 不需要打开 DDMS 工具，直接在命令框中输入抓取 log 的命令，手动点击被测 UI，等待 UI 界面完全显示出来时，再在命令窗口中按“Ctrl+C”停止日志的抓取。对保存的日志进行一个查找统计的过程，就能计算出一个 UI 启动所消耗的时间了。

(2) 自动化脚本工具测试 UI 启动耗时主要是为了提高测试效率。启动时延可以通过在脚本中抓取一个 UI 启动过程中的日志信息，再用正则表达式匹配启动与完成输出的 logcat 信息。自动化脚本编写完后，可以重复使用，同时代替了手动点击启动 UI 的操作过程。

(3) 自动化脚本测试 UI 启动时延的方法提高了测试结果的精确度，解决了传统秒表法的缺陷，即因为测试人员疲劳测试导致测试结果数据的波动与误差。在时延测试过程中，测试人员同时兼顾手机与 LogCat 视图，很容易出现漏点、误点的操作，同时反复的过程也会让测试人员疲倦、产生懈怠的情绪，进而影响最终测试结果。

(4) 脚本适用范围广。该方法不仅仅可以测试一个 UI 的启动耗时，还可以计算一个 UI 界面跳到另一 UI 界面所花费的时间，但前提需要对被测应用的源码熟悉，在代码中添加表示一个 Activity 的启动与结束的相关日志信息。再通过日志分析捕获开发人员在源码中打印出的 log 信息，从而计算两个 Activity 中间花费的时间。

3.4 流畅度测试方法的改进

3.4.1 改进的思路

目前，业界内已经存在一些 Android 系统 UI 性能测试工具，其中，比如 Emmagee 和 GT 两款工具。Emmagee 主要用来监控被测应用软件在使用过程中占用设备的内存、CPU、流量。而 GT 工具比较强大，除了 Emmagee 所包含的功能外，GT 工具还能监控到被测应用在使用过程中的 FPS 值。同时，GT 工具占用内存小，在使用过程中不会影响到其他应用的性能。

GT 工具虽然可以帮助测试人员统计滑动 UI 时的 FPS 值，但是，该工具采用的是手动滑动界面，导致采集到的 FPS 值波动比较大。因为每个人滑动的距离、力度、时间间隔不一样，这些因素都会影响到 FPS 的数值。为了提高测试

结果的精确性，以保证测试一致性和可重复性，在 GT 工具的基础之上做了改进，即：采用自动化的方式滑动 UI 界面，统计对应的 FPS。

改进思路：利用 Android 系统自带的工具 MonkeyRunner 编写脚本，实现自动点击 GT 工具，启动 FPS 监听功能，并连续滑动被测 UI 界面，采集对应的 FPS 值，最后，滑动结束后暂停 GT，结束整个 FPS 值的录制。其中，连续滑动 UI 界面，表示滑动的频率，力度都是一样的，而且滑动多少次都是可以预先设定。在滑动的过程需要模拟用户一样滑动屏幕，此时，会产生一种加速度。加速度与对应的滑动距离、时间、滑动的步长相关联。因此，滑动过程中对应的参数选取比较重要。如图 3-15 所示，列出了自动化操作的流程。

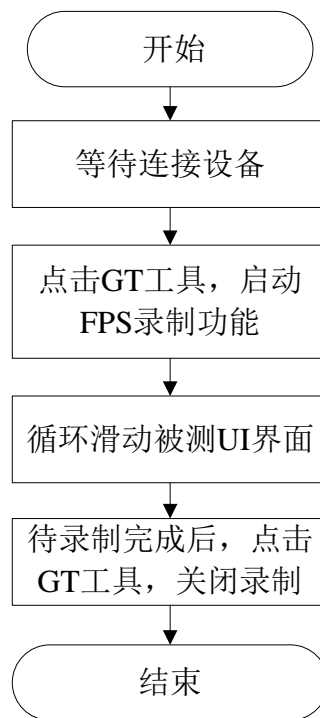


图 3-15 MonkeyRunner 自动滑动 UI 采集 FPS 值

MonkeyRunner 自动滑动 UI 采集 FPS 值，对应的核心代码段如下：

```
from com.android.monkeyrunner import MonkeyRunner
from com.android.monkeyrunner import MonkeyDevice
#滑动 UI
def drag_rui():
    device.drag((600,593),(177,593),0,30)
    MonkeyRunner.sleep(0.5)
```

```

#连接设备，并启动 FPS 录制功能
if __name__=='__main__':
    device=MonkeyRunner.waitForConnection()
    MonkeyRunner.sleep(1)
    device.touch(426,836,'DOWN_AND_UP')
    #假设滑动 20，采集 FPS 值
    for i in range(20):
        drag_rui()
    device.touch(426,836,'DOWN_AND_UP')
    MonkeyRunner.sleep(0.7)

```

MonkeyRunner 工具包含三个模块：MonkeyRunner、MonkeyDevice 和 MonkeyImage^[45]。MonkeyRunner 模块表示连接设备，可以实现等待 PC 与手机进行通信，并在通信的过程中执行休眠，唤醒屏幕等操作。MonkeyDevice 则表示对连接上的设备执行一些操作，比如，滑动、触屏、双击。MonkeyImage 则主要表示通过 PC 端对设备进行截屏。

在自动滑动 UI，采集 FPS 值的脚本中主要用到 MonkeyRunner 与 MonkeyDevice 两个模块的功能。首先，在 main 函数中调用 waitForConnection() 方法，用于唤起设备的连接。等待一秒后调用 MonkeyDevice 下的 touch() 函数，实现启动 GT 工具的 FPS 值录制功能。该函数有三个参数，前二个参数表示点击位置的起始 X 轴坐标值与 Y 轴坐标值，第三个参数表示执行的动作是一个点击的操作。

接着，循环调用 drag_rui() 方法，实现滑动 UI 界面多次，这个次数可以在函数中预先定义。在 MonkeyDevice 模块中，函数 drag((x1,y1),(x2,y2),time, step) 方法有四个参数，(x1,y1) 表示界面上拖动的起点坐标值，(x2,y2) 是终点坐标值，time 表示拖动花费的时间，以秒计算，step 表示起点到终点的拖动过程是分多少步完成的。这四个参数的选取，是模拟人自动滑动界面的关键因素，同时，值的选取不是一蹴而就的，而是经过反复的取值、筛选、测试出来的。等待滑动结束后，暂停 GT。

3.4.2 改进后的优点

(1) 弥补了 GT 工具因人工滑动被测 UI 界面导致采集到的 FPS 值波动比较大的缺点。虽然 GT 工具是 Android 移动应用测试领域中比较强大的测试工具，但是该工具还没有实现全能的自动化操作。在统计 FPS 值的过程中，由于

每个人滑动界面的力度，频率，速度，滑动的位置不一样，造成计算的 FPS 值不精确。在 GT 工具之上进行改进，实现自动化滑动被测 UI 界面，这样滑动的幅度，频率都是同一个指标，最后测试出来的 FPS 值比较精确。

(2) 自动化脚本实现滑动 UI 界面，其滑动次数可以预先设定的。而手动滑动 UI 界面是一个随机化的过程，有时滑下来，连测试人员自己都不知道滑动了多少次。

(3) 同样的，自动化脚本滑动 UI 很多次数时，比较省力。如果需要压力测试统计 N 多次的 FPS 值，测试人员手动滑下去是不能的，手都可能滑到酸疼。使用自动化就不会出现疲劳现象，可以循环滑动。

3.5 改进后的自动化测试方法与现有方法的区别

目前，在国内，一些大型公司已经研发出了一系列的 Android 系统 UI 性能测试的自动化工具。本文在第一章节中的国内外现状分析中做了简要介绍。譬如，百度公司的 MTC 工具、腾讯公司开发的 GT 性能测试工具、网易研发的 Emmagee 工具、北京云测信息公司研发的 Testin 工具等等。

MTC 与 Testin 都属于云平台自动化测试工具。用户只需要将被测应用软件上传到该平台上，就可以做一些相应的测试工作。MTC 与 Testin 工具都可以实现 UI 的性能测试工作，并且这两个测试工具的操作方式也比较相似。其具体步骤：首先，将被测应用软件的安装包上传到对应的测试平台；接着，选择应用软件的性能测试类型；再次，选择测试设备的型号；最后，等待测试结果与查看测试报告。

