计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(8):126−131 [doi: 10.15888/j.cnki.csa.006492]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 GUI 的 Android 移动软件性能测试①

谷林涛, 徐立华

(华东师范大学 计算机科学与软件工程学院, 上海 200062)

摘　要: 移动应用软件已经拥有了数以千万计的用户群体. 根据最新统计, Android 手机以 85.1% 的市场占有份额, 成为了最受欢迎的移动端设备. Android 应用软件的快速开发, 使得如何保证程序质量, 成了难题. 我们不仅要考虑程序的正确性, 也应保证运行时的流畅性. 现有的性能研究工作都基于传统的静态分析或者动态执行. 对于 Android 程序, 静态分析具有一定的局限性, 而动态分析又忽略了 APP 执行时的遍历方式. 因此, 本文提出了基于 GUI 的 Android 自动化性能测试框架, 将着重关注页面状态和 APP 状态的相关性. 通过对页面的分析, 聚合, 尽可能遍历到 APP 的各个状态, 然后从日志中, 找出 APP 性能上的问题. 本框架使用 Java 作为开发语言, 搭建了 Android 移动软件自动化性能测试框架, 并在开源社区 F-Droid 上随机抽取了移动应用软件进行实验. 实验表明, 该技术能更多的遍历 APP 的状态, 发现 APP 在运行过程中出现的性能问题, 取得良好的效果.

关键词: 性能测试; Android 应用程序; GUI 测试; 移动应用; 自动化

引用格式: 谷林涛,徐立华.基于 GUI 的 Android 移动软件性能测试.计算机系统应用,2018,27(8):126–131. http://www.c-s-a.org.cn/1003-3254/6492.html

Performance Testing on Android Application Based on GUI

GU Lin-Tao, XU Li-Hua

(School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

Abstract: Mobile applications have become increasingly popular over the recent years. As shown in the latest statistics, over 85.1% of mobile phones is based on Android operation system. Due to the Android's open-source character, Android application is becoming increasingly popular. Nevertheless, how to ensure program quality has become a severe problem. Previous work focused on doing static or dynamic analysis on the code to find performance problem. Yet they did not focus on the combination of performance testing and oriental testing. Therefore, in this paper we presents a GUI-based software automated performance testing framework. By parsing pages, we try to find the APP states as much as possible, and then analyze the log files to find out the APP performance problems and recover the problem scenarios. To ensure the operability of our tool, we use Java to set up this automated testing framework for Android mobile software. We download mobile applications from the open source community named F-Droid for experiments. The result shows that our technology can find more state of the APP and reveal performance problems in the running process.

Key words: performance testing; Android application; GUI testing; mobile application; automation

随着智能手机的日益普及, 以及移动设备硬件性能的不断提升, 移动应用软件 (以下简称 APP) 越来越多的走进人们的生活. 根据 IDC2017 统计, 85.1% 的手机使用了 Android 平台. 根据国外 Android 开源平台 F-Droid 统计, 每个月在 Google Play 上发布的新型 APP 平均数量多达五万. 受益于 Android 的开源性, 开发一

① 基金项目: 国家自然科学基金 (61502170)
Foundation item: National Natural Science Foundation of China (61502170)
收稿时间: 2017-12-27; 修改时间: 2018-01-16; 采用时间: 2018-01-25; csa 在线出版时间: 2018-07-28

个 APP 的成本相对较低, 研发时间相对较短. 但由于开发者水平不同, 以及 Android 系统定制化现象严重, APP 程序质量一直成为大家关注的焦点. 因此如何保证 APP 上线后的程序质量, 成为了开发者们关注的焦点, 也是学术界关注的一项热点.

现有工作的焦点是对 APP 运行时的正确性展开研究, 避免运行时 APP 崩溃, 给用户带来不良体验. 但 APP 在运行时频繁出现页面渲染时间长、操作反馈慢, 甚至出现未响应的情况, 也会给用户带来的不良的使用体验. 因此, 在 APP 上线之前对其进行性能测试, 确保运行时流畅也是必不可少的.

近几年来, 学术界对 Android 性能的分析已经逐渐开展, 并和自动化测试一起进行研究探索.

