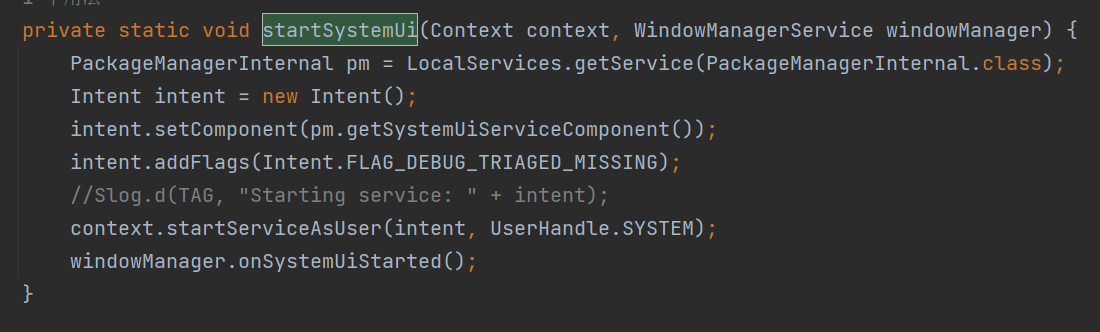
SystemUI介绍

SystemUI其实是一个APK，在Android系统中SystemUI是以应用的形式运行在Android系统当中，即编译SystemUI模块会生产APK文件，源代码路径在frameworks/base/packages/SystemUI/，安装路径system/priv-app/-SystemUI。

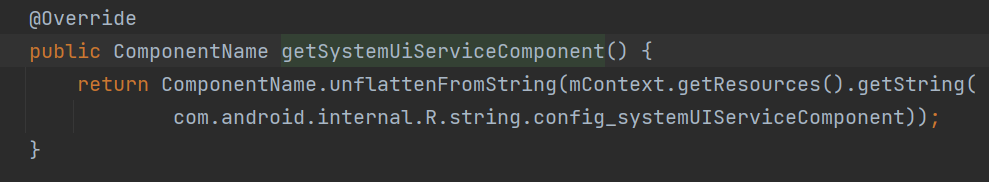
1. SystemUI如何启动
2. SystemUI顾名思义就是全局UI，必须在开机过程中完成启动，并不可退出。
3. 我们在frameworks/base/services/java/com/android/server/SystemServer.java中找到方法

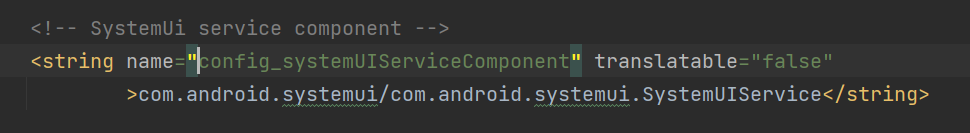
startSystemUi()



