转 # NotificationManagerService使用详解与原理分析（二）

目录[-]

- 前置文章：
- 概况
- NotificationListenerService启动
- 开机启动

- 广播启动
- 数据库变更启动
- NotificationListenerService启动小结
- NotificationListenerService调用流程
- 新增通知
- 删除通知
- NotificationListenerService调用流程小结
- NotificationListenerService重点分析
- Notification access页面不存在
- getActiveNotifications()方法返回为null
- NotificationListenerService失效
- 总结

前置文章：

《Android 4.4 KitKat NotificationManagerService使用详解与原理分析(一)__使用详解》

概况

在上一篇文章《Android 4.4 KitKat NotificationManagerService使用详解与原理分析(一)__使用详解》中详细介绍了NotificationListenerService的使用方法，以及在使用过程中遇到的问题和规避方案。本文主要分析NotificationListenerService实现原理，以及详细分析在上一篇文章中提到的相关问题和产生的根本原因。在原理分析前，先看看NotificationListenerService涉及到的类以及基本作用，如图1所示：

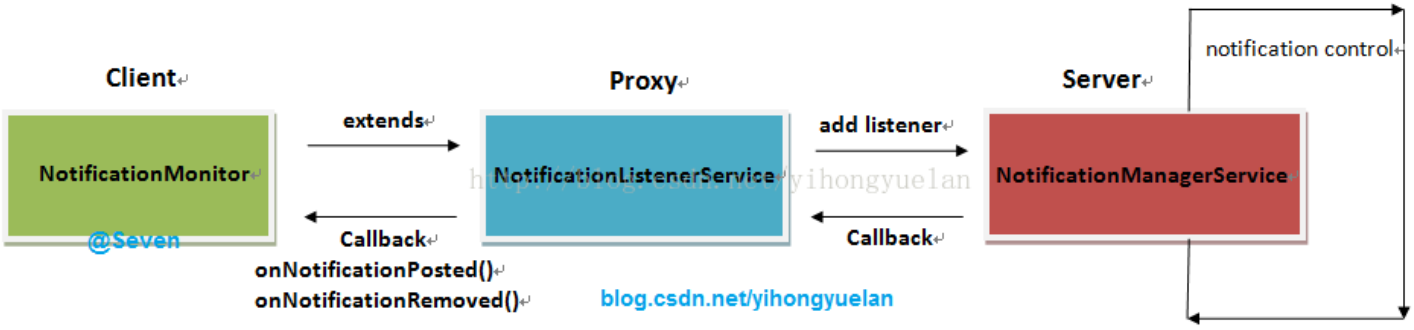

图 1 NLS注册及回调过程

通过图1可以看到，整个通知状态获取分为三部分：

①. 监听器注册；新建一个类NotificationMonitor继承自NotificationListenerService。

②. 系统通知管理；系统通知管理由NotificationManagerService负责。

③. 通知状态回调；当系统通知状态改变之后，NotificationManagerService会通知NotificationListenerService，最后再由NotificationListenerService通知其所有子类。

在整个系统中，通知管理是由NotificationManagerService完成的，NotificationListenerService只是在通知改变时，会获得相应的通知消息，这些消息最终会回调到NotificationListenerService的所有子类中。

NotificationListenerService启动

NotificationListenerService虽然继承自Service，但系统中实际上启动的是其子类，为了表述方便，后文统一使用NotificationListenerService启动来指代。其子类的启动有三个途径，分别是：开机启动、接收PACKAGE相关广播(安装、卸载等)启动、SettingsProvider数据变更启动。

既然NotificationListenerService是一个service，那其子类启动方式自然就是bindService或者startService，在SourceCode/frameworks/base/services/java/com/android/server/NotificationManagerService.java中可以找到，实际上NotificationListenerService的启动是通过bindServiceAsUser来实现的，而bindServiceAsUser与bindService作用一致。

开机启动

因为NotificationListenerService最终是在NotificationManagerService中启动的，因此当系统在开机第一次启动时，会进行

NotificationManagerService初始化，之后会调用其SystemReady方法，继而调用rebindListenerServices以及registerListenerService()，最后使用bindServiceAsUser实现NotificationListenerService的启动。rebindListenerServices代码如下：