Linares-vasquez[1]等通过对开源社区 APP 中 issue 事件的分析, 详细归纳了导致 APP 性能出现异常, 如线程问题, GUI 问题, 内存问题的原因.

Hecht 等[2]提出通过静态分析, 针对 Get/Set、静态方法、以及 HashMap 的使用分析了对性能的影响, 并通过实验验证了优化上述代码可以提高 Android 运行的性能, 减少页面的渲染时间.

Liu 等[3]提出了通过静态扫描的方式, 将 APP 的 bytecode 作为入参, 通过分析主进程中的代码和 getView() 的回调函数的代码, 来检测 Android 程序中是否存在性能问题.

Hussein 等[4]详细分析了 Java 的 GC 机制对 APP 运行时的影响, 并做了量化分析, 从垃圾回收这个角度分析了 APP 的性能问题.

除了通过静态分析, 还有动态执行的方式. 通过动态执行, 监控运行时是否频繁触发 GC. 或是监控 View 层是否多次绘制, 这些手段都可以有效地发现性能问题. 但在测试过程当中, 都没有提出如何进行自动化的测试, 忽略了性能测试也需要像功能测试一样, 实现自动化, 方便开发者使用.

相比传统软件, Android 程序基于事件驱动的交互方式, 使得整个程序不以线性的方式运行. 在 Reaves 等[5]工作中描述了 Android 的生命周期, 使用组件的通讯机制, 频繁的系统和页面的回调机制, 这些特征给 Android 的自动化带来极大的挑战. 在 Choudhary 等[6]工作中, 比较了动态执行的工具的效果, 其中 A^3E[7], Dynodroid[8], Monkey[9], GUIRipper[10]等都尝试通过动态执行来对 Android 程序进行正确性测试. 最终的结论是, 从传统的代码覆盖率效果来评估, Monkey 工具的代码覆盖率最高. 但 Monkey 无意义的点击, 以及无法分析页面信息, 使得这个方法无法满足性能测试中需要对页面分析的条件. 其他的工具使用在性能测试上, 发现其无法较多发现 APP 的页面状态, 因此不能更多的发现性能问题. 因此, 我们需要针对性能测试, 设计特殊的遍历模式, 从而可以有效的发现性能问题.

Su 等[11]通过对 APP 建立概率模型, 对 APP 进行随机测试, 获得了良好的效果. 他们通过随机的方式侧重在程序崩溃的测试上, 使用代码覆盖率评估效果. 而性能问题, 则是希望遍历更多状态不同的页面. 因此, 在自动化的遍历过程中, 能发现更多的页面状态, 才能更有效地发现性能问题.

针对上述问题以及挑战, 本文提出基于 GUI 的 Android 自动化性能测试技术, 并使用 Java 作为开发语言, 实现了该自动遍历框架. 本框架通过使用 UiAutomator 对 Android 页面在测试过程中实施进行抓取, 分析, 通过安卓调试桥 (Android Debug Bridge, ADB) 向终端机发送事件, 通过对页面渲染时间的监控, 捕获页面延迟的情况. 本文提出的自动化性能测试方法具有以下优势.

(1) 该工具通过对页面的分析、聚类, 可以获得 APP 更多的运行页面状态, 通过仅发送有效操作事件, 以缩短测试时间.

(2) 在遍历过程中, 该框架会实时对页面进行截图, 将操作行为记入 log 日志, 通过监控模块获取页面渲染时间, 根据截图和操作信息, 恢复问题现场, 方便开发者方便地定位延迟的原因.

(3) 本工具可以配置操作信息, 在测试过程中, 指定在对应的输入框输入相应的信息, 以解决在 Monkey 测试过程中, 遇到文本框无法输入正确的信息, 导致遍历无法顺利进行的问题.

(4) 本工具可以同时在多台机器上测试 APP, 以提供给开发者进行不同机型上同时进行性能测试的功能.