观察发现我们通过创建Intent对象实现.setCompoment()方法

我们再去寻找PackageManagerInternal中的getSystemUiServiceComponent

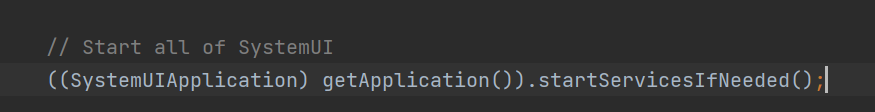




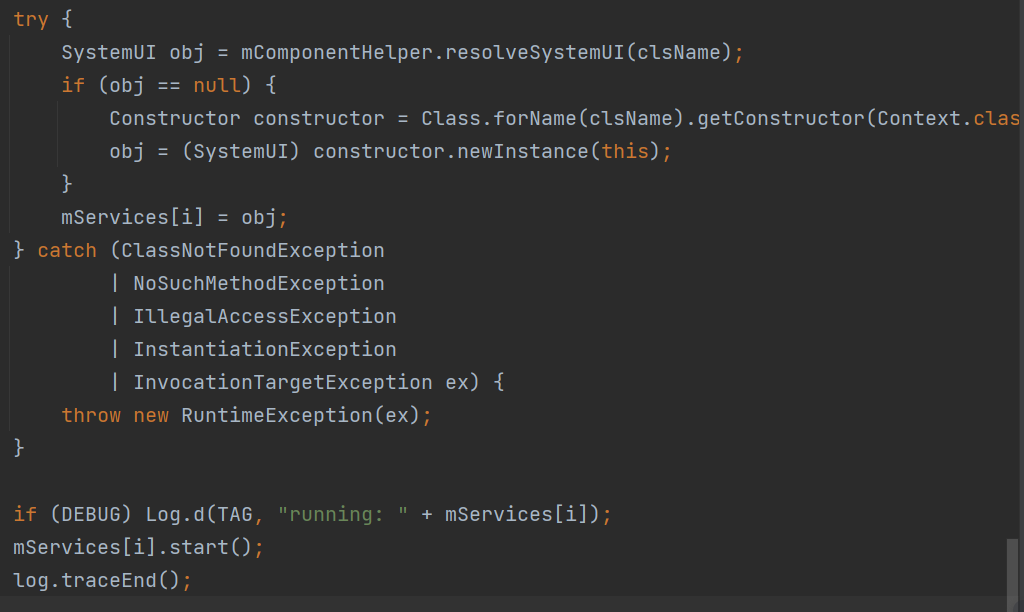
观察发现还执行了一次startServiceAsUser方法这个方法允许你在特定的用户上下文中启动一个服务，而不是默认的应用程序用户。这在一些特定情况下很有用，比如当你需要在后台执行某些任务，但又不希望在用户界面上表现出来时可以使用这个方法

