南京大学

本 科 毕 业 论 文

院　　系　软件学院

专　　业　软件工程

题　　目　安卓不可见控件内存泄漏的自动检测

年　　级　**2016** 级　学　　号　**161250136**

学生姓名　王冬杨

指导老师　马骏　职　　称　副教授

提交日期　**2020** 年 **4** 月 **20** 日

南京大学本科生毕业论文 (设计、作品) 中文摘要

题目：安卓不可见控件内存泄漏的自动检测
院系：软件学院
专业：软件工程
本科生姓名：王冬杨
指导老师（姓名、职称）：马骏副教授
摘要：

在安卓应用中，服务和广播得到了广泛应用，提供诸如下载，数据更新，跨应用通信等功能。但由于开发者经常忽视这些不可见控件的生命周期管理，因此内存泄漏发生的几率很高。

本文将会关注新版本安卓系统（Android 8+）中公开服务（Exported Services）以及静态注册的广播接收器的内存泄露问题，阐述公开服务和广播接收器内存泄露的检测方法，并编写一套供服务开发人员使用的自动分析组件内存泄露的检测工具，最后会从应用市场（App Store）中下载真实的应用，进行内存泄漏检测和分析。

关键词： 安卓系统；内存泄漏；安卓服务；安卓广播

目　　录

插图清单

第一章　绪论

1.1　研究背景

在安卓应用中，服务（Services）和广播（Broadcast）得到了广泛的使用。服务可以在安卓应用的后台保持长期运行，提供诸如下载、数据更新等重要功能。然而，正因为服务长期运行于后台的特点，使其往往容易产生异常（Errors）。如果服务的编写人员缺少警惕性，服务中出现的错误（Bug）可能会导致诸多问题，严重者可能引起应用崩溃，甚至系统死机；广播可以实现跨应用通信，要接收来自系统或者其他应用的广播，应用需要编写广播接收器（Broadcast Receiver），广播接收器将在 UI 线程运行，因此不适合进行耗时操作，通常会在广播接收器中启动服务来进行后续的处理，因此广播接收器也可能通过服务或者自身导致内存泄漏。

安卓应用中的内存泄露指资源（内存对象、句柄、服务等）将不再被使用，但却无法被垃圾回收机制（GC）回收，同时也是服务中的一大类常见问题。服务如果出现内存泄露，将会导致内存使用量意外大幅度增加，进而使得系统效率降低，严重影响用户体验。

服务如果设定’exported:true’，则该服务可以被其他应用所调用，因此内存泄露的问题将会变得更加复杂。

由于在安卓 8 及更高的版本下，安卓操作系统的“电池优化策略”禁止跨应用启动后台服务，而这一方式在安卓 7 以及更早的版本中是可行的，因此在新版本的安卓系统中，公开服务的内存泄漏检测方法与之前的方法[1] 有所差别，也正因为禁止跨应用启动后台服务，公开服务的内存泄漏问题也得到了很大的规避。

1.2　相关工作

Erika 等人在安卓 8 之前的版本中，编写了一个检测跨应用通信安全问题的工具 Com Droid[2]，文中阐述的方法对于跨应用测试具有指导意义。

在安卓 8 之前的版本中，跨应用启动服务这一行为是被允许的，南京大学的马骏等人安卓 8 之前的版本中，实现过一个公开服务（Exported Services）内

存泄漏的检测工具 LES Droid[1]，文中采用的方式分为四步：

1. 使用 apktool 反编译工具[3] 获取被测试应用的 AndroidManifest.xml 文件，解析获取应用中所有的公开服务的包名和类名。
2. 将测试驱动应用、被测试应用通过 adb 安装到模拟器中，启动测试驱动应用。
3. 测试驱动应用重复启动、关闭被测试的服务，在满足一定测试强度之后，导出被测试应用的堆镜像文件（.hprof files）。
4. 基于 MAT 内存分析库[4] 编写堆镜像文件的分析工具，自动检测内存泄漏并统计泄露的入口等。

文中的数据指出：在 41537 个被测试应用中，共在其中 28662（69%）个应用中检测出 74831 个服务，其中 3934（13.7%）个应用拥有公开服务。经过去重、安装测试以及应用商店评分筛选，有 375 个实际测试应用，最终通过不同的测试配置，最终检测到在 18.7% 的应用中有 16.8% 的服务存在内存泄漏问题。