[java] view plaincopy

```
1. void rebindListenerServices() {
2.     final int currentUser = ActivityManager.getCurrentUser();
3.     //获取系统中哪些应用开启了Notification access
4.     String flat = Settings.Secure.getStringForUser(
5.             mContext.getContentResolver(),
6.             Settings.Secure.ENABLED_NOTIFICATION_LISTENERS,
7.             currentUser);
8.
9.     NotificationListenerInfo[] toRemove = new NotificationListenerInfo[mListeners.size()];
10.    final ArrayList<ComponentName> toAdd;
11.
12.    synchronized (mNotificationList) {
13.        // unbind and remove all existing listeners
14.        toRemove = mListeners.toArray(toRemove);
15.
16.        toAdd = new ArrayList<ComponentName>();
17.        final HashSet<ComponentName> newEnabled = new HashSet<ComponentName>();
18.        final HashSet<String> newPackages = new HashSet<String>();
19.
20.        // decode the list of components
21.        if (flat != null) {
22.            String[] components = flat.split(ENABLED_NOTIFICATION_LISTENERS_SEPARATOR);
23.            for (int i=0; i<components.length; i++) {
24.                final ComponentName component
25.                        = ComponentName.unflattenFromString(components[i]);
26.                if (component != null) {
27.                    newEnabled.add(component);
28.                    toAdd.add(component);
29.                    newPackages.add(component.getPackageName());
30.                }
31.            }
32.
33.            mEnabledListenersForCurrentUser = newEnabled;
34.            mEnabledListenerPackageNames = newPackages;
35.        }
36.    }
37.
```

```
38.     //对所有NotificationListenerService全部unbindService操作
39.     for (NotificationListenerInfo info : toRemove) {
40.         final ComponentName component = info.component;
41.         final int oldUser = info.userid;
42.         Slog.v(TAG, "disabling notification listener for user " + oldUser + ": " + component);
43.         unregisterListenerService(component, info.userid);
44.     }
45.     //对所有NotificationListenerService进行bindService操作
46.     final int N = toAdd.size();
47.     for (int i=0; i<N; i++) {
48.         final ComponentName component = toAdd.get(i);
49.         Slog.v(TAG, "enabling notification listener for user " + currentUser + ": "
50.                 + component);
51.         registerListenerService(component, currentUser);
52.     }
53. }
```

在该方法中将获取系统中所有NotificationListenerService，并进行registerListenerService操作，代码如下：

[java] view plaincopy

```
1. private void registerListenerService(final ComponentName name, final int userid) {
2. //... ...省略
3.
4. Intent intent = new Intent(NotificationListenerService.SERVICE_INTERFACE);
5. intent.setComponent(name);
6.
7. intent.putExtra(Intent.EXTRA_CLIENT_LABEL,
8.         R.string.notification_listener_binding_label);
9. intent.putExtra(Intent.EXTRA_CLIENT_INTENT, PendingIntent.getActivity(
10.         mContext, 0, new Intent(Settings.ACTION_NOTIFICATION_LISTENER_SETTINGS), 0));
11.
12. try {
13.     if (DBG) Slog.v(TAG, "binding: " + intent);
14.     //使用bindService启动NotificationListenerService
15.     if (!mContext.bindServiceAsUser(intent,
16.             new ServiceConnection() {
17.                 INotificationListener mListener;
18.                 @Override
19.                 public void onServiceConnected(ComponentName name, IBinder service) {
20.                     synchronized (mNotificationList) {
21.                         mServicesBinding.remove(servicesBindingTag);
22.                         try {
```

```
23.                     mListener = INotificationListener.Stub.asInterface(service);
24.                     NotificationListenerInfo info = new NotificationListenerInfo(
25.                             mListener, name, userid, this);
26.                     service.linkToDeath(info, 0);
27.                     //service启动成功之后将相关信息添加到mListeners列表中，后续通过该列表触发回调
28.                     mListeners.add(info);
29.                 } catch (RemoteException e) {
30.                     // already dead
31.                 }
32.             }
33.         }
34.
35.         @Override
36.         public void onServiceDisconnected(ComponentName name) {
37.             Slog.v(TAG, "notification listener connection lost: " + name);
38.         }
39.     },
40.     Context.BIND_AUTO_CREATE,
41.     new UserHandle(userid)))
42.   //... ...省略
43.   }
44. }
```

