

• 首页

开源项目

问答

代码

博客

翻译

资讯

移动开发

招聘

城市圈


当前访客身份：游客 [登录 | 加入开源中国]

在 40724 款开源软件中搜

软件

软件

搜索



亭子happy

关注此人

关注(12) 粉丝(102) 积分(63)

归纳总结~~~~

[发送私信](#) [请教问题](#)

博客分类

Android(100)

Java(6)

Others(45)

Javascript(10)

life(4)

Linux(5)

Korean(2)

git(6)

杂感(2)

- 阅读排行
1. [1. 低功耗蓝牙（BLE）](#)

2. [2. Android客户端注入及清除Cookie](#)

3. [3. \[Android\]开源中国源码分析——启动界面](#)

4. [4. \[MediaStore\]小米文件管理器android版源码分析——数据来源](#)

5. [5. \[Android\]瀑布流实例android_waterfall源码分析](#)

6. [6. \[Android\]滑动刷新ListView——android-pulltorefresh使用方法解析](#)

7. [7. android_自定义toggle_Button按钮](#)

8. [8. Android事件详解——触屏事件MotionEvent（一）](#)

- 最新评论
- @Mbingley: 好

查看»

@matchbox: Android 文档特别说明了：Note: You can only sc...

查看»

@懂me: 不错

查看»

@hcg0618: Android Wear/DuWear/TicWear/TOS/YunOS For Wea...

查看»

@hcg0618: Android Wear/DuWear/TicWear/TOS/YunOS For Wea...

查看»

@久依: 多谢谢分享学习了!

查看»

@: 1元4700套安卓源码+80G安卓学习视频 新店开张 求...

查看»

@亭子happy: 引用来自“zzqzz”的评论哈哈，没写完呢吧 持续更...

查看»

@zzqzz: 哈哈，没写完呢吧

查看»

@亭子happy: 引用来自“FoxHu”的评论怎么不用Android studi...

查看»

- 访客统计
- 今日访问：17

昨日访问：275

本周访问：534

本月访问：2063

所有访问：98279

空间 » 博客 » Android

转

NotificationManagerService使用详解与原理分析（一）

发表于11个月前(2015-05-12 11:12) 阅读 (595) | 评论 (0) 0人收藏此文章, 我要收藏

赞0

4月23日，武汉源创会火热报名中，期待您的参与>>>>>>

HOT

- 目录[-/]
- 概况

重要关系

使用简介

使用详解

程序运行截图

示例介绍

功能分析

1. 如何检测应用已开启Notification access监听功能?

2. 能不能主动跳转到Notification access监听页面?

3. 如何与NotificationListenerService交互?

4. NotificationListenerService使用过程中有哪些注意事项?

总结

概况

Android在4.3的版本中(即API 18)加入了NotificationListenerService，根据SDK的描述([AndroidDeveloper](#))可以知道，当系统收到新的通知或者通知被删除时，会触发NotificationListenerService的回调方法。同时在Android 4.4 中新增了Notification.extras 字段，也就是说可以使用NotificationListenerService获取系统通知具体信息，这在以前是需要用反射来实现的。

转载请务必注明出处：<http://blog.csdn.net/yihongyuelan>

重要关系

对于系统通知，三方APP使用NotificationListenerService主要目的是为了获取系统通知相关信息，主要包括：通知的新增和删除，获取当前通知数量，通知内容相关信息等。这些信息可以通过NotificationListenerService类提供的方法以及StatusBarNotification类对象来获取。

NotificationListenerService 主要方法(成员变量):

cancelAllNotifications()：删除系统中所有可被清除的通知；
cancelNotification(String pkg, String tag, int id)：删除具体某一个通知；
getActiveNotifications()：返回当前系统所有通知到StatusBarNotification[]；
onNotificationPosted(StatusBarNotification sbn)：当系统收到新的通知后出发回调；
onNotificationRemoved(StatusBarNotification sbn)：当系统通知被删掉后出发回调；

以上是NotificationListenerService的主要方法，通过这些方法就可以在应用中操作系统通知，在NotificationListenerService中除了对通知的操作之外，还可以获取到通知的StatusBarNotification对象，通过该对象可以获取通知更详细的数据。

StatusBarNotification 主要方法(成员变量):

getId()：返回通知对应的id；
getNotification()：返回通知对象；
getPackageName()：返回通知对应的包名；
getPostTime()：返回通知发起的时间；
getTag()：返回通知的Tag，如果没有设置返回null；
getUserId()：返回UserId，用于多用户场景；
isClearable()：返回该通知是否可被清楚，FLAG_ONGOING_EVENT、FLAG_NO_CLEAR；
isOngoing()：检查该通知的flag是否为FLAG_ONGOING_EVENT；