1. 启动SystemUIService服务

我们主要观察onCreate()方法即可

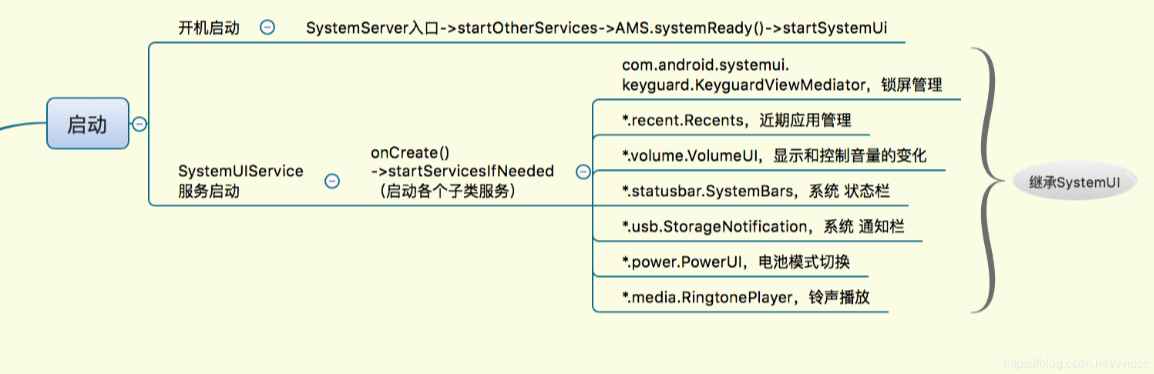


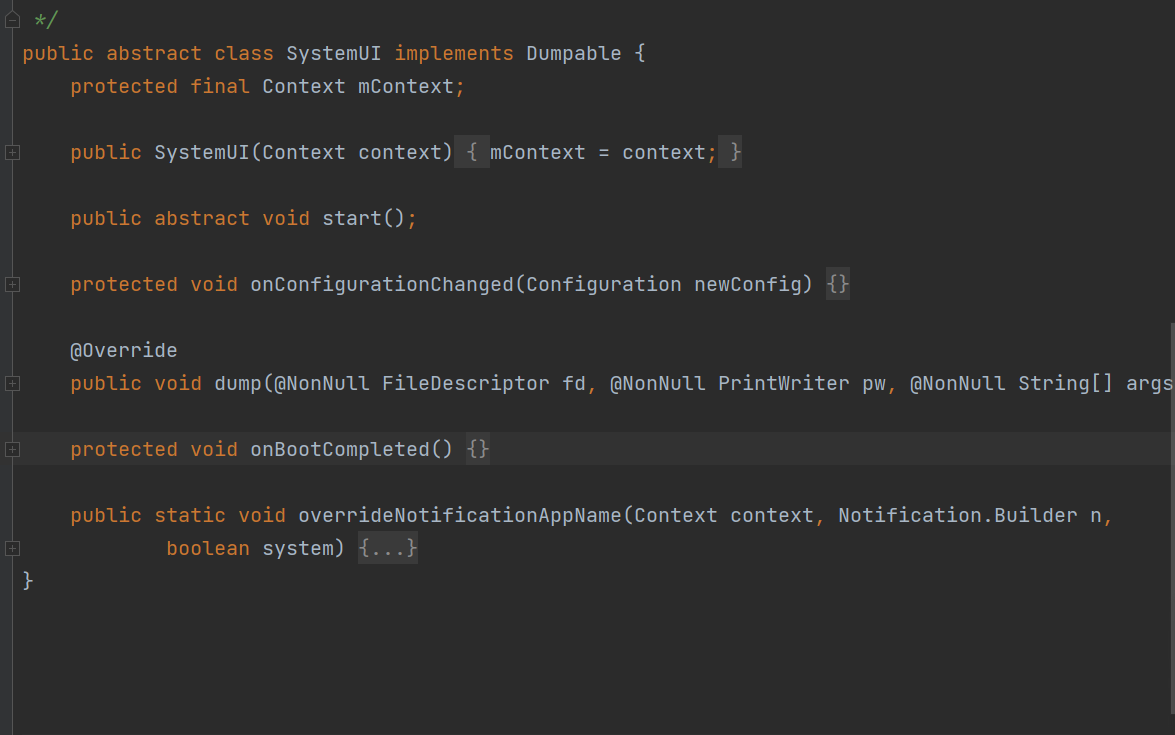
观察startServicesIfNeeded()方法



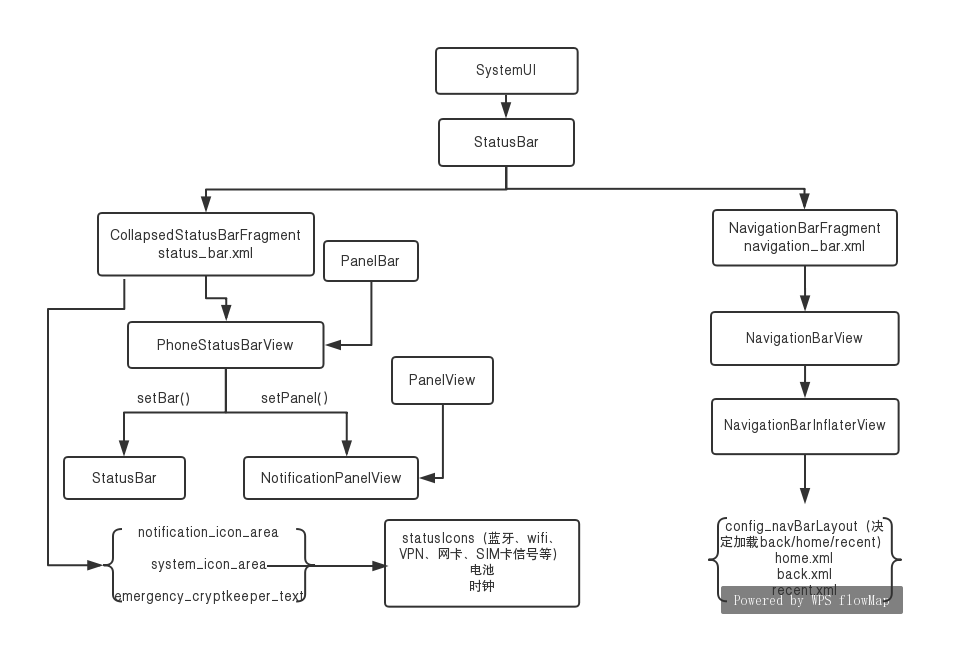
为什么这个服务是个数组呢？

因为SystemUi启动的服务不止一个，包括这些service



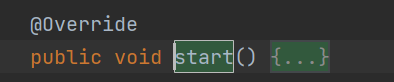


SystemUI中的服务都会实现以下方法

1. 状态栏的创建
2. 