本文中以 Testin 工具为例，将 Testin 工具与改进后的性能测试方法做比较，其区别有：

1) Testin 是一套完整的 UI 性能测试工具，它无法将应用软件的性能问题分开进行测试，例如，当测试人员觉察到某一款 Android 系统 UI 的启动时间有问题时，需要单独测试 UI 的启动时间，观察启动时间的数值变化。这时 Testin 工具无法满足测试人员的需求，而改进后的性能测试方法能够将 Android 系统 UI 的性能问题分开进行测试。

2) 正因为 Testin 工具每测试一次 UI 都要完成内存、流量、启动时间等等一整套的测试工作，所以花费时间比较长，测试结果不可能立马呈现出来，需要测试人员等待二到八个小时才能查看测试报告，这对测试工作的进度造成了

影响。而改进后的性能测试方法，其测试结果可以边测试边观察，这样能够帮助测试人员提前发现 UI 的性能问题。

3) Testin 测试平台上集聚了成千部不同系统，不同型号的测试设备。例如：Huawei、Samsung、HTC 等等。利用该工具能够将 UI 放置在不同型号不同设备上性能测试，而改进后的测试方法，所选择的测试设备比较有限，无法兼容到成千部设备上。

3.6 本章小结

本章首先对影响 UI 性能的因素进行详细分析，包括：一个 UI 的启动时间的过长，UI 在滑动的过程中出现卡顿、UI 占用设备的内存过大或出现内存泄露等。针对这三大类因素，提出了三种对应的性能测试手段，即 UI 启动时延测试方法、内存测试与流畅度测试。其次，对性能测试中的时延测试方法与流畅度测试方法提出改进的方案，并列出了方法改进的具体思路与改进后的优点。最后，简要论述了改进后的性能测试方法与现有的 Android 系统 UI 性能测试的自动化方法之间的区别。

第 4 章 改进 UI 测试方法的使用与结果分析

4.1 项目背景介绍

RUI 手机桌面是一款 Launcher。该桌面的特色是应用自动分类，带有实用的小插件、精品应用导航等。如图 4-1 所示，列出了 RUI 手机桌面的主要组成部分。

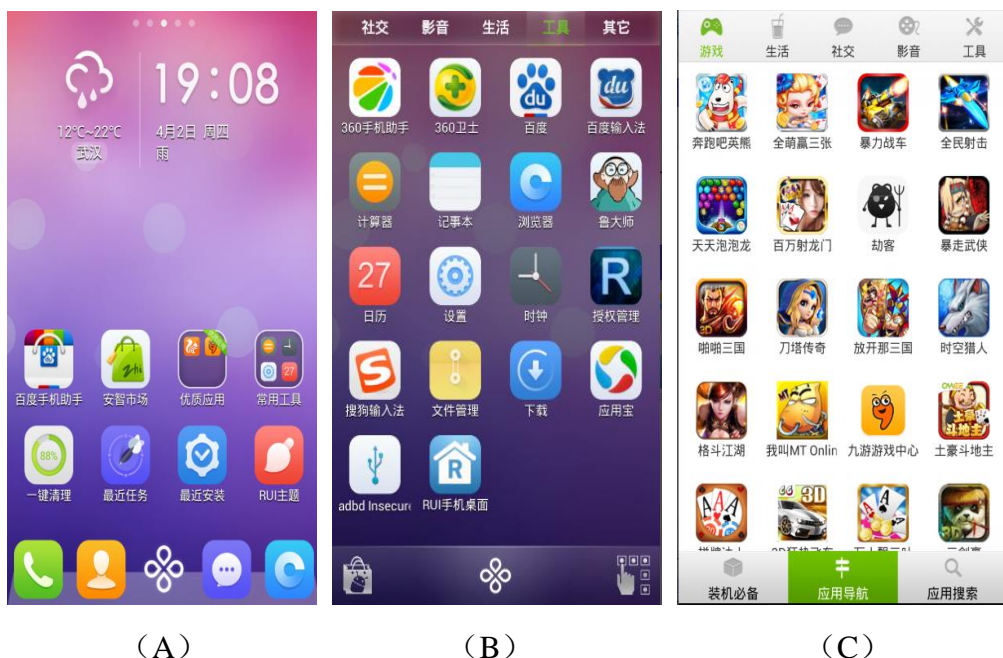


图 4-1 RUI 手机桌面的组成部分

(A) 图为 RUI 手机桌面的主界面。该界面上包含了实用的小插件，例如天气插件、一键清理、RUI 主题等等。

(B) 图为应用分类界面。RUI 手机桌面将手机上的所有应用自动分为六大类，分别是：游戏、社交、影音、生活、工具、其他。该功能方便用户快速定位到自己想要打开的应用软件。

(C) 图是 RUI 手机桌面的精品应用推荐界面。用户可以在该界面内下载到最热门的应用软件。

除了以上列出的三大功能外，RUI 手机桌面还包括手动分类、精品主题、图标智能平铺等等功能。

本文主要是针对最新版本的 RUI 手机桌面（即：V3.2.0）执行测试，包括测试 RUI 手机桌面功能是否完备、UI 启动花费的时间、内存值、流畅度等方面，并对测试结果进行分析。

4.2 需求分析与设计

4.2.1 测试需求分析

需求分析阶段的测试工作，主要有分析需求提出的背景、验证各项功能点、整理测试要点、细化每一项测试要点等等。

RUI 手机桌面的需求功能模块有：实用的小插件、应用自动分类、手动分类、精品应用推荐、应用智能平铺、在线主题、快捷开关等。

表 4-1 “天气插件”模块的需求分析测试

天气插件模块		
1	需求来源	用户习惯需求，用户打开手机必看天气预报，决定出行；
2	需求项	查看天气时方便、操作快捷、精确、省流量；
3	测试点	自动定位用户当前城市； 选择需要查看的城市； 手动刷新当前天气； 同时查看多个城市的天气预报； 设置默认城市；
4	测试项分析	需要开启“GPS”才能实现自动定位功能； 可以选择汉字输入选择查看的城市； 拼音、英文输入搜索城市，查看天气； 连接网络时可以自动刷新默认城市的天气；
5	测试项细化	例如：选择的城市细化到一个小镇上是否能搜索到；
6	测试环境设计	Android 手机，WIFI 网络或是 3G、4G 网络；
7	测试人数	1 人；

表 4-2 “应用自动分类”模块的需求分析测试

应用自动分类模块		
1	需求来源	用户需求。用户设备上的应用过多，通过自动分类功能，可以帮助用户快速定位到自己需要打开的应用软件；
2	需求项	应用分类准确；
3	测试点	对手机上的所有应用进行自动分类； 将应用分为六大类，即游戏、社交、影音、生活、工具、其他； 最新安装的应用软件，直接分类到对应的类别中； 最新卸载的应用软件，其图标在对应的分类中消除；
4	测试项分析	在是否有网络的情况下都能够对应用进行分类； 手机上的每一款应用分类准确；
5	测试项细化	例如：安装“有道词典”，该应用自动分类到工具类别；
6	测试环境设计	Android 手机，WIFI 网络或是 3G、4G 网络；
7	测试人数	2 人；

表 4-3 “精品应用推荐”模块的需求分析测试

精品应用推荐模块		
1	需求来源	用户需求。通过精品应用推荐，可以帮助用户快速下载到最热门的应用软件；
2	需求项	用户不依赖第三方应用软件下载热门应用，快捷、方便；
3	测试点	可以查找到排行榜上的应用软件； 将所有精品应用分为六大类；
4	测试项分析	需要在有网络的情况才能下载到精品应用； 排行榜上的应用都能在精品应用中找到； 精品应用分类准确；
5	测试点细化	例如：热门游戏“奔跑吧英熊”，可以在精品应用中的“游戏”类别中找到，点击图标直接下载；
6	测试环境设计	Android 手机，WIFI 网络或是 3G、4G 网络；
7	测试人数	2 人；

需求分析阶段的测试工作准备的越充分,对后期测试开展越有利。如表 4-1、表 4-2、表 4-3 所示,分别针对 RUI 手机桌面中的“天气插件”、“自动分类”、“精品推荐”功能模块,提出的测试需求分析。

4.2.2 需求评审

需求评审是在测试需求分析之后,开展的另一项测试活动。需求评审的目的是为了对需求分析进行确认,并挖掘出软件中潜藏的风险点。在测试过程中,对风险点进行规避与解决。

对 RUI 手机桌面进行需求评审时,需要关注如下几点:

1) RUI 手机桌面的需求文档是否已经完整的描述了产品的特性,例如,自动分类、精品应用推荐、天气插件、在线主题、手动分类、屏幕编辑、滑屏等功能模块应在需求文档中详细描述。