整个启动流程如图2所示：

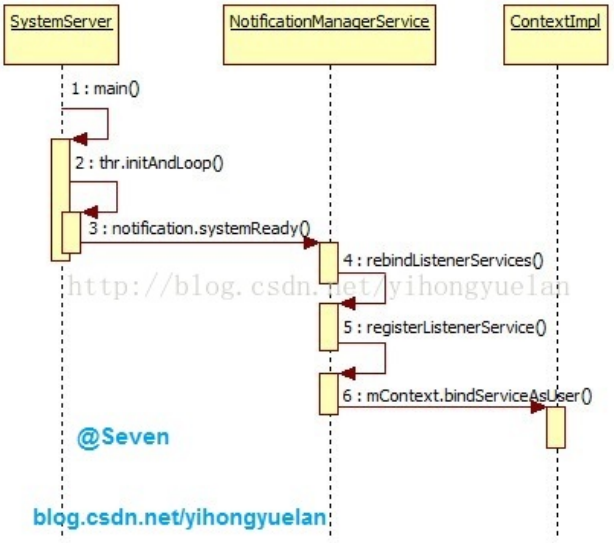

图 2 NLS开机启动时序图

广播启动

当系统安装或者卸载应用的时候，也会触发NotificationListenerService的启动。当一个使用NotificationListenerService的应用被卸载掉后，需要在Notification access界面清除相应的选项，或者当多用户切换时，也会更新NotificationListenerService的状态。在NotificationManagerService中监听了以下广播：

[java] view plaincopy

1. Intent.ACTION_PACKAGE_ADDED
2. Intent.ACTION_PACKAGE_REMOVED
3. Intent.ACTION_PACKAGE_RESTARTED
4. Intent.ACTION_PACKAGE_CHANGED
5. Intent.ACTION_QUERY_PACKAGE_RESTART
6. Intent.ACTION_USER_SWITCHED

这些广播在应用变更时由系统发出，比如安装、卸载、覆盖安装应用等等。当NotificationManagerService接收这些广播后编会调用rebindListenerServices，之后的流程就与前面一样。启动流程如下：

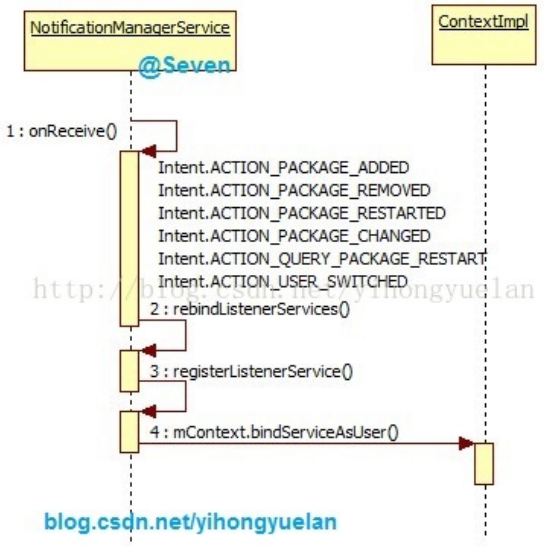

图 3 NLS广播启动

数据库变更启动

在NotificationManagerService中使用了ContentObserver监听SettingsProvider数据库变化，当Notification access有更新时，会更新NotificationListenerService的状态。例如，当用户进入Notification access界面，手动开启或关闭相关应用的Notification access权限时便会触发这种启动方式。当数据库中NotificationListenerService关联的信息改变后，会触发ContentObserver的onChange方法，继而调用update方法更新系统中NotificationListenerService的服务状态，最后调用到rebindListenerServices中。整个流程如下：

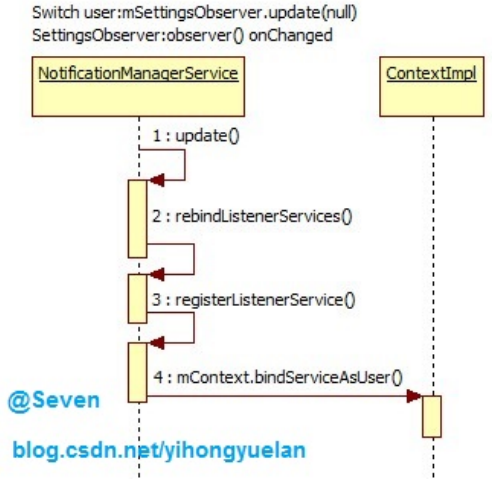

图 4 NLS数据库变更启动

NotificationListenerService启动小结

在系统中实际上运行的是NotificationListenerService的子类，这些子类的启动方式分为三种：开机启动时NotificationManagerService初始化回调；接收相关广播后执行；数据库变更后执行。这些启动方式归根到底还是bindService的操作。