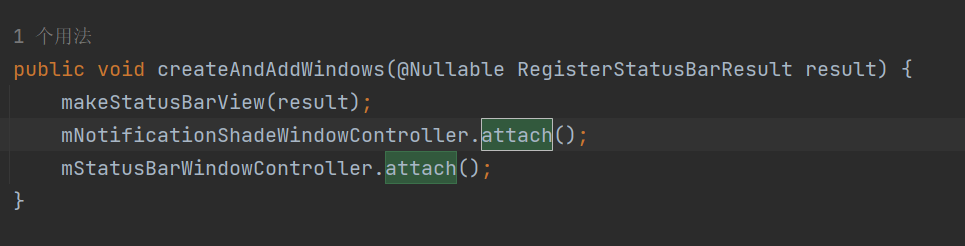
服务创建后,我们需要让用户看到效果，状态栏就是实现的途径

1. 获取各种服务和实例



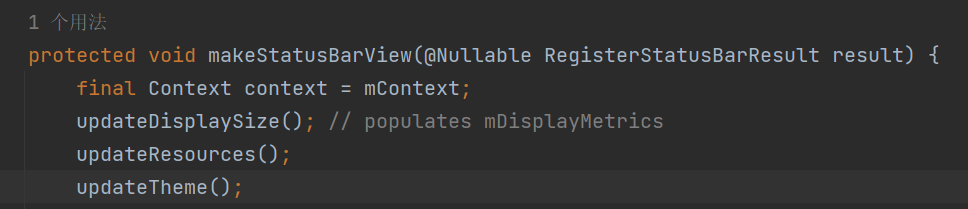
start方法中主要是有关notification的服务类获取

1. 创建状态栏和导航栏



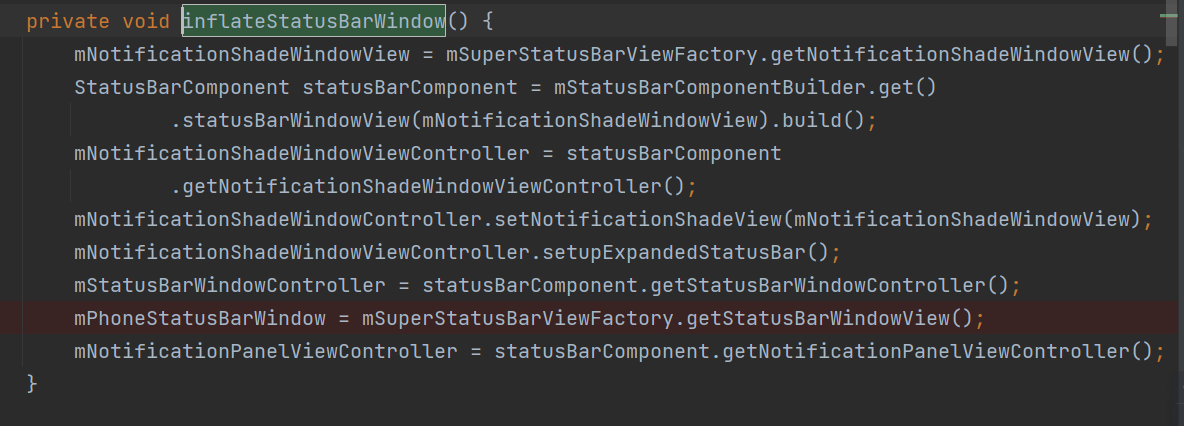
注意下面三个方法

updateDisplaySize()方法,updateResources()方法,updateTheme();方法



初始化导航栏高度，状态栏高度，通知面板位置和高度等

再看inflateStatusBarWindow方法