1 相关技术介绍

1.1 UiAutomator 介绍

UiAutomator[12]是 Android 4.0 之后自带的基于 GUI 的自动化测试工具. UiAutomator 可以将手机页面的布局信息抽象为树形结构, 以 xml 文件的形式存储. 通过对 xml 文件的解析, 可以获取控件的具体信息. 比如控件类型 (TextView, Button 等), 控件可以被操作的行为 (Clickable, Scrollable 等), 控件的布局位置. 也可

获取控件之间的层级关系等信息. 本框架的实现使用该技术, 在测试过程中, 实施抓取、分析测试过程中的页面信息, 并每次生成一次测试操作.

1.2 安卓调试桥 (ADB) 介绍

ADB[13]是一个非常强大的 PC 端命令行工具. 通过使用 ADB, 我们可以从 PC 端轻松地发送指令至手机端. ADB 可以实现对手机的指定位置实现点击事件、滑动事件、截图事件、传输文件、发送广播、以及对手机端文件的操作.

通过使用 ADB forword, 还可以实现手机端和 PC 端通过 Socket 端口进行通信, 本框架的底层使用了 ADB 向手机端发送 shell 指令实现对手机端的点击滑动、文件传输等操作, 通过 ADB forword, 通过 Socket 端口和手机端进行数据交互.

1.3 Xposed 介绍

Xposed[14]是一款在可以不修改 Android 应用程序的前提下, 改变 APP 相关行为的框架. 可以实现对一些指定方法执行前后插入执行代码, 在本框架中, 我们对 Loop, 以及 Android 系统的 VSync 进行监控, 获得实时帧数和 Loop 执行时长的信息.

2 框架的设计与实现

Android 基于事件驱动的运行模式, 使得传统的自动化遍历方法很难在该模式下取得良好的效果. Android 程序运行时具有大量多变的人机交互的 GUI, 运行环境又非常复杂. 因此在现有相关工作中, 对页面的建模、以及测试之前的静态分析在 Android 的测试过程中并未取得良好的测试效果. 综合考虑了 Android 的 GUI 多以及环境复杂的特点, 以及随机测试在 Android 测试中的优越性, 本文提出了基于 GUI 的自动化性能测试, 对 APP 的页面进行分析, 聚合状态, 并根据状态不断地更新每个控件在随机测试下被触发的概率值. 以促使在自动化遍历过程当中, 能寻找到更多的界面状态, 遍历更多的程序状态, 以发现性能上的问题.

该框架总共分为两个模块, 其一是执行模块, 其二是日志分析模块.

在执行引擎中, 主要部分是 ADB 模块, 该框架是通过 ADB 的方式连接手机和计算机, 并发送点击、滑动等指令, 也负责传输 UiAutomator 产生的 xml 文件. 而通信模块负责与计算机实现一部分数据交互.

APP 状态管理模块负责维护遍历 APP 的各个页面状态. 一个 APP 根据 Activity 名称划分为多个 Activity 状态, 每个 Activity 中又存在多个 UI 的状态, 相似的状态会被合并, 而不同的状态会被新建. 每个状态都是由 UiAutomator 提供的 xml 文件抽象而成, 存储了所有的控件信息. 而行为生成器就会根据对应的 UI 状态, 对 APP 发送对应的操作.

日志记录是一个独立的模块, 负责记录遍历 APP 时候的操作行为, 通过 Xposed 所产生的帧数监控, 以及 APP 执行时候的状态监控.

系统架构图如图 1 所示.