2) RUI 手机桌面的各个功能点是否兼容,一个功能点是否影响到其它功能点的实现。最常见的是 RUI 手机桌面的自动分类是否影响到用户对于一个应用软件进行手动分类。

3) 对测试人员的分配是否合理。正如测试 RUI 手机桌面中的天气插件功能模块,需求分析文档规定一人完成。在需求评审阶段应该估算测试的工作量,例如一个人是否能够顺利完成天气插件的测试工作。

4) 测试整个 RUI 项目的周期预算。一般情况下,计划一个月发布一个 RUI 版本,测试应该与应用开发同时进行。

5) 对 RUI 项目中潜藏的风险点进行评估。包括网络风险,手机断网后重新连上是否会影响到 RUI 手机桌面的自动分类功能,应用分类是否产生大流量, RUI 手机桌面是否设置缓存机制; UI 风险, RUI 手机桌面上的应用是否被重绘多次,导致用户滑动界面卡顿;兼容性风险,在各个系统的手机上是否能够正确安装 RUI 手机桌面等等。

4.2.3 测试环境的设计

测试环境部署齐全是顺利开展测试工作的前提和必要条件。在本课题中,测试环境的设计包括 PC 端与手机端两部分。

PC 端的环境部署主要包括 JDK、SDK、ADT、Python 等环境安装,手机端主要是指对手机获取 root 权限。以下分别对 PC 端与智能手机终端设备的环

境部署分别展开叙述。

（1）PC 端测试环境的配置

PC 端主要是通过 Android 调试桥（ADB, Android Debug Bridge）来管理手机。为了在 PC 端直接调用 adb 命令操作手机，需要在 PC 端做如下环境配置工作：

1) JDK（Java Development Kit）的安装与环境配置

下载 JDK 安装包，对安装包进行解压，点击安装，待安装完成后开始配置环境变量。配置环境变量的具体操作：在“我的电脑”图标上单击鼠标的右键，选择“属性”，点击菜单栏中的“高级”选项。接着单击左下角的“环境变量”。在“变量变量”窗口中，新建“JAVA_HOME”变量，将值设置为 JDK 安装的路径，再建立 CLASSPATH 变量，并将其赋值为 jdk 文件下的\lib\dt.jar 与 \lib\tools.jar 目录。最后，在 path 环境变量中添加 jdk 所在文件夹下的 lib 目录。环境变量配置完成后，在命令窗口中输入“java -version”，验证 JDK 环境配置是否成功。

2) SDK（Soft Development Kit）的安装与环境配置

下载 SDK 安装包。对安装包进行解压，然后配置环境变量。具体步骤：打开“环境变量”窗口，将 SDK 文件夹下的 platform-tools 与 tools 目录添加到 path 变量下。最后，通过命令行“adb -version”验证是否安装成功。

3) Python 工具的安装与环境配置

Python 主要是用于编写自动化脚本用例实现自动操作手机的功能。Python 环境的搭建也相对来说比较简单。首先，下载 python 安装包，解压。配置环境变量，只需要将 Python 所在的目录添加到 path 变量后面，前面用分号隔开。测试 Python 安装是否已经成功，在 cmd 窗口中输入 python，验证安装结果。

将 JDK、SDK、Python 安装完成后，接着就是一些可视化插件的安装了，比如 Eclipse、ADT、PyDev。Eclipse 插件的安装主要是在 Eclipse 官网上，选择对应的版本点击安装包进行下载，下载完成后直接解压，就可以运行了。再通过 eclipse 中的“Help”菜单，选择“Install New Software”进行 ADT、PyDev 插件的安装。

（2）手机端的环境配置。

手机上的环境部署主要是指对手机进行刷 root 操作，并获取高级用户权限。手机的高级用户权限是指用户可以对手机执行一系列系统权限的操作，例如，可以删除手机系统中自带的文件，读取手机系统文件的一些信息，将文件复制

到 PC 端上。

获取 Root 权限的具体操作步骤：下载“kingroot”工具，安装至手机，点击并打开该工具，选择其中的“一键 Root”菜单，进行手机 Root 操作，待手机 Root 完成后进行一次设备重启的操作过程。在手机 Root 成功后，就可以通过命令行“adb shell su”获取手机的高级权限了。

如图 4-2 所示，在手机 Root 成功时，命令窗口中输出的是“#”，而没有 root 的手机，输出的是“\$”。

```
C:\Documents and Settings\alisa>adb shell su  
root@android:/ #
```

图 4-2 获取手机 Root 高级权限

4.2.4 测试用例的设计

对一款 Android 系统 UI 应用软件来说，一般采用黑盒测试用例设计方法中的等价类划分法设计测试用例。

RUI 手机桌面的测试用例模块有：

- 1) 天气插件模块；
- 2) 应用自动、手动分类模块；
- 3) 精品应用推荐模块，例如，推荐应用界面中显示热门的应用软件，下载、安装应用软件等功能；
- 4) 主桌面模块，如壁纸、安装卸载应用、拖动图标、屏幕编辑、滑屏功能、智能平铺、备份还原等；
- 5) 主题模块，分为在线主题、本地主题；
- 6) 其他模块，例如切换语言、手机重启、SD 卡卸载安装后 RUI 手机桌面的显示；

RUI 手机桌面测试用例设计的完备与否，决定了后期的操作过程是否能够快速发现软件中存在的一些缺陷。

如表 4-4、表 4-5、表 4-6 所示，分别列出了 RUI 手机桌面中“天气插件”、“应用自动分类”与“精品应用推荐”功能模块的测试用例。

表 4-4 “天气插件”模块的测试用例

功能点：利用天气插件，查看天气					
有效等价类 (网络良好的情况下，查看天气)					
编号	测试目的	测试步骤	预期结果	优先级	实际结果
1	可以定位用户所在城市并显示当前位置的天气预报	1) 在“设置”菜单中开启“我的位置”； 2) 点击“自动定位”；	显示当地天气	高	
2	手动选择城市，可显示所选城市天气预报	1) 点击“城市”，打开“选择城市”界面； 2) 在热门城市中点击“北京”；	显示北京当前天气	高	
3	手动输入城市可显示所输入城市名称天气预报	1) 打开“选择城市”界面； 2) 手动输入“武汉” 3) 选中，点击“搜索”并查看结果；	显示武汉当前的天气预报	高	
4	可以同时查看多个城市的天气预报	1) 在“城市管理”界面添加“深圳”、“长沙”、“上海”、“大连”； 2) 查看城市天气	四个城市的天气情况显示在城市管理界面上	高	
5	点击天气插件图标可显示多天的天气	1) 点击天气图标； 2) 查看结果；	在天气详情界面中，显示了当前城市最近四天的天气情况	高	
无效等价类 (网络异常，查看天气)					
6	网络中断，无法查看天气情况	1) 关掉网络； 2) 点击天气图标 3) 查看天气显示结果	当前天气刷新失败	低	

表 4-5 “应用自动分类”模块的测试用例

功能点：应用自动分类					
有效等价类 (网络良好，应用自动分类)					
编号	测试目的	测试步骤	预期结果	优先级	实际结果
1	第一次启动 RUI 手机桌面，所有应用根据本地数据进行分类	1) 安装“RUI 手机桌面”； 2) 点击应用启动； 3) 查看结果；	所有应用已分类到对应的类别中	高	
2	安装新应用，它被放置在对应类别屏上	1) 下载“应用宝”，点击并安装 2) 查看“应用宝”对应的分类结果；	“应用宝”被放置到工具屏上	高	
3	同时安装多个应用，它们被放置到对应的类别屏上	1) 同时安装“全民射击”、“QQ”、“优酷”、“携程”、“轻松卸载”四款应用软件； 2) 查看每个应用软件对应的分类情况；	这些应用被分别放置在游戏、社交、影音、生活、工具屏上	高	
无效等价类 (网络异常，应用自动分类)					
4	第一次启动 RUI 手机桌面，所有应用根据本地数据分类	1) 关掉网络； 2) 安装“RUI 手机桌面”； 3) 查看结果；	所有应用已分类	低	
5	安装在本地数据库中的新应用，它被放置在相应屏	1) 安装新应用“360 卫士”； 2) 查看应用分类结果；	“360 卫士”被放置到工具屏	低	
6	安装不在本地数据库中的新应用，它被放置在其他屏	1) 安装新应用“GT”； 2) 查看结果；	“GT”被放置到其他类别的屏幕中	低	