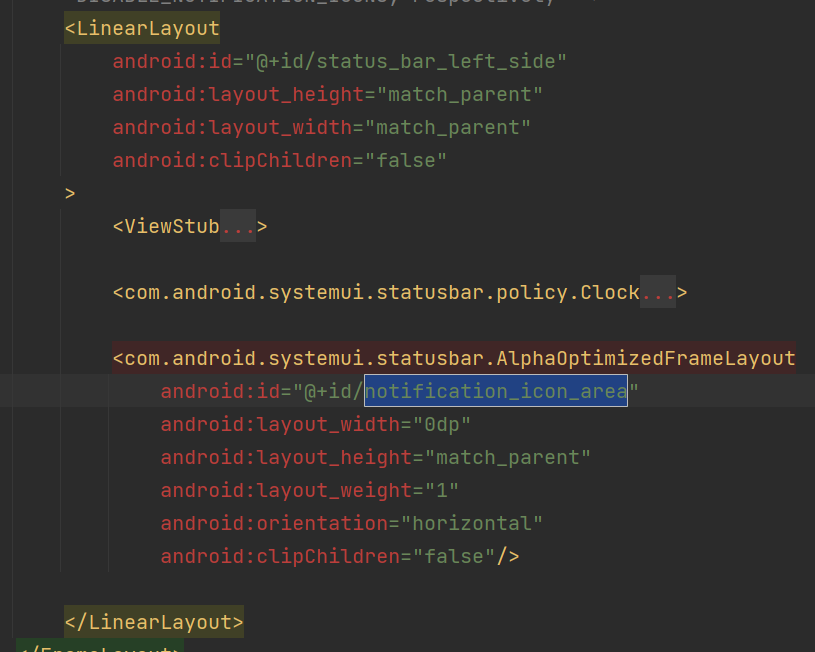


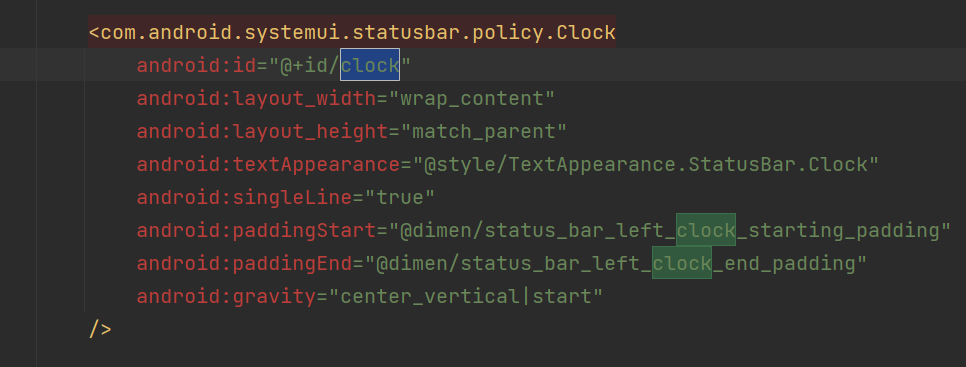
inflateStatusBarWindow()实例化整个statusBar视图，包括状态栏，通知面版，锁屏

需要将状态栏的视图添加，回到createAndAddWindows方法

1. 状态栏的布局

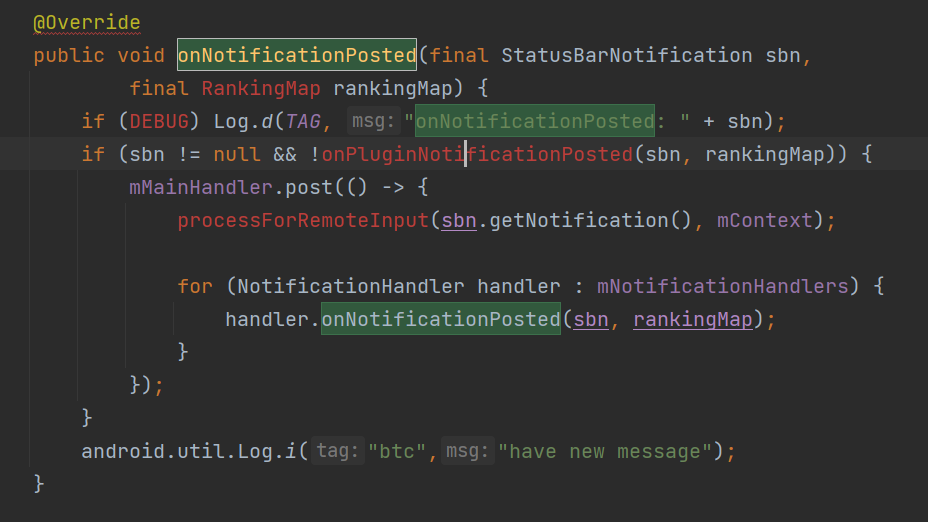
状态栏的布局文件是status\_bar.xml





@+id/status\_bar\_left\_side ，从名字来看就知道是状态栏左边部分。这个状态栏左边部分包含了时钟@+id/clock和短信通知@+id/notification\_icon\_area