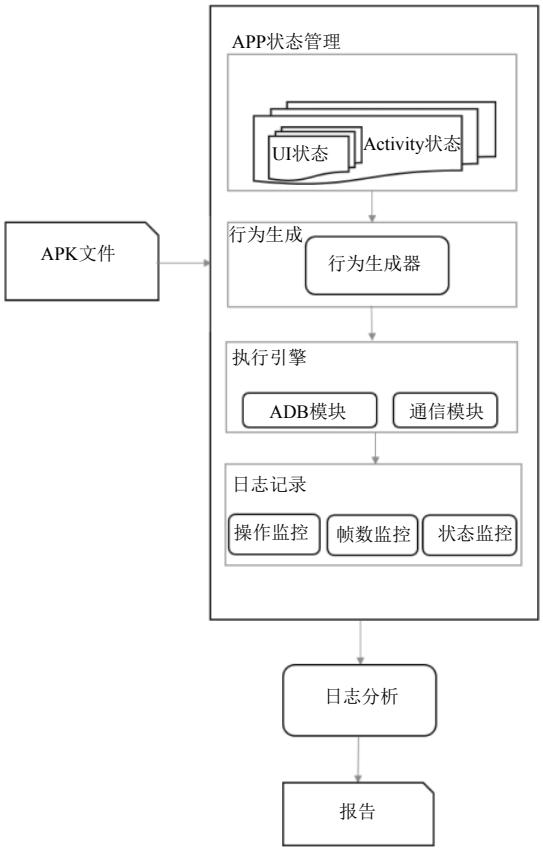

图 1 系统架构图

APP 状态模块的具体行为是在获取一个 Android 手机运行页面后, 首先判断该页面所属的 Activity 是否被遍历过. 若没有被遍历过, 则会生成一个新的页面状态的实例, 加入到 Activity 的页面状态中. 若被遍历过, 则会对比是否与之前的页面相似. 若相似, 则合并两个页面的状态. 若不相似, 则新建页面状态的实例, 并继续加入到该 Activity 的页面状态中去, 是该框架的核心模块, UI 判断的相似算法在下文中阐述.

每个页面实例会记录被执行过的历史记录, 行为

生成器会根据被测的历史记录, 对每个控件生成不同的触发概率值. 最终会根据这些不同的概率触发值, 随机选择一个控件触发. 其详细的工作原理在下文中阐述.

本框架的程序入口会接收一个 APK 文件或者是已经安装好的 APP 的 package 名称. 框架在接受到包名以后会首先加载应用, 然后首先获取初始的页面, 然后框架会获取页面的 XML 文件, 通过 APP 状态管理, 对页面进行解析, 抽象为页面状态的类. 这个类包含了页面的信息, 状态以及页面所属 Activity 名称等. 行为生成器在接收到这些信息后, 会生成一次操作行为, 然后再次会抓取 XML 文件, 提供给状态管理器进行分析. 并由 UI 相似度来区分是否是新的页面状态. 该测试行为会持续直到超过设定的测试时长或者是测试的行为次数才会终止.

在运行过程中, 会有另外的线程负责进行数据的采集, 帧数的分析, 以及拦截运行时 APP 跳出的权限管理, 以及阻止一部分会使得 APP 运行过程中跳出的行为.

最终, 日志分析模块会对 APP 的操作行为和运行时帧数的采集两个日志进行分析, 输出最终的报告作为结果.

本框架的整体遍历逻辑流程图如图 2 所示.

2.1 UI 相似判断

该模块负责对比一个 Activity 下两个页面是否为相似页面. 在相同的 Activity 中, 可能存在多个不同状态的页面, 以实现不同的功能. 例如显示了用户新增的信息, 或者是切换到了不同的选项卡. 因此, 单独依赖 Activity 来区分 APP 的运行状态是不准确的, 一个 Activity 页面可能有多个状态, 而判断两个状态是否相同并不能简单地通过比对 UiAutomator 提供的 xml 文件, 而是要对 xml 文件做具体分析. 本文提出的自动化性能测试框架中的相似判断, 首先分析布局控件. 若布局发生改变, 则被判定为页面不相似, 即两个状态不同. 布局的改变通常说明开发者在同一个 Activity 中, 设有多个 Fragment 组件, 以实现页面不同的功能, 亦或是新增了大量控件, 实现了新的业务需求. 当布局没有发现变化时, 控件数量成为了判定是否相似的第二个条件, 本框架中通过计算相同的控件数和所有的控件数的比值, 从控件相同数量上进行区分, 结合控件的位置信息, 可以判定, 两个页面是否相似. UI 相似度的分析在该系统中有着重要的作用. 其负责合并相同的页面状态, 降低遍历的时间, 同时促使框架能遍历到更多的 APP 状态.

