



# 尼古拉斯.赵四

## Android卸载程序之后跳转到指定的反馈页面

Android技术篇   jiangwei212   2个月前 (05-12)   245°C   0评论

今天去面试，一面还可以，到了第二面的时候也差不多吧，最后来了一题，说那个360被卸载之后会跳转到指定的反馈页面，是怎么弄的？这个之前没有研究过，但是这个效果是见过的。当时想到了，Android中卸载应用的时候会发送一个广播，我们可以接收到这个广播，然后处理一下。结果他来个反问句：这样可以吗？然后仔细想想，既然他这么问了，应该是有问题，在想想，发现的确是有问题，当应用被卸载了，那个接收到广播处理之后的逻辑代码放在那里执行？好吧，然后就没戏了~~

回来了，就百度了一下，果然网上似乎有相关的问题的解答，这里就将他们的步骤在细化一下了：

其实这个问题的核心就在于：应用被卸载了，如果能够做到后续的代码逻辑继续执行

我们再来仔细分析一下场景和流程

一个应用被用户卸载肯定是有理由的，而开发者却未必能得知这一重要的理由，毕竟用户很少会主动反馈建议，多半就是用得不爽就卸，如果能在被卸载后获取到用户的一些反馈，那对开发者进一步改进应用是非常有利的。目前据我所知，国内的Android应用中实现这一功能的只有360手机卫士、360平板卫士，那么如何实现这一功能的？

我们可以把实现卸载反馈的问题转化为监听自己是否被卸载，只有得知自己被卸载，才可以设计相应的反馈处理流程。以下的列表是我在研究这一问题的思路：

1、注册BroadcastReceiver，监听“android.intent.action.PACKAGE\_REMOVED”系统广播

结果：NO。未写代码，直接分析，卸载的第一步就是退出当前应用的主进程，而此广播是在已经卸载完成后才发出的，此时主进程都没有了，

去哪onReceive()呢？

2、若能收到“将要卸载XX包”的系统广播，在主进程被退出之前就抢先进行反馈处理就好了，可惜没有这样的系统广播，不过经过调研，倒是发现了一个办法，读取系统log，当日志中包含“android.intent.action.DELETE”和自己的包名时，意味着自己将要被卸载。

结果：NO。调试时发现此方法有两个缺陷，（1）点击设置中的卸载按钮即发出此Intent，此时用户尚未在弹框中确认卸载；（2）pm命令卸载不出发此Intent，意味着被诸如手机安全管家，豌豆荚等软件卸载时，无法提前得知卸载意图。

3、由于时间点不容易把控，所以干脆不依赖系统广播或log，考虑到卸载过程会删除“/data/data/包名”目录，我们可以用线程直接轮询这个目录是否存在，以此为依据判断自己是否被卸载。

结果：NO。同方法1，主进程退出，相应的线程必定退出，线程还没等到判断目录是否存在就已经被销毁了。

4、改用C端进程轮询“/data/data/包名”目录是否存在

结果：YES。借助Java端进程fork出来的C端进程在应用被卸载后不会被销毁。

解决的方案确定了，下面来看一下代码吧：

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

```

*/

#include <jni.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <android/log.h>
#include <unistd.h>
#include <sys/inotify.h>

#include "com_example_uninstalldemos_NativeClass.h"

/* 宏定义begin */
//清0宏
#define MEM_ZERO(pDest, destSize) memset(pDest, 0, destSize)

#define LOG_TAG "onEvent"

//LOG宏定义
#define LOGD(fmt, args...) __android_log_print(ANDROID_LOG_INFO, LOG_TAG, fmt, ##args)

JNIEXPORT jstring JNICALL Java_com_example_uninstalldemos_NativeClass_init(JNIEnv* env, jobject thiz) {

    //初始化log
    LOGD("init start...");

    //fork子进程，以执行轮询任务
    pid_t pid = fork();
    if (pid < 0) {
        //出错log
        LOGD("fork failed...");
    } else if (pid == 0) {
        //子进程注册"/data/data/pym.test.uninstalledobserver"目录监听器
    }
}