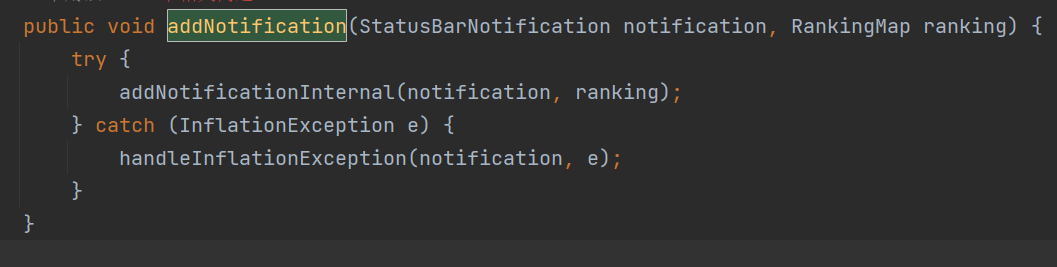
1. 状态栏的消息通知
2. NotificationListener.java类主要是监听系统通知的发送、移除、更新的回调



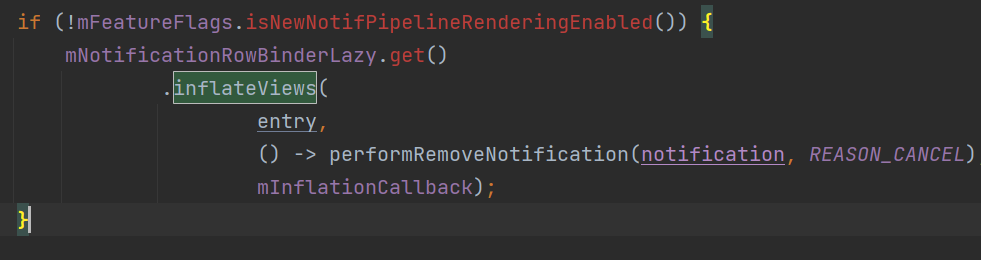
收到通知到，就通过 NotificationEntryManager来进行操作

我们观察一下NotificationEntryManager.java类

主要关注两个方法addNotification和addNotificationInternal方法

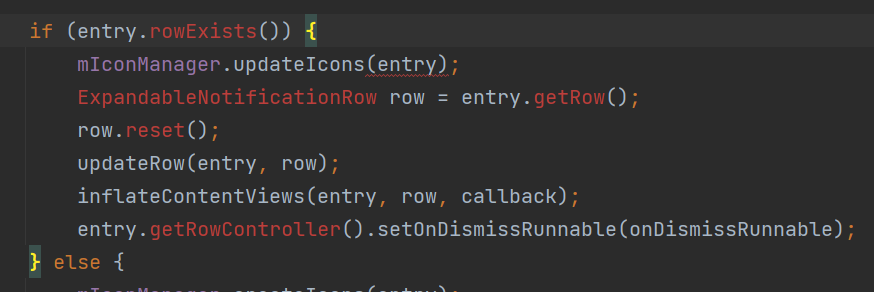


异步创建控件

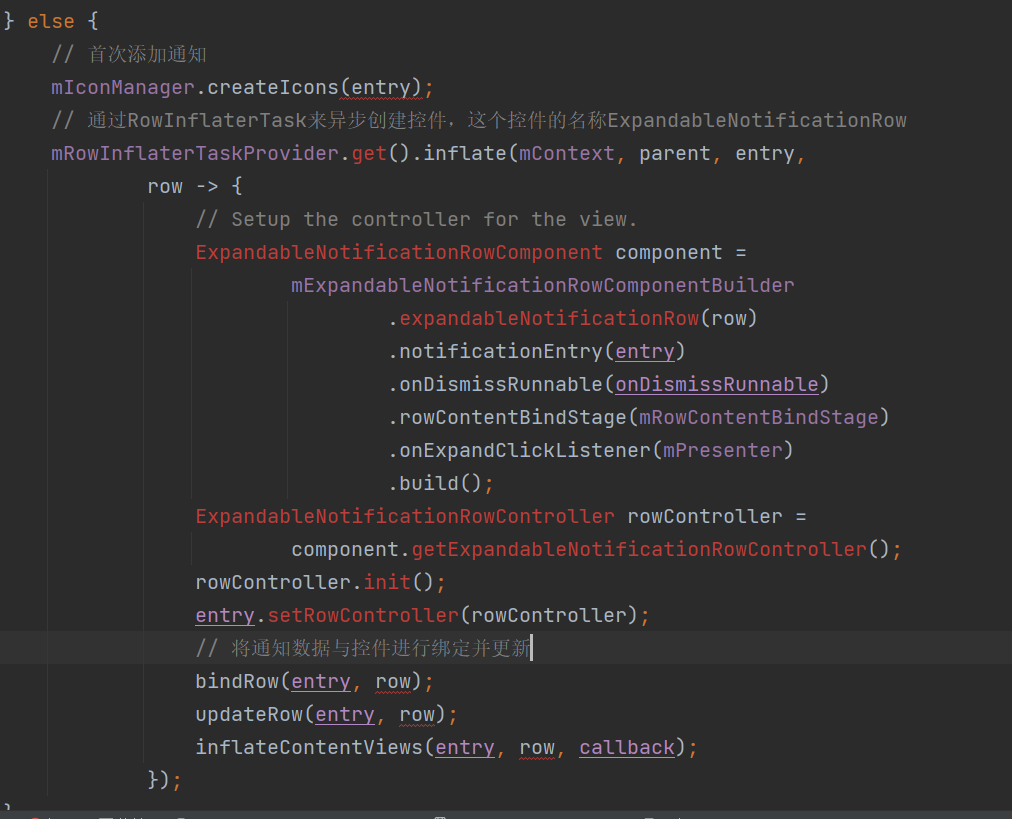


我们再观察inflateViews方法