使用简介

正确使用NotificationListenerService需要注意三点：

(1). 新建一个类并继承自NotificationListenerService，override其中重要的两个方法；

[java] [view plaincopy](#)

```
1. public class NotificationMonitor extends NotificationListenerService {
2.     @Override
3.     public void onNotificationPosted(StatusBarNotification sbn) {
4.         Log.i("SevenNLS", "Notification posted");
5.     }
6.
7.     @Override
8.     public void onNotificationRemoved(StatusBarNotification sbn) {
9.         Log.i("SevenNLS", "Notification removed");
10.    }
11. }
```

(2). 在AndroidManifest.xml中注册Service并声明相关权限：

[html] [view plaincopy](#)

```
1. <service android:name=".NotificationMonitor"
2.     android:label="@string/service_name"
3.     android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
```

4. `<intent-filter>`
5. `<action android:name="android.service.notification.NotificationListenerService" />`
6. `</intent-filter>`
7. `</service>`

(3). 开启NotificationMonitor的监听功能：

完成以上两步之后，将程序编译并安装到手机上，但此时该程序是无法监听到新增通知和删除通知的，还需要在"Settings > Security > Notification access"中，勾选NotificationMonitor。此时如果系统收到新的通知或者通知被删除就会打印出相应的log了。

这里需要注意，如果手机上没有安装使用NotificationListenerService类的APP，Notification access是不会显示出来的。可以在源码/packages/apps/Settings/src/com/android/settings/SecuritySettings.java中看到，如果没有使用NotificationListenerService的APK，直接就不显示这一项了。

[java] [view plain](#) [copy](#)

```
1. mNotificationAccess = findPreference(KEY_NOTIFICATION_ACCESS);
2. if (mNotificationAccess != null) {
3.     final int total = NotificationAccessSettings.getListenersCount(mPM);
4.     if (total == 0) {
5.         if (deviceAdminCategory != null) {
6.             deviceAdminCategory.removePreference(mNotificationAccess);
7.         }
8.     } else {
9.         final int n = getNumEnabledNotificationListeners();
10.        if (n == 0) {
11.            mNotificationAccess.setSummary(getResources().getString(
12.                R.string.manage_notification_access_summary_zero));
13.        } else {
14.            mNotificationAccess.setSummary(String.format(getResources().getQuantityString(
15.                R.plurals.manage_notification_access_summary_nonzero,
16.                n, n)));
17.        }
18.    }
19. }
```

使用详解

通过前面的讲解(实际上就是对AndroidDeveloper的解释)，已经可以正常使用NotificationListenerService了，但对于实际应用中，需要考虑的事情还比较多。比如：

1. 如何检测应用已开启Notification access监听功能？

如果检测到应用没有激活Notification access监听功能，需要提示用户开启：

2. 能不能主动跳转到Notification access监听页面？

如果能够根据第1步的判断自动跳转到对应的页面，那可以省掉很多操作：

3. 如何与NotificationListenerService交互？

涉及到与Service的交互，但又与普通的Service不同，这里后文解释：

4. NotificationListenerService使用过程中有哪些注意事项？

在使用NotificationListenerService过程中自己遇到了一些坑，后文会通过分析给出相应的解决方案：

程序运行截图

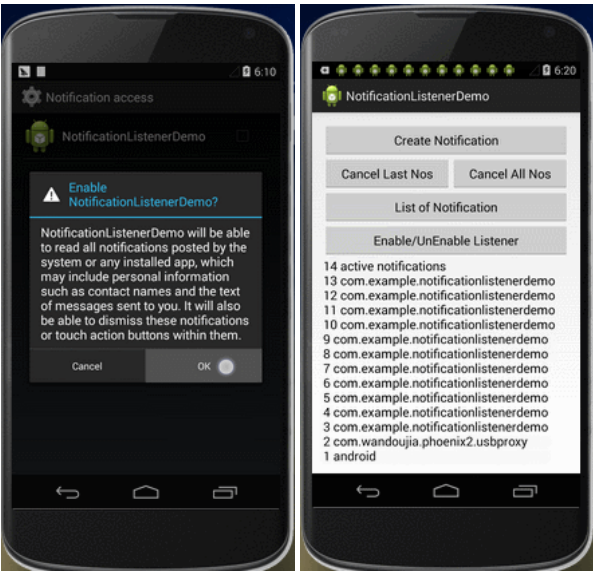


图 1 程序运行截图

示例介绍

NotificationListenerDemo主要用于获取系统当前通知信息，并可手动创建"可清除通知"，逐条删除"可清除通知"，一次性删除"可清除通知"，以及显示系统当前活动的通知信息。实际上该示例回答了前面使用详解中提出的各项疑问，在实际使用过程中相信大部分人都会遇到，因此这里逐条展开与大家分享。