NotificationListenerService调用流程

前面提到了NotificationListenerService的启动流程，当启动完成之后就是调用，整个调用流程分为两种情况，即：新增通知和删除通知。

新增通知

当系统收到新的通知消息时，会调用NotificationManager的notify方法用以发起系统通知，在notify方法中则调用关键方法enqueueNotificationWithTag：

[java] view plaincopy

```
1. service.enqueueNotificationWithTag(......)
```

这里的service是INotificationManager的对象，而NotificationManagerService继承自INotificationManager.Stub。也就是说NotificationManager与NotificationManagerService实际上就是client与server的关系，这里的service最终是NotificationManagerService的对象。这里便会跳转到NotificationManagerService的enqueueNotificationWithTag方法中，实际调用的是enqueueNotificationInternal方法。在该方法中就涉及到Notification的组装，之后调用关键方法notifyPostedLocked()：

[java] view plaincopy

```
1. private void notifyPostedLocked(NotificationRecord n) {
2.     final StatusBarNotification sbn = n.sbn.clone();
3.     //这里触发mListeners中所有的NotificationListenerInfo回调
4.     for (final NotificationListenerInfo info : mListeners) {
5.         mHandler.post(new Runnable() {
6.             @Override
7.             public void run() {
8.                 info.notifyPostedIfUserMatch(sbn);
9.             }});
10.    }
11. }
```

到这里就开始准备回调了，因为前面通知已经组装完毕准备显示到状态栏了，之后就需要将相关的通知消息告诉所有监听者。继续看到notifyPostedIfUserMatch方法：

[java] view plaincopy

```
1. public void notifyPostedIfUserMatch(StatusBarNotification sbn) {
2.     //... ...省略
3.     try {
4.         listener.onNotificationPosted(sbn);
5.     } catch (RemoteException ex) {
6.         Log.e(TAG, "unable to notify listener (posted): " + listener, ex);
7.     }
8. }
```

上面的listener对象是NotificationListenerInfo类的全局变量，那是在哪里赋值的呢？还记得前面注册NotificationListenerService的时候bindServiceAsUser，其中new了一个ServiceConnection对象，并在其onServiceConnected方法中有如下代码：

[java] view plaincopy

```
1. public void onServiceConnected(ComponentName name, IBinder service) {
2. synchronized (mNotificationList) {
3.     mServicesBinding.remove(servicesBindingTag);
4.     try {
5.         //mListener就是NotificationListenerService子类的对象
6.         //service是INotificationListenerWrapper的对象，INotificationListenerWrapper
7.         //继承自INotificationListener.Stub，是NotificationListenerService的内部类
8.         mListener = INotificationListener.Stub.asInterface(service);
9.         //使用mListener对象生成对应的NotificationListenerInfo对象
10.        NotificationListenerInfo info = new NotificationListenerInfo(
11.                mListener, name, userid, this);
12.        service.linkToDeath(info, 0);
13.        mListeners.add(info);
14.    } catch (RemoteException e) {
15.        // already dead
16.    }
17. }
18. }
```

也就是说在NotificationListenerService启动并连接的时候，将binder对象保存到了NotificationListenerInfo中。这里就得看看NotificationListenerService的onBind方法返回了，代码如下：