1.3 本文主要工作

本文旨在探索一套适用于安卓 8 以上版本的公开服务和静态注册广播接收器的内存泄漏检测方法。主要工作如下：

1. 找到在安卓 8 以上版本的安卓系统上可行的跨应用测试方法。
2. 对桩应用上进行测试，并能发现所有泄露。
3. 在应用商店中下载真实应用，进行自动化测试分析实验结果。

1.4 本文结构

本文的各章节组织结构如下：

第一章 绪论。简要说明了安卓组件内存泄漏的现象和后果。并概括地描述了检测安卓不可见控件内存泄漏的方法流程，总结了本文结构。

第二章 自动化检测工具。

第三章 实验。介绍了实验进行的配置环境，测试使用应用的来源，以及实验数据结果。

第四章 总结与讨论。总结全文工作，讨论存在的问题和今后可以继续研究的方向。

第二章　自动化检测工具

本章将介绍安卓控件启动的流程，及检测内存泄露的原理。

2.1　背景

2.1.1　安卓服务

安卓应用中的服务可以通过两种方式启动[5]：

start 方式　其他组件构造特定的 **Intent** 对象，通过调用 **startService() API** 来启动目标服务。

bind 方式　通过调用 **bindService() API** 将目标服务与特定组件绑定。被绑定的服务提供接口供其他组件与之交互。一个服务可以同时通过以上两种方式启动。

2.1.2　服务的生命周期

服务的生命周期根据启动方式不同分为两种[5]：

start 方式　通过 **startService() API** 启动的服务将会一直运行，直到调用 **stopSelf()** 方法将自己停止运行。其他组件也可以通过调用 **stopService() API** 将服务停止运行。

停止运行的服务将会被 **GC(Garbage Collector)** 回收。

bind 方式　通过 **bindService() API** 启动的服务将通过 **IBinder** 接口与其他组件进行交互，直到其他组件调用 **unbindService() API** 解除绑定。

一个服务可以同时绑定到多个组件之上，直到所有组件都解除了绑定时，该服务才会被 **GC** 回收。

每个安卓应用都关联一个 **ActivityThread** 实例，负责调度和执行该应用的各种组件。**ActivityThread** 有一个 **ArrayMap** 类型的成员变量 **mServices**，其中保存了所有没有被销毁的服务的引用。一旦某个服务的实例被销毁，其引用将会从 **mServices** 中删除。

2.1.3 服务的注册方式

Listing 1 服务的注册方式

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.example.myapplication">
    <dist:module dist:instant="true" />
    <application ...>
        ...
        <service android:name=".Service1"
            android:enabled="true"
            android:exported="true">
        </service>
        <service
            android:name = ".Service2"
            android:enabled = "true"
            android:exported = "false">
            <intent-filter>
                <category android:name = "cat1"/>
                <action android:name = "act2"/>
            </intent-filter>
        </service>
        <service android:name = ".Service3"
            android:enabled = "true"
            android:permission = "Permission1">
            <intent-filter>
                <action android:name = "act3"/>
                <category android:name = "cat2"/>
                <data android:scheme = "Scheme1"/>
                <data android:scheme = "Scheme2"/>
            </intent-filter>
        </service>
    </application>
</manifest>
```

2.2 公式

通常，每个服务都要在 **AndroidManifest.xml** 中注册一个 **<service>** 标签（参考 Listing. 1中的样例）

$$\frac{\partial L}{\partial a_k^t} = d(s)^2(y_k^t - \frac{\sum_{lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s)}{y_k^t}) \tag{2-1}$$

$$d_{0j} = \sum_{k=1}^{j} w_{\text{ins}}(a_k), \qquad \text{for } 1 \le j \le n$$

$$d_{ij} = \begin{cases} d_{i-1,j-1} & \text{for } a_j = b_i \\ \min \begin{cases} d_{i-1,j} + w_{\text{del}}(b_i) \\ d_{i,j-1} + w_{\text{ins}}(a_j) \\ d_{i-1,j-1} + w_{\text{sub}}(a_j, b_i) \end{cases} & \text{for } a_j \ne b_i \end{cases} \qquad \text{for } 1 \le i \le m, 1 \le j \le n. \tag{2-2}$$

$$\begin{aligned} \beta_T(|l'|) &= y_b^T \\ \beta_T(|l'|-1) &= y_{l_{|l|}}^T \\ \beta_T(s) &= 0, \forall s < |l'|-1 \end{aligned} \tag{2-3}$$

递归公式

$$\beta_t(s) = \begin{cases} (\beta_{t+1}(s)d(s) + \beta_{t+1}(s+1))d(s+1)\, y_{s'}^t, & if\ l_s' = b\ or\ l_{s+2}' = l_s' \\ (\beta_{t+1}(s)d(s) + \beta_{t+1}(s+1)d(s+1) + \beta_{t+1}(s+2)d(s+2))\, y_{s'}^t, & otherwise \end{cases} \tag{2-4}$$

2.3 表格

		国	内	企	业	包	括	许	多
	0	1	2	3	4	5	6	7	8
国	1	0	1	2	3	4	5	6	7
著	2	1	1	2	2	3	4	5	6

表 2-1: 编辑距离（乐文斯汀距离计算过程示例表格。字符串 “国内企业包括许多” 与 “国著名括许多” 乐文斯汀距离是 3。

2.4 算法

集束宽度可以在搜索过程中保持为一个定值，也可以根据搜索的进行而变化。搜索算法可以根据搜索的结果进行调整，比如，当以一个小的集束宽度搜索解却无法找到适合解的时候，可以增大集束宽度重新进行一次搜索。

算法 2.1 Beam Search