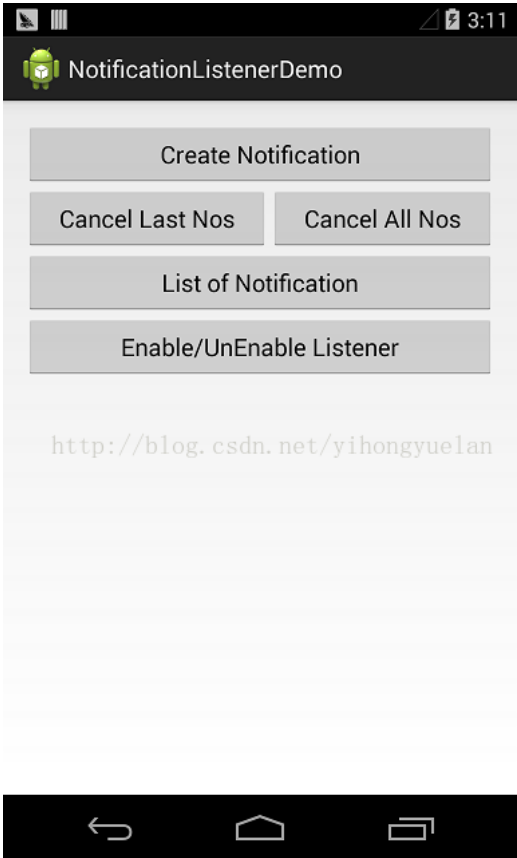


图 2 主界面

功能分析

1. 如何检测应用已开启 **Notification access** 监听功能？

在程序启动时，执行**Notification access**的检测，查看是否访问**Notification**的权限。如果用户没有**Enable Notification access**，则弹出提示对话框，点击**OK**跳转到**Notification access**设置页面。