[java] view plaincopy

```
1. @Override
2. public IBinder onBind(Intent intent) {
3.     if (mWrapper == null) {
4.         mWrapper = new INotificationListenerWrapper();
5.     }
6.     //这里返回的是INotificationListenerWrapper对象
7.     return mWrapper;
8. }
9.
10. private class INotificationListenerWrapper extends INotificationListener.Stub {
11.     @Override
12.     public void onNotificationPosted(StatusBarNotification sbn) {
13.         try {
14.             //onNotificationPosted是抽象方法之一
15.             NotificationListenerService.this.onNotificationPosted(sbn);
16.         } catch (Throwable t) {
17.             Log.w(TAG, "Error running onNotificationPosted", t);
18.         }
19.     }
```

```
20.     @Override
21.     public void onNotificationRemoved(StatusBarNotification sbn) {
22.         try {
23.             //onNotificationRemoved是另一个抽象方法
24.             NotificationListenerService.this.onNotificationRemoved(sbn);
25.         } catch (Throwable t) {
26.             Log.w(TAG, "Error running onNotificationRemoved", t);
27.         }
28.     }
29. }
```

通过以上代码可以知道，当在notifyPostedIfUserMatch执行listener.onNotificationPosted方法时，实际上会调用到NotificationListenerService.INotificationListenerWrapper的onNotificationPosted方法。

NotificationListenerService是一个Abstract类，其中的Abstract方法是onNotificationPosted和onNotificationRemoved。当触发NotificationListenerService.INotificationListenerWrapper的onNotificationPosted方法时，继续调用了NotificationListenerService.this.onNotificationPosted(sbn)。这样会继续调用所有NotificationListenerService子类中的onNotificationPosted方法，系统通知新增的消息便传到了所有NotificationListenerService中。

从整个流程来看，新增通知的发起点是NotificationManager，处理通知则是由NotificationManagerService完成，传输过程是通过NotificationListenerService，最后回调方法是各个继承自NotificationListenerService的子类。整个过程的调用时序图如下：

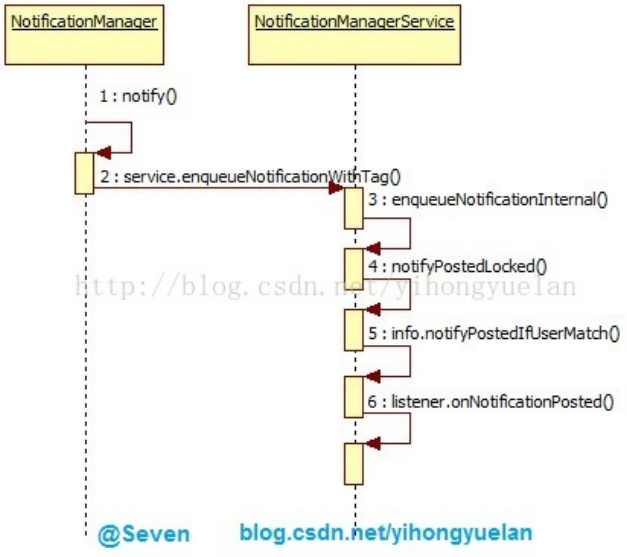

图 5 onNotificationPosted触发流程

删除通知

与"新增通知"类似的流程是"删除通知"，发起点在NotificationManager，之后经由NotificationManagerService处理和NotificationListenerService传递，最后到达各个继承自NotificationListenerService的子类中，只不过最后的处理方法变成了onNotificationRemoved。调用时序图下：

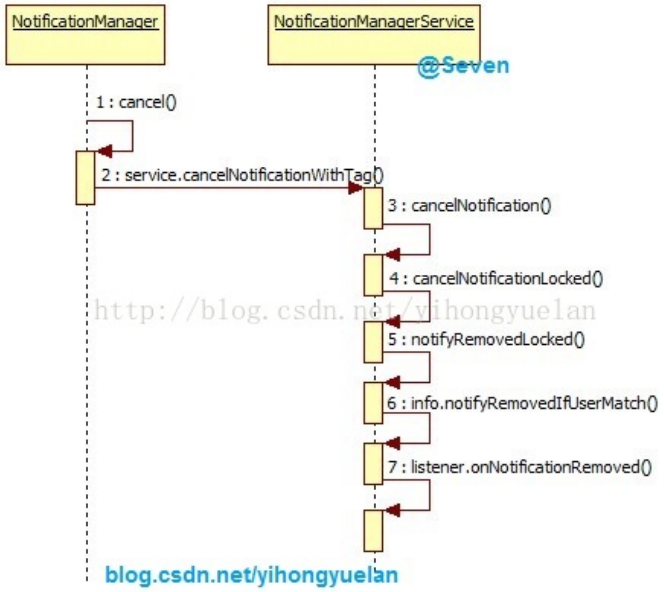

图 6 onNotificationRemoved触发流程

NotificationListenerService调用流程小结

简单来看，NotificationListenerService在系统通知的消息传递过程中，起到了代理的作用。继承自NotificationListenerService的类作为client端，真正的server端则是NotificationManagerService，由它负责整个Notification的控制与管理。NotificationManagerService将处理之后的结果通过NotificationListenerService返回给client端，最终各个client端通过onNotificationPosted和onNotificationRemoved方法拿到系统通知状态变更的相关信息。