表 4-6 “下载推荐应用”模块的测试用例

功能点：点击推荐应用图标，下载应用					
有效等价类 (网络良好，下载推荐应用)					
编号	测试目的	测试步骤	预期结果	优先级	实际结果
1	点击推荐应用界面中的某个应用图标，可下载该应用	1) 点击“有道词典”应用程序的图标； 2) 弹出下载对话框，点击“现在下载”；	通知栏显示：“RUI 提示：有道词典正在下载”	高	
2	点击正在下载的推荐应用图标，提示“正在下载”	1) 点击“有道词典”图标，通知栏中出现正在下载的提示语； 2) 再次点击“有道词典”的图标，查看结果；	弹出提示语“有道词典已经加入下载队列”	高	
3	点击多个推荐应用图标时，打开通知栏，显示 2 个在下载，其它应用软件正在等待中	1) 点击“全民射击”、“QQ”、“优酷”、“携程”、“轻松卸载”图标，打开通知栏； 2) 查看 RUI 手机桌面的提示信息；	通知栏上显示“全民射击”、“QQ”正在下载，“携程”、“轻松卸载”在等待	高	
4	查看下载进度，显示进度从 0%到 100%，表示下载完成	1)点击推荐应用图标，下载开始； 2) 在通知栏上查看进度；	下载进度依次递增，最后显示下载完成	高	
无效等价类 (网络异常，下载推荐应用)					
5	无网络情况下，点击推荐应用图标，提示“无可用网络，请点击设置”	1) 无网络情况下，点击“百度音乐”应用程序进行下载； 2) 查看该应用程序的下载结果；	通知栏提示“无可用网络，请点击设置”	低	
6	下载过程中，网络中断或网络不稳定，提示“下载失败，请点击重试”	1) 下载“百度音乐”应用程序； 2) 下载过程中断开设	通知栏上提示“下载失败，请点击重试”	低	

		备的 wifi 网络; 3) 查看应用程序的下载结果;			
7	网络信号由弱逐渐增强的情况下, 点击的推荐应用被重新下载	1) 网络环境比较差, 下载“百度音乐”应用软件; 2) 弹出对话框, 提示“下载失败, 请点击重试”; 3) 点击“重新下载”; 4) 网络连接上, 该应用软件被重新下载;	该应用开始下载, 进度从 0% 依次递增到 100%	低	

4.3 测试方法的实现

4.3.1 功能测试方法的实现

UI 性能测试的实现是在功能测试完成的基础之上展开的, 所以在测试一个 UI 的性能时, 应该进行 UI 相关的功能测试。

功能测试是根据测试用例, 一步一步执行被测应用, 发现应用中存在的缺陷。针对 Android 应用独特的使用环境而言, 可以对 UI 功能测试进一步划分, 包括安装卸载、在线升级、业务功能、交互性、适配性测试。其中, 交互性测试又划分为用户交互、软件交互; 适配性测试又分为屏幕分辨率与 Android 系统版本的适配。

如图 4-3 所示, 列出了 UI 功能测试所包含的测试点。

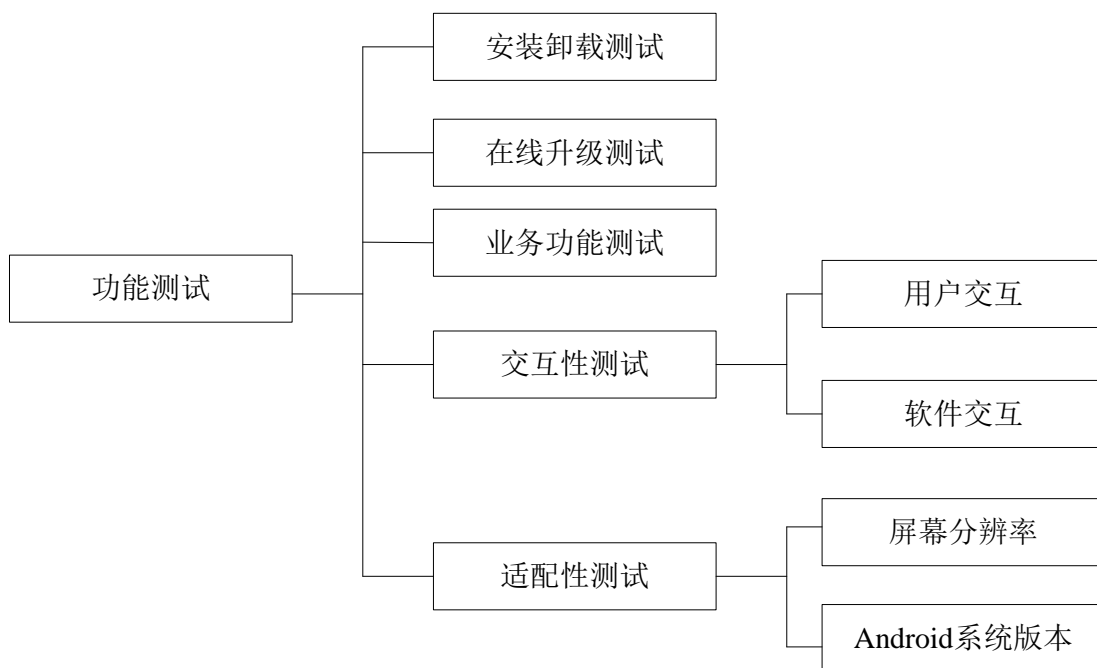


图 4-3 UI 功能测试

(1) 安装卸载测试

安装卸载测试是最重要的测试，如果一个应用安装不成功，则无法打开使用该应用，后面其他的测试都是徒劳的。安装测试方法有：

- 1) 通过命令行“adb install 安装包路径”可以安装相关的应用，如安装 E 盘下的 RUI 手机桌面，输入“adb install E:\uprui_launcher_phone.apk”即可。
- 2) PC 端下载辅助工具，如豌豆荚、应用宝，搜索对应的 UI 应用软件下载安装。
- 3) 可以直接在手机端上安装第三方市场应用软件，如 91 助手、360 手机助手，在市场中查找对应的软件进行安装。

安装成功后，一定要打开 RUI 手机桌面应用软件，观察是否可以正常运行。接下来就是卸载测试了。

同样的，软件卸载可以通过命令行的方式进行卸载。具体命令是“adb uninstall 包名”。也可以在手机设置中的所有应用菜单中，找到被测应用进行卸载。

通过命令行的方式安装 RUI 手机桌面，如图 4-4 所示。

```
C:\Documents and Settings\alisa>adb install E:\papercode\refactor\uprui_launcher_phone.apk
2486 KB/s (12889730 bytes in 5.062s)
    pkg: /data/local/tmp/uprui_launcher_phone.apk
Success
```

图 4-4 安装 RUI 手机桌面命令行

(2) 在线升级测试

指一个软件进行跨版本的安装时，是否安装成功，安装后是否可以正常打开使用。软件包更新时，是否会弹出更新对话框等。例如，V2.2.0 版本的 RUI 手机桌面直接安装 V2.8.0 版本，验证是否可以安装成功，并可以顺利打开。

(3) 业务功能测试

业务功能测试主要是根据需求测试文档，罗列出业务的功能点，然后根据功能点设计测试用例，最后根据设计的用例步骤验证功能点是否实现了。如测试 RUI 手机桌面中的下载推荐应用，验证在网络正常的情况下，可以下载成功。在网络比较弱的情况下，下载推荐应用会出现提示“网络比较慢，请稍后下载”。网络突然中断时，提示“请回设置菜单中，重新选择网络连接”。

(4) 交互性测试

可以将交互性测试分为软件交互与用户交互。软件交互表示手机同时打开多个应用软件，如打开微信、聊 QQ、听音乐等，被测应用是否运行正常。另外，用户交互，是指界面与用户之间的交互，例如，在 RUI 手机桌面中，用户习惯点击中间的按钮，进入分类应用界面。

(5) 适配性测试

适配性测试主要测试应用软件是否兼容不同版本的 Android 操作系统，分辨率，屏幕尺寸等等。RUI 手机桌面主要针对手机设备，而不是平板或电视机。

4.3.2 启动时延测试方法的实现

(1) 通过 DDMS 工具中的 LogCat 视图，分析 RUI 手机桌面从启动到界面完全呈现出来时弹出的日志信息。该日志对应的时间点，即为 RUI 手机桌面的启动耗时。

具体实现步骤：首先，检查手机中是否已安装 RUI 手机桌面，如果未安装则进入应用市场查找，下载并安装。如果手机中已经安装了 RUI 手机桌面，则

需要清除应用数据，具体清除数据的命令行方式是“adb shell pm clear com.uprui.phone.launcher”。其次，打开 DDMS 工具，锁定 Logact 视图。最后，点击 RUI 手机桌面应用，观察 LogCat 视图里弹出各种各样的日志信息。等待完全进入 RUI 手机桌面后，按下鼠标。