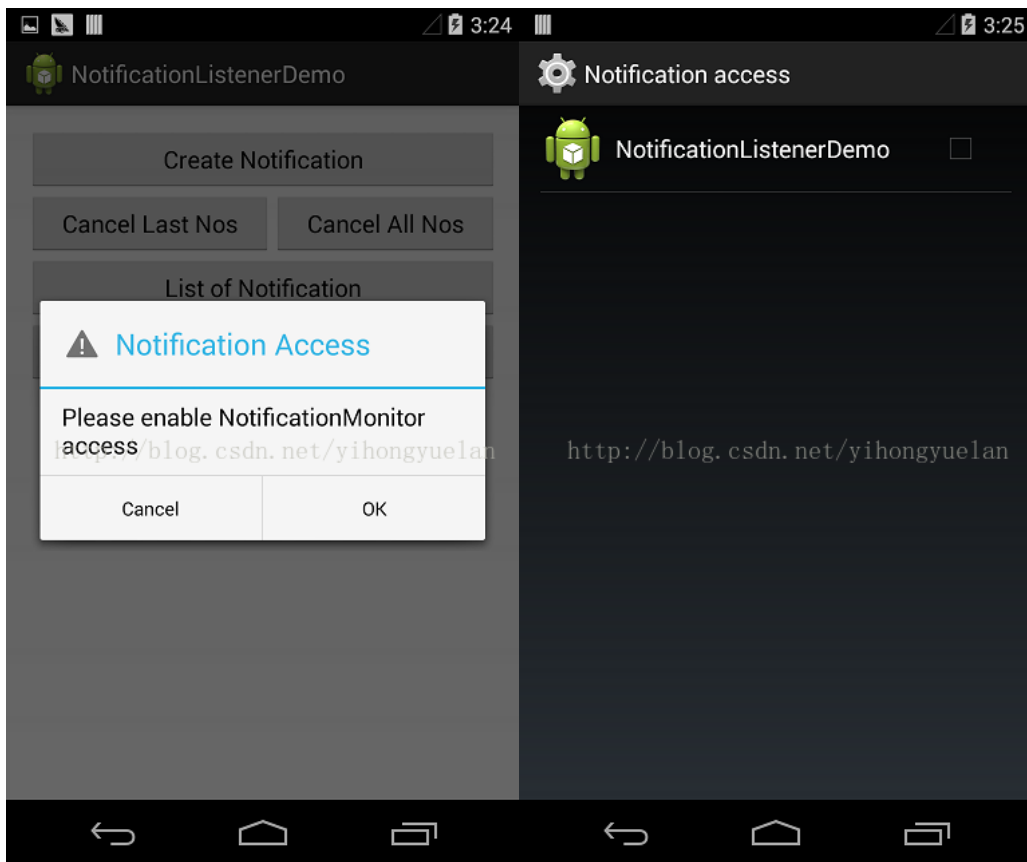


图 3 首次启动 isEnable

使用NotificationListenerService的应用如果开启了Notification access，系统会将包名等相关信息写入SettingsProver数据库中，因此可以从数据库中获取相关信息并过滤，从而判断应用是否开启了Notification access，代码如下：

[java] [view plaincopy](#)

```

1. private static final String ENABLED_NOTIFICATION_LISTENERS = "enabled_notification_listeners";
2. private boolean isEnabled() {
3.     String pkgName = getPackageName();
4.     final String flat = Settings.Secure.getString(getContentResolver(),
5.         ENABLED_NOTIFICATION_LISTENERS);
6.     if (!TextUtils.isEmpty(flat)) {
7.         final String[] names = flat.split(":");
8.         for (int i = 0; i < names.length; i++) {
9.             final ComponentName cn = ComponentName.unflattenFromString(names[i]);
10.            if (cn != null) {
11.                if (TextUtils.equals(pkgName, cn.getPackageName())) {
12.                    return true;
13.                }
14.            }
15.        }
16.    }
17.    return false;
18. }

```

在返回值flat中如果包含了应用的包名，即可确定应用已开启Notification access，反之则表示没有开启。

2. 能不能主动跳转到Notification access监听页面？

通过查看可以知道，Notification access界面接收action为"android.settings.ACTION_NOTIFICATION_LISTENER_SETTINGS"的intent启动，因此使用startActivity可以很容易的跳转到该页面，从而避免用户在Settings中查找。代码如下：

[java] [view plaincopy](#)

```
1. private static final String ACTION_NOTIFICATION_LISTENER_SETTINGS = "android.settings.ACTION_NOTIFICATION_LISTEN
2. private void openNotificationAccess() {
3.     startActivity(new Intent(ACTION_NOTIFICATION_LISTENER_SETTINGS));
4. }
```

3. 如何与NotificationListenerService交互？

因为NotificationListenerService中包含了四个重要的方法，分别是：onNotificationPosted、onNotificationRemoved、cancelNotification、cancelAllNotifications。通过这些方法我们才能实现诸如通知信息的获取以及删除等功能，虽然这些方法是public的，那是不是意味着我们只要拿到NotificationListenerService的对象就可以直接调用这些方法了呢？那如何拿到Service的对象呢？在之前的博文中，曾有提到与Service的交互(具体可参考拙作《[Android中程序与Service交互的方式——交互方式](#)》)，可以看到与Service的交互有很多种方法，但如果要拿到Service的对象，归根到底还是需要Binder。

也就是说得使用bindService的办法，将onServiceConnected回调中的IBinder对象转型成NotificationListenerService的对象。测试代码如下：

[java] [view plaincopy](#)

```
1. //在MainActivity.java的onCreate方法中使用bindService帮顶NotificationMonitor服务
2. bindService(new Intent(this,NotificationMonitor.class ), new ServiceConnection() {
3.     @Override
4.     public void onServiceDisconnected(ComponentName arg0) {
5.     }
6.
7.     @Override
8.     public void onServiceConnected(ComponentName arg0, IBinder arg1) {
9.         NotificationMonitor.MyBinder localBinder = (MyBinder)arg1;
10.        NotificationMonitor mMonitor = localBinder.getService();
11.    }
12. }, BIND_AUTO_CREATE);
```

[java] [view plaincopy](#)

```
1. //NotificationMonitor的onBind方法返回构造的Binder对象
2. public class NotificationMonitor extends NotificationListenerService {
3.     private MyBinder mBinder = new MyBinder();
4.     public class MyBinder extends Binder{
5.         public NotificationMonitor getService(){
6.             return NotificationMonitor.this;
7.         }
8.     }
9.
10.    @Override
11.    public IBinder onBind(Intent arg0) {
12.        return mBinder;
13.    }
14.
15.    @Override
16.    public void onNotificationPosted(StatusBarNotification sbn) {
17.        getActiveNotifications();
18.        cancelAllNotifications();
19.    }
20.
21.    @Override
```

```
22. public void onNotificationRemoved(StatusBarNotification sbn) {  
23. }  
24. }
```