NotificationListenerService重点分析

前文分析了整个NotificationListenerService的启动和调用，通过以上分析可以很清楚的了解NotificationListenerService的工作流程。在上一篇文章《Android 4.4 KitKat NotificationManagerService使用详解与原理分析(一)__使用详解》中，文末分析了在NotificationListenerService在使用过程中的遇到的一些问题，但并没有深究出现这些问题的根本原因，下文会对这些问题进行详细分析。

Notification access页面不存在

当手机上没有安装任何使用NotificationListenerService的应用时，系统默认不会显示"Notification access"选项。只有手机中安装了使用NotificationListenerService的应用，才可以在"Settings > Security > Notification access" 找到对应的设置页面。在SourceCode/packages/apps/Settings/src/com/android/settings/SecuritySettings.java中可以看到如下初始化代码：

[java] view plaincopy

```
1. //... ...省略
2. mNotificationAccess = findPreference(KEY_NOTIFICATION_ACCESS);
3. if (mNotificationAccess != null) {
4.     final int total = NotificationAccessSettings.getListenersCount(mPM);
5.     if (total == 0) {
6.         if (deviceAdminCategory != null) {
7.             //如果系统中没有安装使用NLS的应用则删除显示
8.             deviceAdminCategory.removePreference(mNotificationAccess);
9.         }
10.    } else {
11.        //获取系统中有多少启动了Notification access的应用
```

```
12.         final int n = getNumEnabledNotificationListeners();
13.         //根据启用的数量显示不同的Summary
14.         if (n == 0) {
15.             mNotificationAccess.setSummary(getResources().getString(
16.                     R.string.manage_notification_access_summary_zero));
17.         } else {
18.             mNotificationAccess.setSummary(String.format(getResources().getQuantityString(
19.                     R.plurals.manage_notification_access_summary_nonzero,
20.                     n, n)));
21.         }
22.     }
23. }
24. //... ...省略
```

getActiveNotifications()方法返回为null

有很多人在使用getActiveNotifications方法时返回为null，多数情况下是因为在onCreate或者在onBind中调用了getActiveNotifications方法。比如NotificaionMonitor extends NotificationListenerService：

[java] view plaincopy

```
1. public class NotificationMonitor extends NotificationListenerService {
2. @Override
3. public void onCreate() {
4.     //getActiveNotifications();
5.     super.onCreate();
6. }
7.
8. @Override
9. public IBinder onBind(Intent intent) {
10.     getActiveNotifications();
11.     return super.onBind(intent);
12. }
13.
14. }
```

找到NotificationListenerService中的getActiveNotifications方法实现，代码如下：

[java] view plaincopy

```
1. public StatusBarNotification[] getActiveNotifications() {
2.     try {
3.         //getActiveNotifications成功执行的两个关键点：
4.         //1.getNotificationInterface方法返回正常
5.         //2.mWrapper对象不为null
6.         return getNotificationInterface().getActiveNotificationsFromListener(mWrapper);
7.     } catch (android.os.RemoteException ex) {
```

```
8.          Log.v(TAG, "Unable to contact notification manager", ex);
9.      }
10.     return null;
11. }
12.
13. //通过查看可以知道，getNotificationInterface没有问题，如果为null会进行初始化
14. private final INotificationManager getNotificationInterface() {
15.     if (mNoMan == null) {
16.         mNoMan = INotificationManager.Stub.asInterface(
17.                 ServiceManager.getService(Context.NOTIFICATION_SERVICE));
18.     }
19.     return mNoMan;
20. }
21.
22. //如果mWrapper为null则进行初始化
23. @Override
24. public IBinder onBind(Intent intent) {
25.     if (mWrapper == null) {
26.         mWrapper = new INotificationListenerWrapper();
27.     }
28.     return mWrapper;
29. }
```