```

```
int fileDescriptor = inotify_init();
if (fileDescriptor < 0) {
    LOGD("inotify_init failed...");
    exit(1);
}

int watchDescriptor;
watchDescriptor = inotify_add_watch(fileDescriptor, "/data/data/com.example.uninstalldemos", IN_DELETE);
LOGD("watchDescriptor=%d", watchDescriptor);
if (watchDescriptor < 0) {
    LOGD("inotify_add_watch failed...");
    exit(1);
}

//分配缓存，以便读取event，缓存大小=一个struct inotify_event的大小，这样一次处理一个event
void *p_buf = malloc(sizeof(struct inotify_event));
if (p_buf == NULL) {
    LOGD("malloc failed...");
    exit(1);
}
//开始监听
LOGD("start observer...");
size_t readBytes = read(fileDescriptor, p_buf, sizeof(struct inotify_event));

//read会阻塞进程，走到这里说明收到目录被删除的事件，注销监听器
free(p_buf);
inotify_rm_watch(fileDescriptor, IN_DELETE);

//目录不存在log
LOGD("uninstall");

//执行命令am start -a android.intent.action.VIEW -d http://shouji.360.cn/web/uninstall/uninstall.html
execlp(
```

```

        "am", "am", "start", "-a", "android.intent.action.VIEW", "-d",
        "http://shouji.360.cn/web/uninstall/uninstall.html", (char *)NULL);
//4.2以上的系统由于用户权限管理更严格，需要加上 --user 0
//execvp("am", "am", "start", "--user", "0", "-a",
// "android.intent.action.VIEW", "-d", "https://www.google.com", (char *) NULL);

    } else {
        //父进程直接退出，使子进程被init进程领养，以避免子进程僵死
    }

    return (*env)->NewStringUTF(env, "Hello from JNI !");
}

```

这里面主要是用到了Linux中的inotify,这个相关的内容可以自行百度一下~~

这里有一个很重要的知识，也是解决这个问题的关键所在，就是Linux中父进程死了，但是子进程不会死，而是被init进程领养。所以当我们应用(进程)卸载了，但是我们fork的子进程并不会销毁，所以我们上述的逻辑代码就可以放到这里来做了。(学习了)

Android应用程序代码：

MyActivity.java

```

package com.example.uninstallldemos;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class MyActivity extends Activity {

    @Override

```

```

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent intent = new Intent(this, SDCardListenSer.class);
        startService(intent);
        NativeClass nativeObj = new NativeClass();
        nativeObj.init();
    }

    static {
        Log.d("onEvent", "load jni lib");
        System.loadLibrary("hello-jni");
    }
}

```

## SDCardListenSer.java

```

package com.example.uninstallldemos;

import android.annotation.SuppressLint;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.os.Environment;
import android.os.FileObserver;
import android.os.IBinder;
import android.util.Log;
import java.io.File;
import java.io.IOException;

public class SDCardListenSer extends Service {

```

```

SDCardListener[] listeners;

@SuppressLint("SdCardPath")
@Override
public void onCreate() {
    SDCardListener[] listeners = {
        new SDCardListener("/data/data/com.example.uninstalldemos", this),
        new SDCardListener(Environment.getExternalStorageDirectory() + File.separator + "1.tx
t", this) };

    this.listeners = listeners;

    Log.i("onEvent", "=====onCreate=====");
    for (SDCardListener listener : listeners) {
        listener.startWatching();
    }

    File file = new File(Environment.getExternalStorageDirectory() + File.separator + "1.txt");
    Log.i("onEvent", "dddddddddddddddddddd nCreate=====");
    if (file.exists())
        file.delete();
    /*try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }*/
}

@Override
public void onDestroy() {
    for (SDCardListener listener : listeners) {
        listener.stopWatching();
    }
}
}

```

```

@Override
public IBinder onBind(Intent intent) {
    return null;
}
}

class SDCardListener extends FileObserver {
    private String mPath;
    private final Context mContext;

    public SDCardListener(String parentpath, Context ctx) {
        super(parentpath);
        this.mPath = parentpath;
        this.mContext = ctx;
    }

    @Override
    public void onEvent(int event, String path) {
        int action = event & FileObserver.ALL_EVENTS;
        switch (action) {

            case FileObserver.DELETE:
                Log.i("onEvent", "delete path: " + mPath + File.separator + path);
                //openBrowser();
                break;

            case FileObserver.MODIFY:
                Log.i("onEvent", "更改目录" + mPath + File.separator + path);
                break;

            case FileObserver.CREATE:
                Log.i("onEvent", "创建文件" + mPath + File.separator + path);