那这样操作之后是不是就意味着可以拿到NotificationMonitor的对象并直接调用getActiveNotifications()方法，用于获取当前系统通知的信息了呢？很抱歉，事实证明这样是不行的。这里简单的分析下，在后面的NotificationListenerService原理分析中再详细讲解。在NotificationListenerService的源码中可以看到：

[java] [view plaincopy](#)

```
1. @Override  
2. public IBinder onBind(Intent intent) {  
3.     if (mWrapper == null) {  
4.         mWrapper = new INotificationListenerWrapper();  
5.     }  
6.     return mWrapper;  
7. }
```

这里的INotificationListenerWrapper是NotificationListenerService的一个内部类：

[java] [view plaincopy](#)

```
1. private class INotificationListenerWrapper extends INotificationListener.Stub
```

而NotificationMonitor继承自NotificationListenerService，默认的onBind方法却是：

[java] [view plaincopy](#)

```
1. @Override  
2. public IBinder onBind(Intent intent) {  
3.     return super.onBind(intent);  
4. }
```

这里注意，一般情况下service的onBind方法返回要么是null要么是Binder对象，可这里直接调用父类NotificationListenerService的onBind方法，而父类返回的是INotificationListenerWrapper的对象。这说明Binder对象已经被指定了，不能再给NotificationMonitor指定其它的Binder对象。如果你非要给NotificationMonitor指定其它的Binder对象，那么就无法使用INotificationListenerWrapper提供的方法。也就是说要么就用系统NotificationListenerService提供的方法，要么就把NotificationMonitor当一个普通的Service来用，系统提供的方法都不能使用。

那应该如何使用NotificationListenerService中的方法呢？在拙作《[Android中程序与Service交互的方式——交互方式](#)》中，已经提供了很多的例子，这里仅以广播的方式为例。

既然NotificationMonitor可以使用NotificationListenerService的方法，那通过NotificationMonitor把通知状态的改变以及数据获取到，并使用static数据进行存储，之后再在MainActivity中直接使用即可。在MainActivity中控制通知的单个删除和全部删除，则使用广播的方式发送给NotificationMonitor进行处理。MainActivity与NotificationMonitor的关系类图如下：