通过上面的代码可以知道getActiveNotifications方法调用失败的原因：

1. service的生命周期会先从**onCraete->onBind**逐步执行；

2. 此时调用**getActiveNotifications**方法会使用**NotificationListenerService**中的**mWrapper**对象；

3. mWrapper对象必须在**NotificationMonitor**完成**super.onBind**方法之后才会初始化；

综上所述，当在onCreate或者onBind方法中使用getActiveNotifications方法时，会导致mWrapper没有初始化，即mWrapper == null。解决方案可以在onCreate或者onBind方法中使用handler异步调用getActiveNotification方法，具体可参考《Android 4.4 KitKat NotificationManagerService使用详解与原理分析(一)__使用详解》。

NotificationListenerService失效

如果NotificationMonitor在onCreate或onBind方法中出现crash，则该NotificationMonitor已经失效。就算修改了NotificationMonitor的代码不会再crash，但NotificationMonitor还是不能收到onNotificationPosted和onNotificationRemoved回调，除非重启手机。

这个问题是google设计上的缺陷导致，出现NotificationListenerService失效的必要条件： 在NotificationMonitor的onCreate或者onBind中出现异常，导致service crash，也就是说service还没有完全启动的情况下出现了异常导致退出。

这里需要回到NotificationManagerService中，NotificationListenerService的注册方法registerListenerService中：

[java] view plaincopy

```
1. private void registerListenerService(final ComponentName name, final int userid) {
2.     //servicesBindingTag可以理解为需要启动的service的标签
3.     final String servicesBindingTag = name.toString() + "/" + userid;
```

```
4.  //如果mServicesBinding中已经包含正在处理的service则直接return退出
5.  if (mServicesBinding.contains(servicesBindingTag)) {
6.  // stop registering this thing already! we're working on it
7.      return;
8.  }
9.  //将准备启动的service标签添加到mServicesBinding中
10. mServicesBinding.add(servicesBindingTag);
11.
12. //... ...省略
13. //使用bindServiceAsUser启动service
14. Intent intent = new Intent(NotificationListenerService.SERVICE_INTERFACE);
15. intent.setComponent(name);
16.
17. intent.putExtra(Intent.EXTRA_CLIENT_LABEL,
18.         R.string.notification_listener_binding_label);
19. intent.putExtra(Intent.EXTRA_CLIENT_INTENT, PendingIntent.getActivity(
20.         mContext, 0, new Intent(Settings.ACTION_NOTIFICATION_LISTENER_SETTINGS), 0));
21.
22. try {
23.     if (DBG) Slog.v(TAG, "binding: " + intent);
24.     if (!mContext.bindServiceAsUser(intent,
25.             new ServiceConnection() {
26.                 INotificationListener mListener;
27.                 @Override
28.                 public void onServiceConnected(ComponentName name, IBinder service) {
29.                     synchronized (mNotificationList) {
30.                         //服务成功启动之后删除标签
31.                         mServicesBinding.remove(servicesBindingTag);
32.                         try {
33.                             mListener = INotificationListener.Stub.asInterface(service);
34.                             NotificationListenerInfo info = new NotificationListenerInfo(
35.                                     mListener, name, userid, this);
36.                             service.linkToDeath(info, 0);
37.                             mListeners.add(info);
38.                         } catch (RemoteException e) {
39.                             // already dead
40.                         }
41.                     }
42.                 }
43.
```

```
44.                 @Override
45.                 public void onServiceDisconnected(ComponentName name) {
46.                     Slog.v(TAG, "notification listener connection lost: " + name);
47.                 }
48.             },
49.             Context.BIND_AUTO_CREATE,
50.             new UserHandle(userid)))
51.         {
52.             //绑定服务失败后删除标签
53.             mServicesBinding.remove(servicesBindingTag);
54.             Slog.w(TAG, "Unable to bind listener service: " + intent);
55.             return;
56.         }
57.     } catch (SecurityException ex) {
58.         Slog.e(TAG, "Unable to bind listener service: " + intent, ex);
59.         return;
60.     }
61. }
62. }
```