当前获取到的日志信息就是 UI 弹出时的日志信息，如图 4-5 所示，Displayed 日志后面的 3s515ms 就是 RUI 手机桌面的启动耗时。

Tag	Text
LauncherAppl...	Receive a action:com.uprui.phone.launcher.action.COLLECT_APPS
ActivityManager	Start proc com.uprui.phone.launcher:bdservice_v1 for service com.uprui.phone.lau
	ncher/com.baidu.android.pushservice.PushService: pid=1360 uid=10002 gids={3003, 1015, 1006, 3002, 3001, 1007}
AllAppsView	shutDownThread
ActivityManager	Displayed com.uprui.phone.launcher/com.rui.phone.launcher.Launcher: -3s515ms
ActivityManager	Displayed com.uprui.phone.launcher/com.rui.phone.launcher.userguide.LauncherGuid
	e: +1s12ms

图 4-5 LogCat 视图下测试 RUI 手机桌面的启动时延值

(2) 不依赖于 DDMS 工具中的 LogCat 视图，通过命令行或脚本的方式实现日志的抓取与分析，来计算 RUI 手机桌面的启动耗时。

从第一种测试方法上，可以看出，RUI 手机桌面从启动后待界面出现时，弹出的日志信息为“Displayed com.uprui.phone.launcher...”，这样就可以直接在捕获到的日志文件中搜索出对应的日志来计算启动时延。具体实现流程：

- 1) 对 RUI 手机桌面清除数据。
- 2) 通过命令行“adb logcat -c”清除缓存日志。
- 3) 接着在命令框中键入“adb logcat -v time -b main -b system -b events >D:\alisa\log.txt”，进入日志捕获的状态中。
- 4) 点击 RUI 手机桌面，观察界面启动情况，等界面出现时，在命令框中输入 Ctrl+C，停止日志的抓取操作。
- 5) 在“D:\alisa\”路径下，打开已保存的 log 文件，并搜索日志“Displayed com.uprui.phone.launcher...”，就可以统计出当时启动桌面时的时间。

将以上的步骤用脚本来实现，按照自动化脚本测试启动时延的方法，选择的基类，功能不变，仍然实现设备的连接，脚本环境初始化等工作。接着复用 Logcat.py 方法。因为 Logcat.py 文件实现的是日志的抓取、分解、保存等操作。

测试 RUI 手机桌面的启动耗时主要需要修改脚本用例，因为测试的 UI 对应的包名不一样，弹出的起始日志与结束日志有区别，具体核心代码如下：

```
def runTest(self):
    from collections import OrderedDict
    config = OrderedDict()
    config['begin'] = ur'Start.*?com.rui.phone.launcher.Launcher'
    config['01.UI 出现时间'] =
ur'Displayed.*?com.rui.phone.launcher.Launcher '
    self.init_monitors(**config)
```

代码中定义了一个 `OrderedDict` 有序键值对的容器类,用于存放启动日志与显示界面的日志,启动耗时为两个日志弹出时间的差值。其中,“.*?”属于正则表达式中的非贪婪模式,表示的意思是与字符串进行匹配,成功一次就结束。

如图 4-6 所示,就是脚本实现的结果。测试的 RUI 手机桌面的启动耗时为 3534ms 与 LogCat 视图中测试出来的 3s515ms 比较一致。

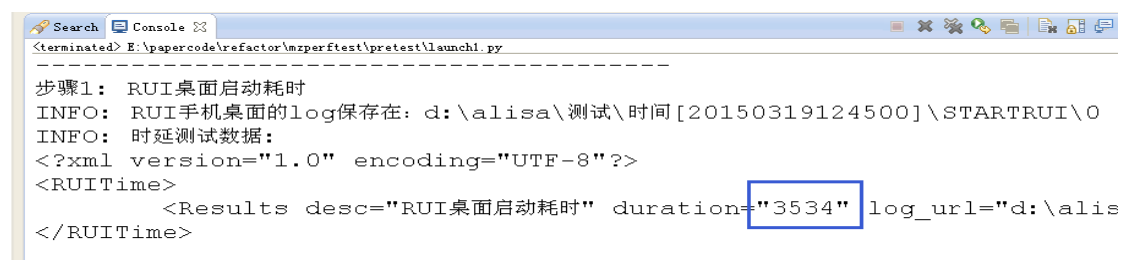


图 4-6 脚本测试 RUI 手机桌面的启动时延值

4.3.3 内存测试方法的实现

(1) 按照 DDMS 工具中的 Heap 视图查看内存的方法,实现对被测 UI,即 RUI 手机桌面展开内存测试。

首先,通过数据线将手机与 PC 端相连接,并在应用宝市场上搜索最新的 RUI 手机桌面 (V3.2.0) 进行下载操作。接着,打开 DDMS 工具,并指定到 Heap 视图下。在 Device 视图下选择 RUI 手机桌面的进程。再次,点击监听内存的“Heap 按钮”。最后,按 Gause GC 按钮并操作 RUI 手机,进行刷新,观察 Allocated 值的变化范围。具有流程,如图 4-7 所示。



图 4-7 内存测试流程图

按照 Heap 视图测试内存的方法，对 RUI 手机桌面进行测试。如图 4-8 所示，可以得出当前占用的内存值为 5.196MB。

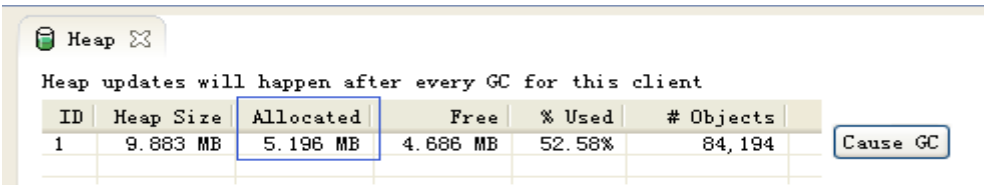


图 4-8 Heap 视图显示内存值

(2) 通过 `dumpsys meminfo` 命令行的方式来实现 RUI 手机桌面的内存值计算。

该方法实现比较简单，只需要知道 RUI 手机桌面的包名后，在命令窗口中输入“adb shell dumpsys meminfo 包名”。其中，查看应用的包名，可以直接问项目的开发人员，也可以通过 DDMS 工具去查看包名。

如图 4-9 所示，测试出了 RUI 手机桌面的内存值。按照该方法测试的内存值为 5382KB = 5.25MB，与 Heap 视图下的内存值比较一致。

```
C:\Documents and Settings\alisa>adb shell dumpsys meminfo com.uprui.phone.launcher
Applications Memory Usage <kB>:
Uptime: 14555037 Realtime: 413097813

** MEMINFO in pid 4331 [com.uprui.phone.launcher] **
      native  dalvik  other  total
size:   21340   8775   N/A   30115
allocated: 18438   5382   N/A   23820
```

图 4-9 dumpsys meminfo 测试 RUI 手机桌面的内存值

4.3.4 流畅度测试方法的实现

使用 GT 工具实现 RUI 手机桌面的流畅度测试。下载安装 GT 工具软件，打开 GT 工具进行初始化设置，并选择被测应用为 RUI 手机桌面，同时，将已关注的参数设置为 FPS。

初始化设置完成后，回退到 RUI 手机桌面中，点击 GT 工具的悬浮开关按钮并滑动 UI 界面进行 FPS 录制，待滑动结束后，停止 FPS 值的采集。最后，进入 GT 工具，打开 FPS 窗口，查看 FPS 值的趋势图，或是点击保存，将运行的结果保存到手机里面。

接着导出 FPS 值的测试数据，可以通过 adb pull 命令行实现数据导出。数据导出后，可以通过 FPS 值的数据进行分析，也可以根据 FPS 值的折线图分析。显示结果。如图 4-10 所示，列出了 FPS 值的折线图。可以看出手动滑动 RUI 手机桌面的 UI，得出的 FPS 值最大为 57fps，平均值为 43fps。



图 4-10 FPS 值的显示图

同样的，可以在 GT 工具之上录制脚本实现自动滑动 RUI 手机桌面，并采集对应的 FPS 值，脚本代码的编写与第三章流畅度测试方法的改进思路中类似，可以直接复用。测试结果的比较将会在第五节展开讨论。