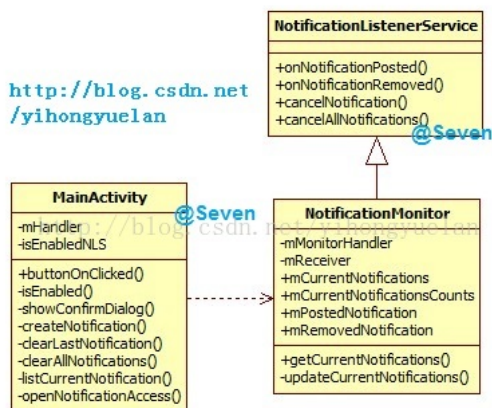


图 4 结构类图

NotificationMonitor和MainActivity关键代码如下：

[java] [view plaincopy](#)

```
1. public class NotificationMonitor extends NotificationListenerService {  
2.     private static final String TAG = "SevenNLS";
```

```
3.  private static final String TAG_PRE = "[" + NotificationMonitor.class.getSimpleName() + "] ";
4.  private static final int EVENT_UPDATE_CURRENT_NOS = 0;
5.  public static final String ACTION-NLS_CONTROL = "com.seven.notificationlistenerdemo.NLSCONTROL";
6.  //用于存储当前所有的Notification的StatusBarNotification对象数组
7.  public static List<StatusBarNotification[]> mCurrentNotifications = new ArrayList<StatusBarNotification[]>();
8.  public static int mCurrentNotificationsCounts = 0;
9.  //收到新通知后将通知的StatusBarNotification对象赋值给mPostedNotification
10. public static StatusBarNotification mPostedNotification;
11. //删除一个通知后将通知的StatusBarNotification对象赋值给mRemovedNotification
12. public static StatusBarNotification mRemovedNotification;
13. private CancelNotificationReceiver mReceiver = new CancelNotificationReceiver();
14. // String a;
15. private Handler mMonitorHandler = new Handler() {
16.     @Override
17.     public void handleMessage(Message msg) {
18.         switch (msg.what) {
19.             case EVENT_UPDATE_CURRENT_NOS:
20.                 updateCurrentNotifications();
21.                 break;
22.             default:
23.                 break;
24.         }
25.     }
26. };
27.
28. class CancelNotificationReceiver extends BroadcastReceiver {
29.
30.     @Override
31.     public void onReceive(Context context, Intent intent) {
32.         String action;
33.         if (intent != null && intent.getAction() != null) {
34.             action = intent.getAction();
35.             if (action.equals(ACTION-NLS_CONTROL)) {
36.                 String command = intent.getStringExtra("command");
37.                 if (TextUtils.equals(command, "cancel_last")) {
38.                     if (mCurrentNotifications != null && mCurrentNotificationsCounts >= 1) {
39.                         //每次删除通知最后一个
40.                         StatusBarNotification sbnn = getCurrentNotifications()[mCurrentNotificationsCounts - 1];
41.                         cancelNotification(sbnn.getPackageName(), sbnn.getTag(), sbnn.getId());
42.                     }
43.                 } else if (TextUtils.equals(command, "cancel_all")) {
44.                     //删除所有通知
45.                     cancelAllNotifications();
46.                 }
47.             }
48.         }
```



```
49.     }
50.
51. }
52.
53. @Override
54. public void onCreate() {
55.     super.onCreate();
56.     logNLS("onCreate...");
57.     IntentFilter filter = new IntentFilter();
58.     filter.addAction(ACTION-NLS-CONTROL);
59.     registerReceiver(mReceiver, filter);
60.     //在onCreate时第一次调用getActiveNotifications()
61.     mMonitorHandler.sendMessage(mMonitorHandler.obtainMessage(EVENT_UPDATE_CURRENT_NOS));
62. }
63.
64. @Override
65. public void onDestroy() {
66.     super.onDestroy();
67.     unregisterReceiver(mReceiver);
68. }
69.
70. @Override
71. public IBinder onBind(Intent intent) {
72.     // a.equals("b");
73.     logNLS("onBind...");
74.     return super.onBind(intent);
75. }
76.
77. @Override
78. public void onNotificationPosted(StatusBarNotification sbn) {
79.     //当系统收到新的通知后，更新mCurrentNotifications列表
80.     updateCurrentNotifications();
81.     logNLS("onNotificationPosted...");
82.     logNLS("have " + mCurrentNotificationsCounts + " active notifications");
83.     mPostedNotification = sbn;
84.     //通过以下方式可以获取Notification的详细信息
85.     /*
86.      * Bundle extras = sbn.getNotification().extras; String
87.      * notificationTitle = extras.getString(Notification.EXTRA_TITLE);
88.      * Bitmap notificationLargeIcon = ((Bitmap)
89.      * extras.getParcelable(Notification.EXTRA_LARGE_ICON)); Bitmap
90.      * notificationSmallIcon = ((Bitmap)
91.      * extras.getParcelable(Notification.EXTRA_SMALL_ICON)); CharSequence
92.      * notificationText = extras.getCharSequence(Notification.EXTRA_TEXT);
93.      * CharSequence notificationSubText =
94.      * extras.getCharSequence(Notification.EXTRA_SUB_TEXT);
```

```
95.     * Log.i("SevenNLS", "notificationTitle:"+notificationTitle);
96.     * Log.i("SevenNLS", "notificationText:"+notificationText);
97.     * Log.i("SevenNLS", "notificationSubText:"+notificationSubText);
98.     * Log.i("SevenNLS",
99.     * "notificationLargeIcon is null:"+(notificationLargeIcon == null));
100.    * Log.i("SevenNLS",
101.    * "notificationSmallIcon is null:"+(notificationSmallIcon == null));
102.    */
103. }
104.
105. @Override
106. public void onNotificationRemoved(StatusBarNotification sbn) {
107.     //当有通知被删除后，更新mCurrentNotifications列表
108.     updateCurrentNotifications();
109.     logNLS("removed...");
110.     logNLS("have " + mCurrentNotificationsCounts + " active notifications");
111.     mRemovedNotification = sbn;
112. }
113.
114. private void updateCurrentNotifications() {
115.     try {
116.         StatusBarNotification[] activeNos = getActiveNotifications();
117.         if (mCurrentNotifications.size() == 0) {
118.             mCurrentNotifications.add(null);
119.         }
120.         mCurrentNotifications.set(0, activeNos);
121.         mCurrentNotificationsCounts = activeNos.length;
122.     } catch (Exception e) {
123.         logNLS("Should not be here!!");
124.         e.printStackTrace();
125.     }
126. }
127.
128. //获取当前状态栏显示通知总数
129. public static StatusBarNotification[] getCurrentNotifications() {
130.     if (mCurrentNotifications.size() == 0) {
131.         logNLS("mCurrentNotifications size is ZERO!!");
132.         return null;
133.     }
134.     return mCurrentNotifications.get(0);
135. }
136.
137. private static void logNLS(Object object) {
138.     Log.i(TAG, TAG_PRE + object);
139. }
140.
```



```
141. }
```