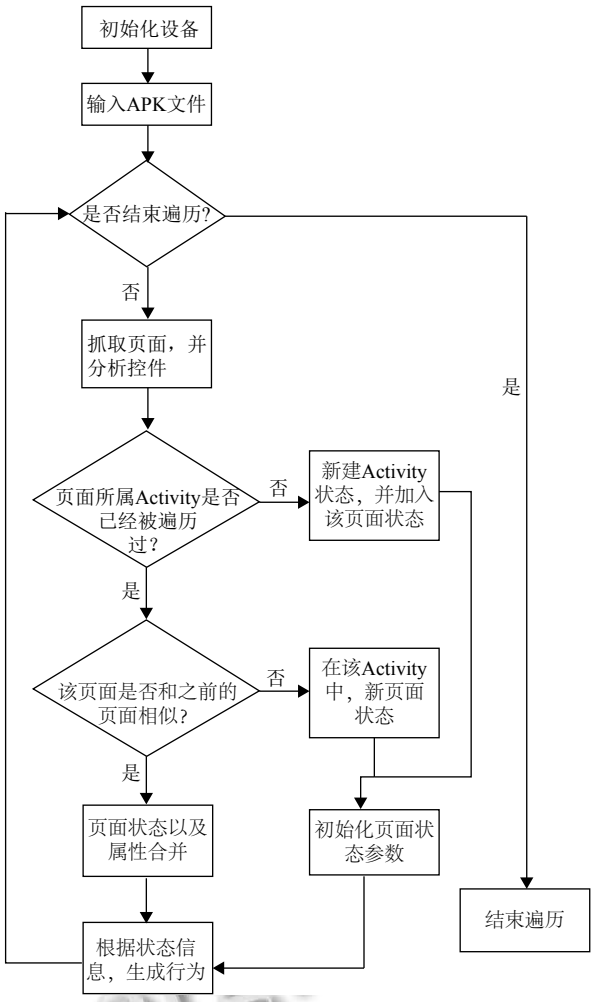

图 2 遍历流程图

当一个页面文件被作为输入后, 该模块首先会对其进行初始化. 根据控件类型, 划分为滑动组件, 点击组件, 输入框组件, 以及不必要关注的组件. 其次会分析布局的组件是否有相互覆盖, 相互重叠的部分, 通过这个条件, 我们经过分析, 可以获取页面可执行的组件, 并封装在对应的数据结构中. 在页面相似的分析中, 我们仅考虑经过过滤之后的控件信息, 这些行为的目的是尽可能从结构上还原用户的操作页面.

2.2 遍历策略

考虑到 Android 基于事件驱动的运行模式, 本文采取优化的随机遍历方法对 Android 的 APP 进行遍历. Monkey 并没有考虑到每个 Activity 的复杂度、深度以及历史执行情况. 因此, 本文提出了一种新型的随机方法, 以解决上述难题和挑战.

我们使用一个有向图来表示一个 APP 在运行过程当中 Activity 的跳转情况. 针对一个 Activity 中的每一个 UI 状态, 每个控件都抽象为四个状态, 分别为操作次数, 操作后的状态, 操作类型以及被触发的概率值. 概率值在页面第一次生成时, 同一个页面所有控件被触发的概率值都是相同的. 在随机遍历过程中, 再次回到该页面时, 每个控件被触发的概率值会根据历史测试行为和 Activity 的跳转图来更新为一个新的值. 其目的是为了希望工具能找到更多的 APP 状态, 遍历业务逻辑复杂的模块. 其具体算法如下.

算法 1. 控件概率生成算法

1) 第一次生成页面, 统计控件数, 并对每个可操作的控件初始化状态. 操作次数为 0, 操作后的状态为 null, 以及被触发的概率为 1/可触发控件总数.

2) 当第二次返回页面时, 判定是否每个控件都被触发过, 若每个控件都触发过, 则跳转到 3), 否则跳转到 4).

3) 若每个控件都被触发过时, 根据 Activity 跳转的有向图, 分析每个控件操作后的状态, 并对每个组件的后驱状态的页面控件数进行排序, 将后驱状态更复杂的控件赋予更大的概率值.

4) 若还存在控件没有被触发, 则更新概率值, 提高没有被触发过的控件被触发的概率值.

在遍历过程当中, 测试程序会根据每个控件状态, 随机产生行为, 进而尝试去发现新的页面状态, 并把操作行为发送到手机端.