4.4 测试结果的分析

实现了 RUI 手机桌面的启动时延、内存、流畅度的测试方法后，最重要的一步就是分析测试数据。一般有两种分析方法：第一种是与自身产品的其他版本进行数据对比，分析性能数据变化的趋势。

第二种是与竞争产品对比性能数据。在同一个测试场景与环境下，不同产品的性能数据可以直接反映出一些问题。通过对比分析后，可以发现自身产品的缺点与需要优化的方面。比如 RUI 手机桌面的竞争产品有 Android 91 桌面、360 手机桌面、魔秀桌面等。本文主要选取 Android 91 桌面与 RUI 手机桌面做性能数据对比。

4.4.1 启动时延测试结果的分析

利用启动时延的测试脚本，在多台不同型号的硬件设备上，分别对 RUI 手机桌面与 Android 91 手机桌面进行测试。常用的测试设备如表 4-7 所示。

表 4-7 测试设备信息

编号	手机型号	操作系统	分辨率
1	InFocus(M310)	Android 4.2.1	1280*720
2	Huawei(U8860)	Android 2.3.6	854*480
3	Samsung (I9250)	Android 4.0.1	720*1280
4	Samsung (I9100)	Android 2.3.1	800*480
5	HTC Sensation XE	Android 4.0.3	960*540

如表 4-8 所示，显示了在型号为 InFocus(M310)，Android 系统 4.2.1，分辨率为 1280*720 的设备上，分别对 RUI 手机桌面与 Android 91 桌面测试启动耗时。

表 4-8 启动耗时对比表

序号	应用启动耗时 (ms)	
	RUI 手机桌面 (V3.2.0)	Android 91 桌面 (V6.3.1)
1	3165	1944
2	3534	1941
3	3515	2021
4	3700	1922
5	3539	1892
6	3636	2031
7	3585	1933
8	3898	2012
9	3758	2127
10	3824	2087
平均值	3615.4	1991
差值	1624.4	

可以看出，RUI 手机桌面的启动时延比 Android 91 桌面多花费了 1624.4ms。

1624.4ms 严重影响了用户的体验效果，RUI 手机桌面需要在启动耗时这方面做重大优化。

4.4.2 内存测试结果的分析

在多台不同型号的硬件设备上，按照命令行“`dumpsys meminfo`”的内存测试方法，分别测试 RUI 手机桌面与 Android 91 桌面。

如表 4-9 所示，列出了在设备型号为 Huawei U8860，Android 系统 2.3.6，分辨率为 854*480 的手机上，分别测试 RUI 手机桌面与 Android 91 桌面的内存值。

表 4-9 内存值对比表

序号	应用占用的内存值（MB）	
	RUI 手机桌面（V3.2.0）	Android 91 桌面（V6.3.1）
1	5.196	5.095
2	5.503	5.270
3	5.288	5.551
4	5.332	5.318
5	5.337	5.318
6	5.122	5.470
7	5.518	5.106
8	5.402	5.713
9	5.347	5.034
10	5.609	5.501
平均值	5.365	5.337

如果在操作过程中内存值突然猛增，增长的幅度比较大，并且内存值降不下来，就是内存泄漏的问题。从内存值的对比表可以看出，RUI 手机桌面的内存值，在 5M 与 6M 之间波动，不存在泄漏的情况。

从内存值对比表，可以看出 RUI 手机桌面的内存值与 Android 91 桌面的内存值比较接近，但仍然有优化的空间。

4.4.3 流畅度测试结果的分析

在 GT 工具的基础之上，利用自动化脚本测试 RUI 手机桌面与 Android 91

桌面。也就是说在同一个测试环境,使用同一套自动化脚本采集两个应用的 FPS 值。

如图 4-11 所示,显示了在型号为 InFocus(M310),分别滑动 UI 采集 FPS 值。其中,(A)图表示的是 RUI 手机桌面(V3.2.0)采集到的 FPS 值,(B)是 Android 91 手机桌面(V6.3.1)采集的 FPS 值。



图 4-11 采集 FPS 值的趋势图

从 (A) 与 (B) 图可以看出,两个桌面的 FPS 值都达了 30fps 以上,说明两个桌面的流畅度是在用户可以接受的范围之类。将两个结果图对比来看,RUI 手机桌面的 FPS 值波动比较大,中间有四个小于 50fps 的数值,表示滑动过程中出现了丢帧,即 Jank,需要不断完善。

4.5 改进方法的结果对比

4.5.1 启动时延测试方法的对比

(1) 测试结果的精度对比

脚本自动化测试 RUI 手机桌面启动耗时是根据抓取的 logcat 信息来计算时间的，日志什么时候打印出来，启动时间就是多少，不依赖人的主观意识操控。而常用的秒表测试方法，主要是在启动 RUI 手机桌面前按一下秒表，待 RUI 手机桌面全部呈现出来时，再按一下。这个过程主观因素比较强，一不留神可能就错过了按秒表的时间。

如图 4-12 所示，列出了自动化与秒表法测试五十次 RUI 手机桌面的启动耗时，所得到的结果对比。

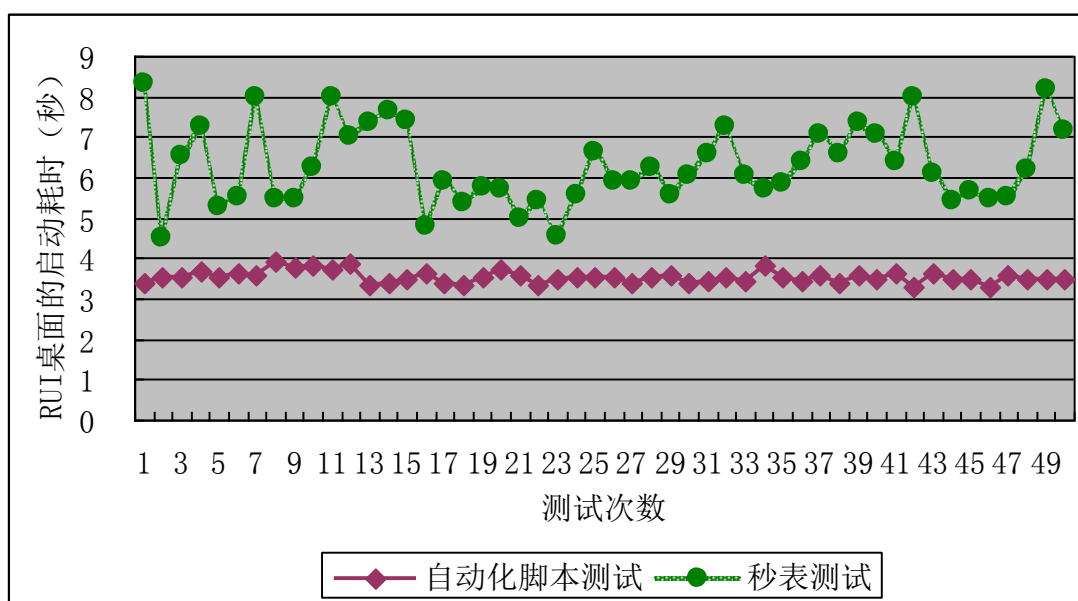


图 4-12 RUI 手机桌面启动时间的结果对比

从启动时间测试结果对比图可以看出，采用自动化脚本测试 RUI 手机桌面的启动耗时，精度比较高，取值在 3.5 秒与 4 秒之间，根据这五十组值计算的方差值为 0.145，可以看出测试的 RUI 手机桌面的启动耗时的值波动很小。

而利用手工，秒表的测试手段，得出来的时间跨度在 4 秒到 9 秒之间，远远大于自动化的测试结果。根据五十组值计算的方差值为 0.974，可以看出值的波动远远大于自动化测试的结果。例如，在测试人员比较疲惫时，利用秒表测试出来的时间可能达到九秒，正常情况测试的时间是五秒，所有误差比较大。