而MainActivity主要负责界面显示与交互，关键代码如下：

[java] [view plain](#)copy

```
1. public class MainActivity extends Activity {
2.
3.     private static final String TAG = "SevenNLS";
4.     private static final String TAG_PRE = "["+MainActivity.class.getSimpleName()+"] ";
5.     private static final int EVENT_SHOW_CREATE_NOS = 0;
6.     private static final int EVENT_LIST_CURRENT_NOS = 1;
7.     private static final String ENABLED_NOTIFICATION_LISTENERS = "enabled_notification_listeners";
8.     private static final String ACTION_NOTIFICATION_LISTENER_SETTINGS = "android.settings.ACTION_NOTIFICATION_LIST
9.     private boolean isEnabledNLS = false;
10.    private TextView mTextView;
11.
12.    private Handler mHandler = new Handler() {
13.        @Override
14.        public void handleMessage(Message msg) {
15.            switch (msg.what) {
16.                case EVENT_SHOW_CREATE_NOS:
17.                    //显示创建的Notification对应的pkgName、Tag、Id
18.                    showCreateNotification();
19.                    break;
20.                case EVENT_LIST_CURRENT_NOS:
21.                    //显示当前所有的Notification数量及其包名
22.                    listCurrentNotification();
23.                    break;
24.
25.                default:
26.                    break;
27.            }
28.        }
29.    };
30.
31.    @Override
32.    protected void onCreate(Bundle savedInstanceState) {
33.        super.onCreate(savedInstanceState);
34.        setContentView(R.layout.activity_main);
35.        mTextView = (TextView) findViewById(R.id.textview);
36.    }
37.
38.    @Override
39.    protected void onResume() {
40.        super.onResume();
41.        //判断是否有开启Notification access
42.        isEnabledNLS = isEnabled();
43.        logNLS("isEnabledNLS = " + isEnabledNLS);
```

```
44.     if (!isEnabledNLS) {
45.         //如果没有开启则显示确认对话框
46.         showConfirmDialog();
47.     }
48. }
49.
50. public void buttonOnClicked(View view) {
51.     mTextView.setTextColor(Color.BLACK);
52.     switch (view.getId()) {
53.         case R.id.btnCreateNotify:
54.             logNLS("Create notifications...");
55.             //创建可清除的Notification
56.             createNotification(this);
57.             //显示当前状态栏中所有Notification数量及其包名
58.             mHandler.sendMessageDelayed(mHandler.obtainMessage(EVENT_SHOW_CREATE_NOS), 50);
59.             break;
60.         case R.id.btnClearLastNotify:
61.             logNLS("Clear Last notification...");
62.             //清除最后一个Notification
63.             clearLastNotification();
64.             //显示当前状态栏中所有Notification数量及其包名
65.             mHandler.sendMessageDelayed(mHandler.obtainMessage(EVENT_LIST_CURRENT_NOS), 50);
66.             break;
67.         case R.id.btnClearAllNotify:
68.             logNLS("Clear All notifications...");
69.             //清除所有"可被清除"的Notification
70.             clearAllNotifications();
71.             mHandler.sendMessageDelayed(mHandler.obtainMessage(EVENT_LIST_CURRENT_NOS), 50);
72.             break;
73.         case R.id.btnListNotify:
74.             logNLS("List notifications...");
75.             listCurrentNotification();
76.             break;
77.         case R.id.btnEnableUnEnableNotify:
78.             logNLS("Enable/UnEnable notification...");
79.             //打开Notification access启动/取消界面
80.             openNotificationAccess();
81.             break;
82.         default:
83.             break;
84.     }
85. }
86.
87. //.....省略
88. }
```

4. NotificationListenerService使用过程中有哪些注意事项？

如果细心察看代码的童鞋，一定发现代码中有使用Handler，以及一些奇怪但又被注释掉的代码，比如"a.equals("b")"。从使用上来说，没有必要使用handler，那干嘛要多次一举？这里就给大家分享一下在写NotificationListenerDemo时遇到的一些坑。