2.3 系统事件的触发

Android 手机的行为操作不仅限于用户点击、滑动等操作, 在该测试过程中, 测试工具也会以恒定概率向手机发送一些系统事件, 如, 横竖屏, 音量改变. 在 AVD 模式下, 提供 AndroidManifest.xml 支持设置网络变化, 电量变化等虚拟操作. 以测试在系统事件被触发时, 被测 App 的行为是否会异常.

2.4 监视器的设计

在整个测试过程中, 监视器主要负责如下工作.

(1) 对每一步实现截图, 并在截图上做出标记, 记录每一步操作过程.

(2) 实时监控被测 App 是否处于 Activity 的栈顶. 因测试过程一部分行为会导致被测 App 退出, 则重启 APP, 并阻止测试工具继续进行相同的行为.

(3) 由于 Android5.0 之后授权方式的改变, 运行过程中有的 APP 经常弹出授权按钮. 监视器通过页面分析, 可以自动授权, 使得遍历程序继续运行.

2.5 日志分析

Android 手机的屏幕刷新率现在是 60 Hz, 也就是在 1000 ms 内需要生成 60 张图像来保证页面使用过程的流畅性. 因此 Android 需要在 16 ms 就发出一次 Vsync 信号, 进行图像的绘制, 否则就会导致使用过程中画面不流畅. 本框架通过对页面的分析、聚合以获得更多的 APP 运行页面和状态, 在遍历过程当中, 通过监视器, 监控渲染时长, 以 16 ms 为标准, 判断页面是否出现延迟. 在遍历过程中, 我们通过对帧数的监控, 发现性能问题.

当遍历结束之后, 会得到行为的记录信息文件, 记录页面帧数异常的文件, 以及运行操作时的运行截图, 以及状态信息. 所有文件都以时间戳作为标志, 用于恢复运行时的场景. 其中页面帧数异常的文件由 xposed 拦截的函数运行时间戳分析得到. 展现的是帧数异常的时间戳信息. 将这个信息和行为信息结合在一起分析, 即可以得到是哪些操作引起的异常. 在运行过程当中, 程序也会记录在每个 Activity 中发现了多少状态, 提供对测试效果的参考值. 通过行为操作记录, 可以恢复出 APP 的状态, 并可以获取造成延迟问题操作的前驱操作和状态, 提供更多的信息.

3 实验

我们实验环境使用的是 MacBook Pro, CPU 为 i7-6700H, 内存大小 16 G, 硬盘为 1 TB SSD. 测试手机使用的是 Samsung Galaxy NOTE Ⅱ, Android 系统为 4.1, 预装有 Xposed 框架.

我们设定每个 APP 的测试时间为 15 分钟, 测试次数为 2 次. 随机在 F-Droid 上抽取的 50 个 APP 作为样本进行实验, 同时也在 github 上获取了对应 APP 的源码. 以方便在测试到问题以后对源码进行分析.

在实验当中, 我们通过对日志的分析, 总共发现有 7 个 APP 存在性能问题, 详细数据如表 1 所示.

表 1 性能测试报告以及原因简析

APP 名称	VSync 监控	Activity 监控	原因
xbmc-Kore	异常	异常	主线程异常
k9mail	异常	异常	主线程异常
ccrama	异常	正常	滑动时异常
pocmo-Yaaic	异常	正常	主线程存在 IO 操作
Swati4star-Images-to-PDF	异常	异常	主线程异常
NightWhistler-PageTurner	异常	异常	主线程异常
zagaberoo-diode	异常	正常	线程异常

在实验中, 我们通过分析日志, 发现异常信息后, 对源码进行了查看. 除了 ccrama 这个 APP 以外, 我们

均发现其余 APP 在主线程中执行时间过长, 同时在 Xposed 提供的日志中发现了信号超过 16ms 的情况, 即在运行过程中, 页面发生了不流畅的现象, 存在性能问题, 给用户带来不良的使用体验. 其中在 ccrama 的操作过程中, 我们并没有发现任何执行时间过长的问题, 却在 VSync 中发现了异常. 根据对日志时间戳的进一步分析, 我们发现是在该 APP 滑动的过程中, 发生了页面不流畅的问题, 而并非是传统的主线程异常. 而在其他的 APP 中, 我们同样对源码进行了分析, 确定了异常的原因, 是主线程中大图像加载导致的, 或者是在主线程中的阻塞线程, 以及不合理的多线程问题引起的.