```
1: 将初始节点插入到集束中。
2: while 遍历未结束 do
3:     遍历集束中所有节点的后续节点。
4:     if 该节点是目标节点 then
5:         算法结束。
6:     else
7:         扩展该节点，取集束宽度的节点入堆。
8:     end if
9: end while
```

第三章　实验

3.1　实现细节

我们在 Tensorflow 框架上实现了我们的网络系统。实验在一个搭载 2.40GHz 英特尔志强 Xeon E5-2673 CPU，32GB RAM 和一块英伟达 1080Ti 12GB 显存的服务器电脑上运行。网络系统使用 Adam 训练算法。

3.2　文本分行结果

尽管如此，在局部损失切割和局部水平投影切割之后，每一个竖直段的分行结果的对应关系却很难处理。在一些特殊情况下，无法做到每一竖直段分行关系的对应。所以这两个方法不适用。

3.3　识别结果

3.3.1　准确率

我们根据数据集中人的笔迹将数据集分为了 **HWDB1-HWDB3**，并实现了 Wang 等人[6] 和 Mishra 等人[7] 的方法，通过调用百度的文字识别系统[8], 进行对比实验得到以下结果。

方法	HWDB1	HWDB2	HWDB3
Wang 等人[6]	74.0	60.0	68.0
Mishra 等人[7]	80.8	63.6	73.5
百度通用文字识别[8]	64.8	36.8	60.8
我们的方法（没有字典信息）	81.5	67.5	73.6
我们的方法	**81.8**	**67.8**	**73.9**

表 3-1: 识别准确率

第四章　总结与讨论

在本文中，我们使用预处理层-卷积层-循环卷积层-转录层网络来处理手写中文文本识别的问题。这种网络很好地结合了卷积网络和循环网络各自的优势。

参考文献

[1] JUN M, SHAOCONG L, JIANG Y, et al. LESDroid-A Tool for Detecting Exported Service Leaks of Android Applications[C] // 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC). 2018 : 244 – 24410.

[2] CHIN E, FELT A P, GREENWOOD K, et al. Analyzing inter-application communication in Android[C] // Proceedings of the 9th international conference on Mobile systems, applications, and services. 2011 : 239 – 252.

[3] CONNOR TUMBLESON R W. A Tool for Reverse Engineering Android Apk Files[K/OL]. 2019 [2020-04-06]. https://ibotpeaches.github.io/Apktool/.

[4] AndrodMat 2012.[K/OL]. 2012 [2020-04-06]. https://github.com/joebowbeer/andromat.

[5] Android Sercice Guide[K/OL]. 2020 [2020-04-20]. https://developer.android.com/guide/components/services.html.

[6] WANG T, WU D J, COATES A, et al. End-to-end text recognition with convolutional neural networks[C] // Pattern Recognition (ICPR), 2012 21st International Conference on. 2012 : 3304 – 3308.

[7] MISHRA A, ALAHARI K, JAWAHAR C. Scene text recognition using higher order language priors[C] // BMVC 2012-23rd British Machine Vision Conference. 2012.

[8] BAIDU. BAIDU Text Recognition[EB/OL]. BAIDU, 2018 (2018/05/10) [2018/05/10]. https://cloud.baidu.com/product/ocr.html.

致　　谢

感谢在实验室度过的两年时光，老师无论在学术还是人生的指导上都对我起到了很大的帮助；师兄师姐小伙伴们的鼓励支持和陪伴是我坚持下去的动力。