①. NotificationMonitor的onCreate方法中使用handler来调用getActiveNotifications()方法

若直接在onCreate或者onBind方法中调用getActiveNotifications()方法是无法获取当前系统通知。主要是因为NotificationMonitor还未完成初始化，而根本原因则是INotificationListenerWrapper对象mWrapper还未初始化，此时使用getActiveNotifications()方法又会调用到mWrapper，因此无法返回正常数据。在NotificationListenerService中可以看到getActiveNotifications()的源码：

[java] [view plaincopy](#)

```
1. public StatusBarNotification[] getActiveNotifications() {
2.     try {
3.         return getNotificationInterface().getActiveNotificationsFromListener(mWrapper);
4.     } catch (android.os.RemoteException ex) {
5.         Log.v(TAG, "Unable to contact notification manager", ex);
6.     }
7.     return null;
8. }
```

也就是说只要在onBind方法完成之后，再调用getActiveNotifications()方法就可以正常获取数据了，因此这里使用了handler多线程的方式。当然，为了保险可以使用sendEmptyMessageDelay加上延时。

②. 如果NotificationMonitor在onCreate或onBind方法中crash，则该service已经失效，需重启手机才能进行后续开发验证

如果在onCreate或者onBind方法中，出现异常导致NotificationMonitor发生crash，就算找到问题并将其改正，之后的验证还是无法继续进行的，也就是无法收到通知的新增和删除消息，onNotificationPosted和onNotificationRemoved方法不会被调用。

这也是我在onBind方法中故意注释导致空指针异常的代码，有兴趣的童鞋可以把注释去掉后尝试，去掉注释会导致NotificationListenerDemo异常停止，此时你再加上注释再次运行NotificationListenerDemo，虽然程序可以正常启动，但无法正常执行NotificationMonitor中的onNotificationPosted和onNotificationRemoved方法。这个涉及NotificationListenerService的原理，后面会另行分析。

③. MainActivity中onClick方法里使用handler操作

当点击删除通知时，系统通知相关状态还未更新，此时还没有回调到NotificationMonitor中，所以获取的数据就还是上一次的数据。为了能够获取到正确的Notification数据，可以使用handler并加上延时，这样再去获取Notification信息时，系统已经触发了NotificationMonitor回调，数据也有正常了。另外，50ms的延时几乎是感知不到的。

④. 为什么要使用ArrayList来保存StatusBarNotification数组对象

当新增或者删除通知时，会触发onNotificationPosted或onNotificationRemoved回调，在该方法中调用getActiveNotifications()方法用以获取当前系统通知信息。而getActiveNotifications()返回的是StatusBarNotification[]数组，因为这个数组是可变长的，也就是长度会随时变化，因此无法直接存储。使用ArrayList可以很好的解决这个问题，在ArrayList对象中添加一个StatusBarNotification[]对象，之后使用ArrayList.set(0,statusbar[])方法对数据进行更新即可。

总结

NotificationListenerService是Android 4.3 之后新增的接口服务，用于获取系统Notification信息，这在之前的Android版本是无法直接办到的。在Android 4.4中，增加了Notification.extra变量，使得获取Notification相关信息更加丰富，这些接口的开放更加利于三方应用的使用，但同时也会带来一些隐私问题。

本文针对NotificationListenerService的使用进行了详细分析，当然其中不乏有失偏颇的地方，本着互联网知识共享精神也将自己的一些记录发布出来，一来可做笔记，二来希望能够给苦苦寻觅的童鞋一些帮助。

后续会对NotificationListenerService的原理进行分析，敬请期待。

NotificationMonitor代码免积分下载：[下载Demo](#)

为了后续能够更新，已经代码传到github上，有兴趣的童鞋可以在github上查看，连接戳[这里](#)。

分享到： [新浪微博](#)  [腾讯微博](#) 0赞

原文地址：<http://blog.csdn.net/yihongyuelan/article/details/40977323>

- [« 上一篇](#)
- [下一篇 »](#)

最新热门职位

更多开发者职位上 开源中国·招聘



架构师 中网银科
月薪: 20-30K



测试工程师 轩悦行
月薪: 10-20K



Android研发工程师急急... 艾炽公司
月薪: 10-15K



PHP研发工程师 飞华健康网
月薪: 10-20K

评论0



插入: [表情](#) [开源软件](#)

发表评论

[关闭](#)插入表情
关闭相关文章阅读

- 2015/04/08 [Android系统原理与开发要点详解](#)
- 2013/08/06 [原子操作的实现原理详解](#)
- 2015/05/12 [NotificationManagerService使用详解...](#)
- 2016/01/03 [LVS原理详解及部署之一: ARP原理准...](#)
- 2013/11/20 [Java核心-内存分配原理详解](#)