(2) 测试效率对比

与 LogCat 视图测试时延的方法进行比较，自动化测试大大提高了测试人员的工作效率。因为在 LogCat 视图中测试 RUI 手机桌面的启动耗时是一种手工

操作方式，每测一次，需手动清除 RUI 手机桌面的数据、统计结果等。

如图 4-13 所示，列出了测试 RUI 手机桌面五十次启动时延，花费测试员的时间对比图。其中，每次启动都是初始化的启动，即清除数据，不带缓存的。

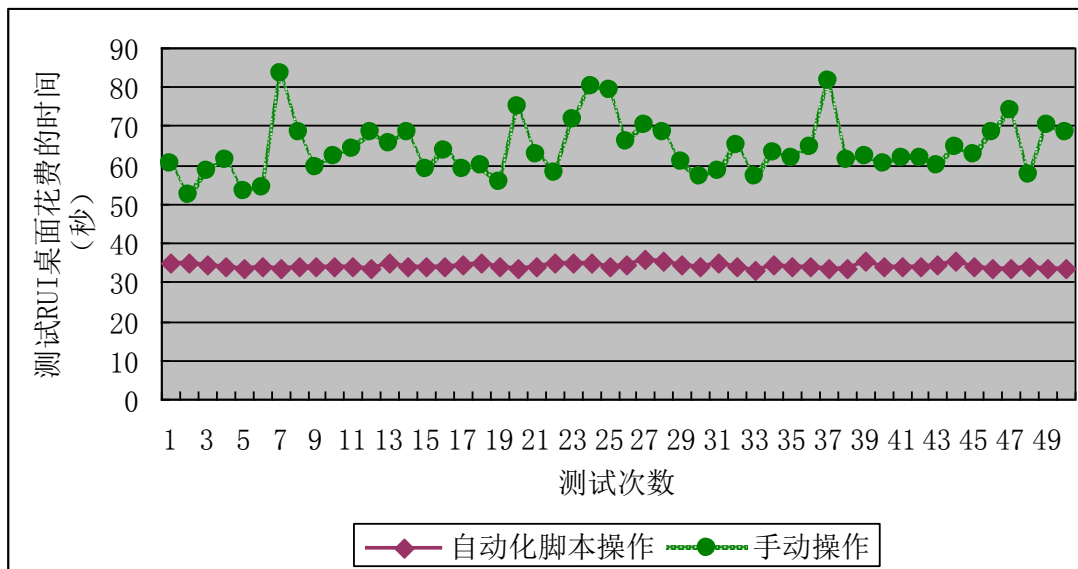


图 4-13 花费测试时间的对比图

从测试 RUI 手机桌面的启动耗时所花费时间对比图可以看出，自动化脚本测试 RUI 手机桌面比手动操作节省了一半的时间，而且每次操作消耗的时间都比较一致。完成一次启动时延的测试操作基本需要花费 32 秒左右。

而手动测试花费的时间忽大忽小，没有规律，测试效率依赖测试人员的操作速度与反应力。在测试人员比较熟练的情况下，花费的时间比较一致，在 65 秒左右。但是，随着测试次数增多，人也容易疲劳，测试花费时间又开始变化。

4.5.2 流畅度测试方法的对比

改进后的流畅度测试方法与没有改进之前的方法进行对比，主要体现在 FPS 值的变化。手动滑动 UI，随意性比较大，不能在同一个频度下采集 FPS 值，所以最后得到的 FPS 值悬殊比较大。

而在自动化脚本中，设定了滑动的频率，滑动的距离，滑动的次数等参数，因此采集到的 FPS 值不会出现极端情况。譬如，手动滑动 UI 界面时，会出现滑动疲劳，突然停顿的情况。

如图 4-14 所示，列出了自动化滑动与手动滑动 RUI 手机桌面，分别采集五十次 FPS 值。

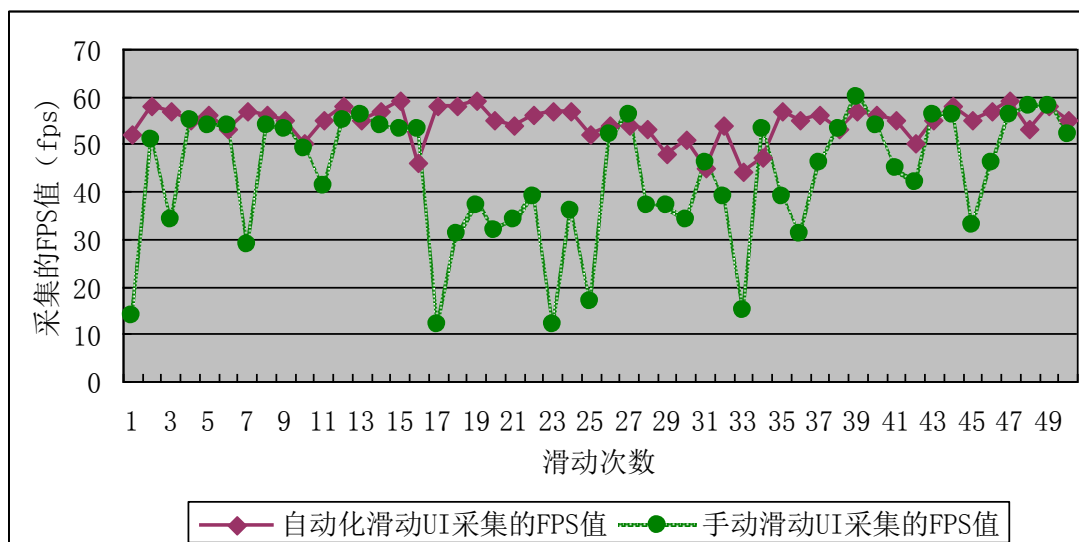


图 4-14 采集 FPS 值的结果对比图

从图可以看出，脚本滑动的过程中，滑动的速度，频率都一样的，因此采集到的 FPS 值都是在 50fps 与 60fps 值之间，其中有 7 个小于 50fps 的值，表示 RUI 手机桌面界面滑动的过程中出现的丢帧，即 Jank。

而手动滑动 UI 时，FPS 值没有规律，在测试人员滑动得比较快的时候达到了 55fps 值，滑动的过程中稍有停顿，可能降到 12fps。所以不能准确判断哪些是正常采集的 FPS 值，哪些是丢帧后的 FPS 值。

4.6 本章小结

本章主要内容是将第三章提出的启动耗时，流畅度，内存相关的三大类测试方案应用到一个具体的项目中。首先，介绍了被测项目的背景知识，接着阐述了在测试方法执行前的一些准备工作。再次，对具体测试方法进行实现。并选取 RUI 手机桌面的竞争产品 Android 91 桌面作为测试结果的对比对象，将 UI 启动时延、内存、流畅度测试的数据做比较，得出自身不足与需要优化的地方。最后，对改进后的启动时延测试方法与流畅度测试方法取得的有效成果进行验证，主要包括测试时间效率与测试结果数据精度的提高。

第 5 章 总结与展望

5.1 总结

同类 Android 应用软件中,在所具备功能接近的情况下,Android 用户对一个应用的体验效果主要是由 Android 系统 UI 的性能好坏来决定的,如:Android 系统 UI 的启动速度的快慢、界面滑动的流畅度、占用内存的大小等等。这些性能指标的好坏会影响到一个应用在市场上的排行榜地位。

对 Android 系统 UI 的相关性能测试是保证这些指标符合用户要求的重要手段。本文取得的主要成果包括:

(1) 详细分析了影响 UI 性能的三类因素,分别是 UI 的启动时间,内存泄露,UI 的流畅度。并对这三类影响因素,提出了三种高效的测试手段,包括可以利用 LogCat 视图测试 UI 启动的时间,通过 GT 工具测试 UI 的流畅度,用命令行“`dumpsys meminfo`”与 Heap 视图对内存进行测试。

(2) 对性能测试中的 UI 启动耗时与流畅度测试提出了改进方案,并对改进的方案进行分析,分析了改进后的优点。其中,UI 启动耗时主要提倡的是利用编写脚本自动化代码去抓取 logcat 日志,然后对日志进行分解,进而正则匹配计算出 UI 从启动开始一直到界面显示出来时消耗的时间。流畅度测试方法的改进是将手工打开 GT 工具,滑动 UI 采集 FPS 值改为编写脚本程序去实现自动滑动 UI,自动操作 GT 工具进行采集。

(3) 将改进的测试方法应用到一个具体的实际项目中,完成性能测试工作。首先展开测试的前期工作,包括测试需求分析、需求评审、测试环境设计、测试用例设计等。接着执行 UI 测试工作,包括功能、内存、启动时延、流畅度等测试方法的实现。最后,利用竞品测试方法,在手机应用市场上取与被测项目同一类型的一款产品进行测试,并对测试出来的结果进行对比,提出自身应用需要优化的地方。

(4) 用测试的数据来证明 UI 启动时延测试方法与流畅度测试方法的改进取得的成效。改进后的方法主要节约了测试人员测试的时间,提高了测试结果的精确度,避免了一些异常数据的出现。