在抽取的 APP 当中, 也有 2 个发生了无法进行遍历, 正常执行的情况. 我们查看原因, 发现这两个 APP 并不是传统的 APP, 而是使用了游戏引擎控件, 导致我们无法通过 UiAutomator 抓取并分析页面, 因此无法获取可以被点击的控件, 最终导致遍历无法进行, 并在测试过程中一直停留在初始页面.

通过实验, 可以体现本文提出的自动化框架通过动态执行, 可以遍历到 APP 的各个状态, 并通过 Log 日志分析, 发现性能问题. 具有良好的实用价值.

4 结语

本文提出了一个自动化的 Android 应用软件的性能性能测试框架, 既考虑到 Android 运行状态的多样性, 也考虑到系统事件的调用, 使用了随机遍历的方式, 相对于功能性测试, 更侧重遍历页面的状态, 发现 APP 在运行过程中产生不同的页面状态. 通过对运行状态的暂存, 我们可以快速地从出现性能问题的状态中实现场景恢复, 方便开发人员快速定位问题.

在实验过程中, 我们发现了不能进行遍历的情况的 APP, 是由于无法被 UiAutomator 捕获到正确的页面, 导致无法进行页面分析所引起的. 在以后的工作中, 我们可以考虑使用图像识别, OCR 技术等方式, 尝试在非传统 APP 上对页面进行分析, 然后抽象出对应的页面状态, 用这种方法替换我们现在所依赖 UiAutomator 的方式, 在该类 APP 上执行遍历行为.

本工具易于使用, 仅需配置基本的 Java 环境和 ADB 环境即可使用, 无需配置其他信息. 本工具还支持多线程测试多台实体机, 提高效率. 从实验中可以看出我们有效的检测到性能问题, 同时结合了遍历的策略, 弥补了 Monkey 的不足, 优化现有动态执行工具不能自动化的不足, 具有良好的使用价值和工业用途.

参考文献

1 Linares-vásquez M, Vendome C, Luo Q, *et al*. How developers detect and fix performance bottlenecks in Android apps. IEEE International Conference on Software Maintenance and Evolution. Bremen, Germany. 2015. 352–361.

2 Hecht G, Moha N, Rouvoy R. An empirical study of the performance impacts of Android code smells. IEEE/ACM International Conference on Mobile Software Engineering and Systems. Austin, TX, USA. 2017. 59–69.

3 Liu YP, Xu C, Cheung SC. Characterizing and detecting performance bugs for smartphone applications. Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India. 2014. 1013–1024.

4 Hussein A, Payer M, Hosking A, *et al*. Impact of GC design on power and performance for Android. ACM International Systems and Storage Conference. Haifa, Israel. 2015. 13.

5 Reaves B, Bowers J, Gorski III SA, *et al*. droid: Assessment and evaluation of Android application analysis tools. ACM Computing Surveys, 2016, 49(3): 55.

6 Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. Lincoln, NE, USA. 2015. 429–440.

7 Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of android apps. ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications. Indianapolis, IN, USA. 2013. 641–660.

8 Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android apps. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg, Russia. 2013. 224–234.

9 Google. Monkey. http://developer.android.com/tools/help/monkey.html.

10 Amalfitano D, Fasolino AR, Tramontana P, *et al*. Using GUI ripping for automated testing of Android applications. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany. 2012. 258–261.

11 Su T, Meng GZ, Chen YT, *et al*. Guided, stochastic model-based GUI testing of Android apps. Proceedings of ESEC/FSE 2017. Paderborn, Germany. 2017. 245–256.

12 Google. UiAutomator. http://android.toolib.net/tools/help/uiautomator/index.html.

13 Google. Android debugger bridge. http://developer.android.com/tools/help/adb.html.

14 Xposed module repository. http://repo.xposed.info/.