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 **21** 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

目　录 ····· III

1 绪论 ····· 1
1.1 研究背景 ····· 1
1.2 相关工作 ····· 1
1.3 本文主要工作 ····· 2
1.4 本文结构 ····· 2

2 自动化检测工具 ····· 5
2.1 安卓服务 ····· 5
2.1.1 服务的生命周期 ····· 5
2.1.2 服务的注册方式 ····· 5
2.1.3 服务的内存泄漏 ····· 6
2.2 安卓广播接收器 ····· 7
2.2.1 广播接收器的生命周期 ····· 8
2.2.2 广播接收器的注册方式 ····· 8
2.2.3 广播接收器的内存泄漏 ····· 8
2.3 自动化检测工具原理 ····· 10

3 总结与讨论 ····· 11

参考文献 ····· 13

致　谢 ····· 15

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　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。

2.1.1　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。

2.1.2　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>
```

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）。同时服务可以通过设置”**android:exported**” 属性来指定该服务是否将被导出。当设置 **android:exported = ”true”** 时，该服务可以被其他应用使用，反之不可。

2.1.3 服务的内存泄漏

通常，一个服务的实例不再被使用时应该被 **GC(Garbage Collector)** 回收，并释放资源。然而在某些情况下，一个被销毁的服务可能会意外的被引用，从而使得 **GC** 无法将其回收并释放资源，这样就造成了服务的内存泄漏。

例如在游戏 *com.siendas.games2048* 中，就出现了原理如图（见 **Listing. 2**）的内存泄漏。具体导致内存泄露的原理为：在 **LeakedService** 的实例被构造的时候，将会调用他的 **onCreate()** 方法，在该方法中延迟 **1000ms** 启动了一个匿

Listing 2 服务的内存泄漏

```
public class LeakedService extends Service{
    private static final String TAG = "LeakedService";
    // Method will be called when an instance is creating.
    public void onCreate(){
        ...
        new Timer().scheduleAtFixedRate(new TimerTask(){
            public void run(){
                Log.d(TAG, LeakedService.this.getPackageName()
                ↪  + ".LeakedService is running!");
            }
        },1000L,3000L);
    }
    // Method will be called when an instance is destroying.
    public void onDestroy(){
        ...
    }
}
```

名计时器，该计时器将以 **3000ms** 的周期打印调试信息，可以看到在 **TimerTask** 类中持有了 **LeakedService** 的引用，而在该服务被销毁时，其 **onDestroy()** 方法中并没有对该匿名计时器进行销毁。因此在该服务被销毁后，将会一直存在一个匿名计时器持有该服务的引用，导致 **GC** 无法将其回收，从而导致了内存泄漏。

2.2 安卓广播接收器

安卓应用中的广播接收器亦有两种方式启动[6]:

清单声明的接收器 通过在 **AndroidManifest.xml** 中添加 **<receiver>** 标签注册广播接收器，通过 **<intent-filter>** 标签指定接收器所订阅的广播操作。系统软件包管理器会在应用安装时注册接收器。然后，该接收器会成为应用的一个独立入口点，这意味着如果应用当前尚未运行，系统可以启动应用并发送广播。系统会创建新的 **BroadcastReceiver** 组件对象来处理它接收到的每个广播。次对象仅在调用 **onReceive(Context, Intent)** 期间有效。一旦从此方法返回代码，系统便会认为该组件不再活跃。

上下文注册的接收器 通过在代码中构造出 **BroadReceiver** 实例，以及 **IntentFilter** 实例来指定订阅的广播内容，调用 **registerReceiver(BroadcastReceiver, IntentFilter) API** 来注册接收器。只要上下文有效，通过改种方式注册的广播接收器就会接收广播。如果要停止接收广播，需要调用 **unregister-**

Receiver(BroadcastReceiver) API 来注销广播接收器

2.2.1 广播接收器的生命周期

广播接收器的生命周期根据启动方式不同亦分为两种[6]：

清单声明的接收器 静态注册的接收器生命周期不限于 **Activity** 甚至整个应用。即使应用并不在运行，接收器也可以接收到订阅的广播。将会在 **onReceive()** 方法结束后被销毁。

上下文注册的接收器 上下文注册的接收器，其生命周期仅限于注册的上下文，例如在 **Activity** 上下文注册的接收器，在整个 **Activity** 存活期间可以持续接收广播；在应用上下文中注册的接收器，则会在整个应用运行期间都可以接收广播。需要注意的是：这种方式启动的接收器必须手动进行销毁，即调用 **unregisterReceiver() API**，否则在上下文失效时，系统会抛出异常（并不会导致应用崩溃），同时接收器会引发泄露（见图. 2-1）。

```
2020-04-21 18:06:39.557 17978-17978/com.example.myapplication E/ActivityThread: Activity
 com.example.myapplication.SecondActivity has leaked IntentReceiver com.example.myapplication
 .LeakReceiver@e03b545 that was originally registered here. Are you missing a call to
 unregisterReceiver()?
```

图 2-1: 没有回收接收器将会导致异常以及泄露

2.2.2 广播接收器的注册方式

一般而言，清单声明的广播接收器（见 2.2）需要在 **AndroidManifest.xml** 文件中添加 **<receiver>** 标签（参考 **Listing. 3**），在 **<intent-filter>** 子标签中可以指定订阅的广播内容等，也可以通过设置**"android:exported"** 属性来指定该广播是否将被导出。而上下文注册的广播接收器（见 2.2）则不需要进行前文的操作。

2.2.3 广播接收器的内存泄漏

广播接收器的内存泄漏原理类似与服务内存泄漏 2.1.3。但是由广播接收器引起的内存泄漏往往比服务更为严重，因为广播接收器被系统认为只进行不耗时的操作（如果超过 10s 未从 **onReceive()** 方法中返回，将抛出 **ANR Exception**），因此通常广播接收器在接到广播后，很有可能会启动其他的 **Service** 进行后续的耗时操作，进而可能会导致一连串的内存泄漏。

Listing 3 广播接收器的注册方式

```
<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 ...>
        ...
        <receiver
            android:name = ".Receiver1">
            <intent-filter>
                <action android:name = "act1" />
            </intent-filter>
        </receiver>
        <receiver
            android:name = ".Receiver2"
            android:exported = "false"
            android:enabled = "true">
            <intent-filter>
                <category android:name = "cat1" />
                <action android:name = "act2" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

例如图中（见 **Listing. 4**）所示的广播接收器，不仅本身会导致内存泄漏，而且还会启动一个会导致内存泄漏的服务（见 **Listing. 2**），因此后果将会更加严重。

Listing 4 广播接收器的内存泄漏

```
public class LeakReceiver extends BroadcastReceiver {
    private final String TAG = "LeakReceiver";
    private final int ID = new Random().nextInt();
    @Override
    public void onReceive(Context context, Intent intent) {
        ...
        context.startService(new
        ↪  Intent(context,LeakService.class));
        new Timer().scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                Log.i(TAG,LeakReceiver.this.ID + " is
                ↪  running!");
            }
        }, 1000L, 3000L);
    }
}
```

2.3 自动化检测工具原理

第三章　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] Android Broadcast Guide[K/OL]. 2020 [2020-04-21]. https://developer.android.com/guide/components/broadcasts.

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。