```



```

                break;

            default:
                break;
        }
    }

    protected void openBrowser() {
        Uri uri = Uri.parse("http://aoi.androidesk.com");
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        mContext.startActivity(intent);
    }

    public void exeShell(String cmd) {
        try {
            Runtime.getRuntime().exec(cmd);
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }
}

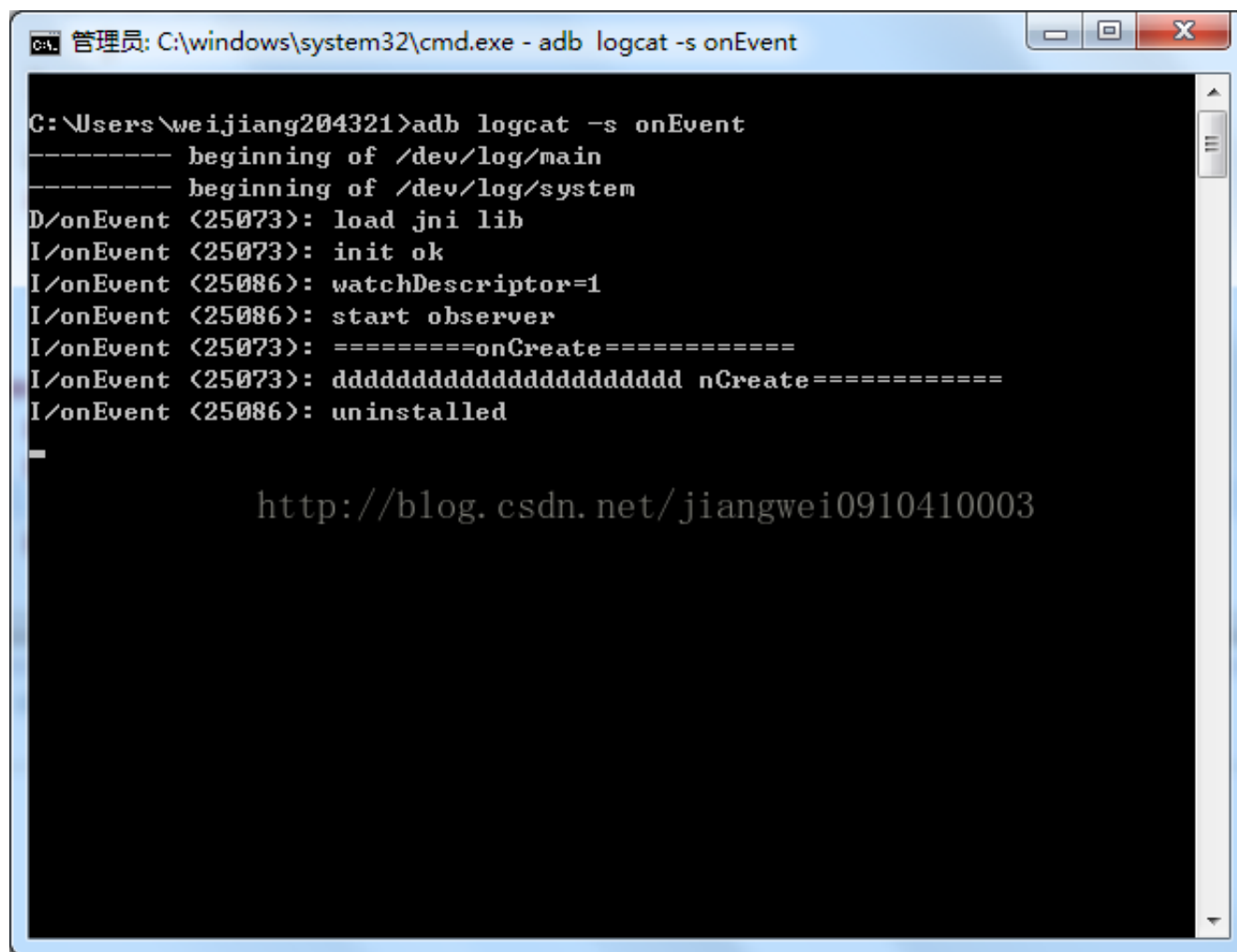
```