当调用registerListenerService方法时，使用了一个mServicesBinding的ArrayList<String>用来记录当前正在启动的服务。在启动之前会判断当前service是否在mServicesBinding之中，如果是则表明正在执行bindServiceAsUser操作，直接退出，否则就继续执行bindServiceAsUser流程。调用bindServiceAsUser之前会在mServicesBinding中会添加标签，当连接成功之后也就是onServiceConnected返回后，以及绑定失败后会在mServicesBinding中删除标签。

google这样设计的目的可能是为了避免同一个service多次启动，因此在执行bindServiceAsUser之前就打上标签，当处理完成之后(onServiceConnected回调)就删掉这个标签，表明这个service 绑定完成。但是，如果执行bindServiceAsUser之后，NotificationMonitor在onCreate或者onBind的时候crash了，也就是NotificationMonitor还没有完成启动，因此就不会去调用onServiceConnected方法，并最终导致不会调用mServicesBinding.remove(servicesBindingTag)方法，从而使得NotificationMonitor的标签被一致记录在mServicesBinding中。那么当下一次想再次注册该服务的时候，系统发现该服务已经在mServicesBinding中了，所以直接return，后面的bindServiceAsUser就不会被调用了。

虽然代码已经更新，但service无法正常启动，那么onNotificationPosted和onNotificationRemoved的回调自然就无法使用，此时的解决办法就只能重启手机，清空mServicesBinding的值。

总结

NotificationListenerService在系统通知获取的流程中，自身并没有启动，而是起到了一个代理的作用，每一个继承自NotificationListenerService的类，当系统通知变化后最终都会收到onNotificationPosted和onNotificationRemoved的回调。

bindService方法的返回与service是否成功启动无关，因此才会导致NotificationListenerService失效。

最后再看一下整个NotificationListenerService的关系类图：

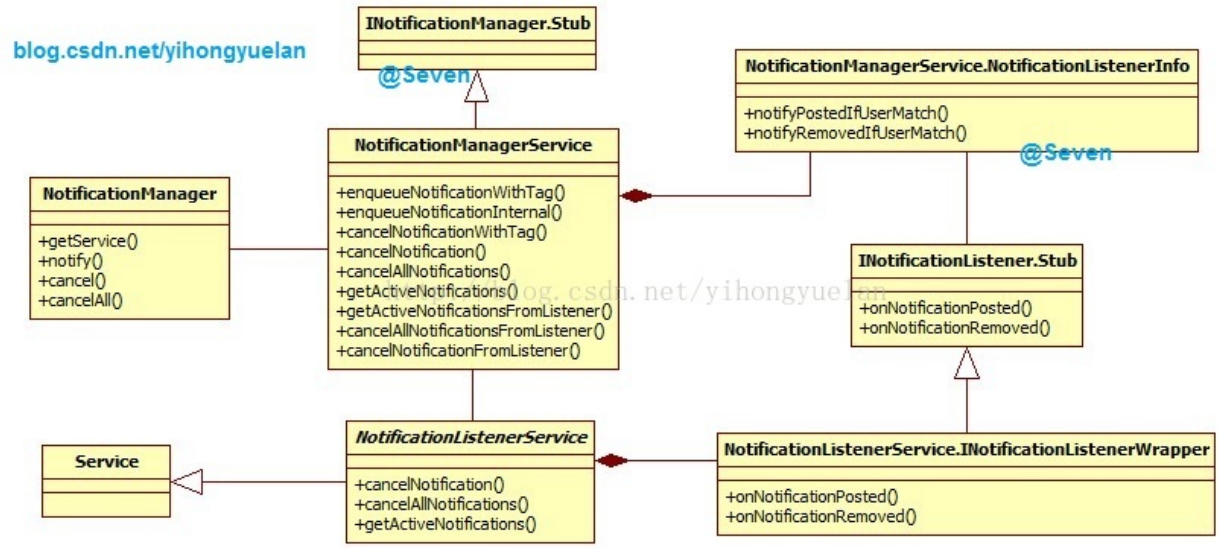

图 7 NLS关系类图

示例代码：

https://github.com/yihongyuelan/NotificationListenerServiceDemo

分享到： 新浪微博 weibo.com 腾讯微博 t.qq.com 0赞
原文地址：http://blog.csdn.net/yihongyuelan/article/details/41084165

- « 上一篇
- 下一篇 »

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