5.2 展望

本文主要针对一款 Android 系统 UI 软件的测试进行了研究，着重对 UI 性能测试中的内存、启动耗时、流畅度等方面的测试进行深入的研究与探索，并提出了一些改进的方案。如，在时延测试方法中，为了提高测试人员的测试效率，编写脚本代码实现启动耗时的测试。

但是由于个人的能力与研究的时间有限，对 UI 测试方法的研究工作未能进一步深入，还存在需要优化与改进的地方。主要表现在以下几个方面：

（1）虽然对影响 UI 的性能因素进行了分析，但是分析的还不够全面，未能从 UI 的耗电量，流量等指标进行考虑。因此，对 UI 耗电量与流量两个指标的探讨，将会是以后工作中的一项任务。

（2）主要对性能测试中的 UI 启动时延与流畅度的测试方法提出了改进的思路与方案，可以进一步考虑对 UI 的内存测试方法进行改进，实现自动化的采集内存值并分析值的变化。

（3）流畅度测试方法的改进主要是建立在已存在的 GT 工具上进行自动滑动 UI 采集 FPS 值的，需要进一步对 GT 工具的实现原理做更透彻的理解，直接将原理应用到脚本中。

致谢

光阴似箭，三年的研究生生涯如指间的流水一晃而过，在面临即将毕业走向社会的日子，回首过往，感慨颇多。在此，特向三年来，所遇到的恩师益友们表示最诚恳的谢意。

首先，感谢我的导师张能立，他勇于追求真理，不断创新的精神，成为了我学习的榜样。同时，要深深的感谢 UP 实验室的主指导老师李宁，这三年来，李老师认真负责。在专业技术方面，带领着我们一步一步深入钻研并教导我们做技术不能只停留在表面，要不断深挖。在毕业论文的选题与写作方面，他耐心指导，认真批注，反复帮助我们修改，直到我们的论文全部通过。

其次，要感谢小火软件有限公司和腾讯科技（深圳）有限公司提供给我的实习机会。在小火软件有限公司实习使我对 Android 应用测试有了一个系统性的认识，并能开展 RUI 手机桌面的常规功能测试工作。在腾讯科技公司半年多的实习经历，让我的测试技术得到了升华，测试不仅仅停留在发现软件中的功能缺陷，还要求对软件的性能进行严格把关，并通过编写脚本提高测试效率。

再次，要感谢王博和孙岳龙师兄，感谢他们带领着我学习编程，并指导我开发一些工具，为找一份好工作做准备。同时，要感谢我们这一届 UP 团队的 7 个小伙伴，我们一起奋力拼搏，一起欢声笑语，有你们的存在，给我的研究生生活增添了无尽的色彩。还要感谢三年的室友邓颖、任肖弦、杨茜，感谢你们在学习和生活上对我的关心与帮助。

最后，要感谢我的父母和哥哥，感谢他们对我的关心与照顾，让我在学校无后顾之忧，能够安心学习。

苏敏

2015 年 4 月

参考文献

- [1] 2014年最新全球智能手机厂商销量排行榜[EB/OL]. <http://www.askci.com/chanye/2014/12/19/16112aw2.shtml>.
- [2] 王建华. Android开发实用教程[M]. 中国水利水电出版社, 2014.
- [3] Android takes almost 50% share of worldwide smart phone market[EB/OL]. <http://www.docin.com/p-432735515.html>.
- [4] 王丽. 移动应用软件测试探索[J]. 计算机系统应用, 2013, 22(01): 1-4.
- [5] 王华斌. 基于Android平台的智能终端安全研究[D]. 北京交通大学, 2014.
- [6] K.Mustafa & R.A.Khan. Software Testing Concepts and Practics[M]. Science Press, 2009.
- [7] 2014非常好用的开源Android测试工具[EB/OL]. <http://www.oschina.net/news/56142/open-source-android-testing-tools>.
- [8] Facebook网络模拟测试工具ATC使用[EB/OL]. <http://www.cnblogs.com/coderzh/p/AugmentedTrafficControl.html>.
- [9] 李刚. 疯狂Android讲义[M]. 电子工业出版社, 2013.
- [10] 王博. Android系统UI定制关键技术研究[D]. 武汉理工大学, 2013.
- [11] 余志龙. Google Android SDK开发范例大全[M]. 人民邮电出版社, 2009.
- [12] Reto Meier. Professional Android for Application Development[M]. Wiley Publishing, 2012.
- [13] 佐冰冰. Android平台下Launcher启动器的设计与实现[D]. 哈尔滨工业大学, 2012.
- [14] 罗雷, 韩建文. Android系统应用开发实战详解[M]. 人民邮电出版社, 2014.
- [15] 杨丰盛. Android应用开发揭秘[M]. 北京: 机械工业出版社, 2010.
- [16] Yu Z L. Google Android SDK Application[M]. Beijing: Posts & Telecom Press, 2012.
- [17] 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: 110-112.
- [18] 赵鲲. 基于Android平台的图书管理系统手机客户端开发[D]. 电子科技大学, 2013.
- [19] Lauren Darcey, ShaneConder. Android Wireless Application Development[M]. Addison-Wesley Professional, 2009.
- [20] Jin T Y. Research of Android Architecture[M]. Beijing: Posts & Telecom Press, 2012.

- [21] Fangchun Jiang, Yunfan Lu. Software testing model selection research based on Yin-Yang testing theory[C]. International Conference on Computer Science and Information Processing (CSIP), 2012: 590-594.
- [22] Muccini H, Di Francesco A, Esposito P. Software testing of mobile applications: Challenges and future research directions[J]. 7th International Workshop on Automation of Software Test (AST), 2012: 29-35.
- [23] Shane Conder, Lauren Darcey. Android移动应用开发从入门到精通[M]. 北京: 人民邮电出版社, 2010.
- [24] Mark L. Murphy. The Busy Coder's Guide to Android Development[M]. CommonsWare LLC, 2008.
- [25] 胡霞. Android版某记事本旅游记忆模块的设计与实现[D]. 北京交通大学, 2013.
- [26] Android系统UIautomator实例使用[EB/OL].
<http://blog.csdn.net/huiguixian/article/details/22398193>.
- [27] UIAutomatorViewer [EB/OL]. <http://blog.csdn.net/itfootball/article/details/41944393>.
- [28] 万年红, 李翔. 软件黑盒测试的方法与实践[J]. 计算机工程, 2000, 26(12): 91-93.
- [29] 陈晔. Android与iOS应用测试指南[M]. 清华大学出版社, 2014.
- [30] Ron Patton. Software testing[M]. Beijing: China Machine Press, 2006.
- [31] 宋春雨. Android平台自动化测试的研究与实践[D]. 北京邮电大学, 2012.
- [32] Kanglin Li, Mengqi Wu. Effective GUI Test Automation[M]. Beijing: Publishing House of Electronics Industry, 2005.
- [33] 聂长海. 软件测试的概念与方法[M]. 北京: 清华大学出版社, 2013.
- [34] 张岩. 自动生成测试用例的方法设计及实现[J]. 计算机系统应用, 2011, 20(09): 238-240.
- [35] 薛冲冲, 陈坚. 软件测试研究[J]. 计算机系统应用, 2010, 20(02): 240-244.
- [36] 陈昊. Android手机自动化测试系统的设计与实现[D]. 西安电子科技大学, 2014.
- [37] 陈卫俊. 互联网单元测试及实践[M]. 电子工业出版社, 2008.
- [38] Randal E. Bryant, David R.O Hallaren. Computer Systems: A Programmer's Perspective Software testing[M]. Beijing: China Machine Press, 2007.
- [39] Liu B L. Memory Leak in Android[EB/OL]. <http://www.apkbus.com/android-71944-1-1.html>.
- [40] Android SurfaceFlinger服务代理对象获取过程源码分析[EB/OL]. <http://www.tuicool.com/articles/NNRZni>.
- [41] Li N. Director of Android Developer[M]. Beijing: Posts & Telecom Press, 2012.

- [42] 刘宇光. 基于众核处理器的高清内窥镜图像处理软件研发[D]. 浙江大学, 2014.
- [43] 移动操作系统架构的趋势[EB/OL]. <http://www.docin.com/p-722966732.html>.
- [44] Andrew Watt. Beginning Regular Expressions[M]. Beijing: Tsinghua University Press, 2008.
- [45] 周颖. Android平板稳定性自动化测试的研究和实现[D]. 华中师范大学, 2014.

攻读硕士期间的研究成果

1、计算机软件著作权

软件名称：《安卓智能终端设备远程控制软件》

证书编号：2015SR064066

出版单位：中华人民共和国国家版权局

时 间：2015 年 4 月 16 日