开启一个服务，在这个服务中我们可以看到，用到了一个很重要的一个类FileObserver，也是用来监听文件的变更的，这个和上面的inotify功能差不多。关于这个类的具体用法和介绍，可以自行百度呀~~

运行：

我们将应用安装之后，打开log进行检测日志：

adb logcat -s onEvent



The image shows a Windows command prompt window titled "管理员: C:\windows\system32\cmd.exe - adb logcat -s onEvent". The command prompt shows the execution of the command "adb logcat -s onEvent" and the resulting log output. The log output includes several lines of text, including "beginning of /dev/log/main", "beginning of /dev/log/system", "D/onEvent <25073>: load jni lib", "I/onEvent <25073>: init ok", "I/onEvent <25086>: watchDescriptor=1", "I/onEvent <25086>: start observer", "I/onEvent <25073>: =====onCreate===== ", "I/onEvent <25073>: ddddddddddddddddddddd nCreate===== ", and "I/onEvent <25086>: uninstalled". Below the log output, there is a URL: "http://blog.csdn.net/jiangwei0910410003".

```
C:\Users\weijiang204321>adb logcat -s onEvent
----- beginning of /dev/log/main
----- beginning of /dev/log/system
D/onEvent <25073>: load jni lib
I/onEvent <25073>: init ok
I/onEvent <25086>: watchDescriptor=1
I/onEvent <25086>: start observer
I/onEvent <25073>: =====onCreate===== 
I/onEvent <25073>: ddddddddddddddddddddd nCreate===== 
I/onEvent <25086>: uninstalled

http://blog.csdn.net/jiangwei0910410003
```

当我们从设置中卸载应用的时候，会弹出如下界面：



注：这里我特定说了是从设置界面中去卸载应用，因为当我使用小米手机自带的那种快捷卸载应用的时候并不会跳转。这个具体的原因还有待解决(当然360的这个问题也没有解决掉。。)

总结：

我写这篇文章的目的以及我从这个过程中唯一学习到的一个知识点就是当父进程消亡了，子进程并不会消亡，所以我们可以记住这个知识点，以后遇到像应用被卸载之后的一些逻辑操作都可以采用这种方式去解决。

[JNI的Demo下载](#)

[Android中的Demo下载](#)

**关注微信公众号，最新Android技术实时推送**




觉得文章对您有帮助，就打个赏呗！




« [Android中应用锁的实现之账号盗取](#)

[Android中自定义视图View之一前奏篇](#) »

 [Android逆向之旅—运行时修改内存中的Dalvik指令来改变代码逻辑](#)

 [Apk脱壳圣战之一脱掉“360加固”的壳](#)

 [Android逆向之旅—反编译利器Apktool和Jadx源码分析以及错误纠正](#)

 [Apk脱壳圣战之一脱掉“爱加密”家的壳](#)

[Android逆向之旅—运行时修改内存中的Dalvik指令来改变代码逻辑](#)

[Apk脱壳圣战之一脱掉“360加固”的壳](#)

[Android逆向之旅—反编译利器Apktool和Jadx源码分析以及错误纠正](#)

[Apk脱壳圣战之一脱掉“爱加密”家的壳](#)

- [Android逆向之旅—运行时修改内存中的Dalvik指令来改变代码逻辑](#)
- [Android逆向之旅—反编译利器Apktool和Jadx源码分析以及错误纠正](#)
- [Android逆向之旅—动态方式破解apk终极篇\(应对加固apk破解方案\)](#)
- [Android逆向之旅—Android应用的安全的攻防之战](#)

- [Apk脱壳圣战之一脱掉“360加固”的壳](#)
- [Apk脱壳圣战之一脱掉“爱加密”家的壳](#)
- [Android中apk加固完善篇之内存加载dex方案实现原理\(不落地方式加固\)](#)
- [Android逆向之旅—动态方式破解apk进阶篇\(IDA调试so源码\)](#)

 [发表我的评论](#)

写点什么...