（1）如果本身通知就存在,再有新的通知就会更新操作

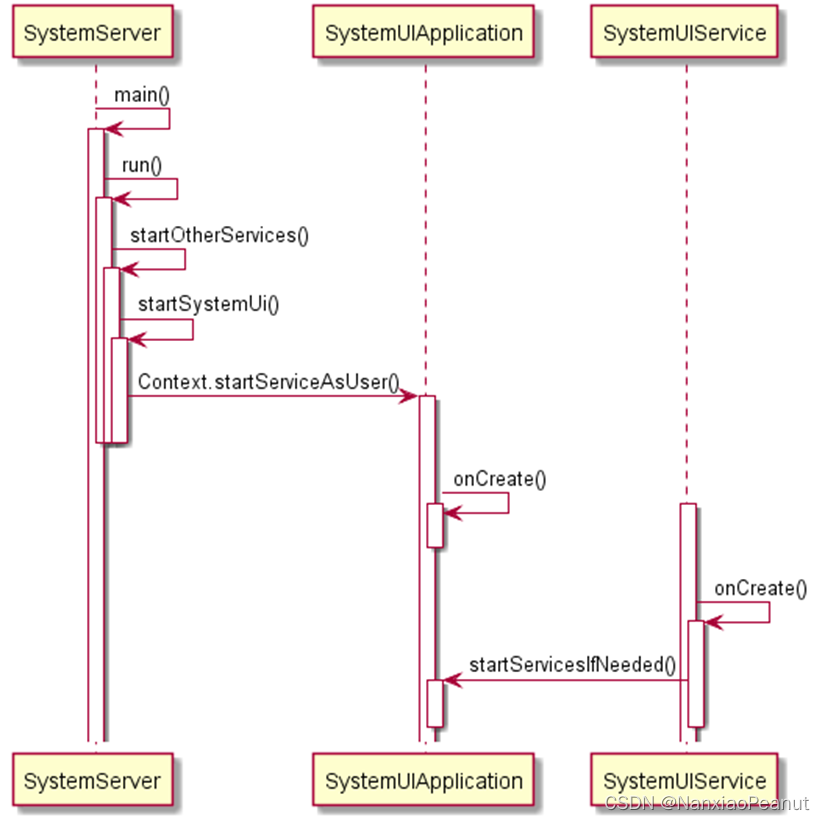


1. 如果通知是第一次添加



在 inflateViews()方法中会根据当前的通知是否存在进行更新或者添加，首次添加通知时会使用一个异步构建控件的接口来创建 ExpandableNotificationRowComponent实例，这个就是在通知栏中显示的具体控件。

SystemUI的启动流程：



SystemServer启动了 SystemUIService，然后通过反射创建在 xml中配置的 SystemUI服务组件列表，这其中就包括了 StatusBar组件，这是系统状态栏。

这个状态栏会与系统的状态栏服务 StatusBarService建立 Binder通信。同时还会使用 NotificatioinListener注册系统通知的监听。

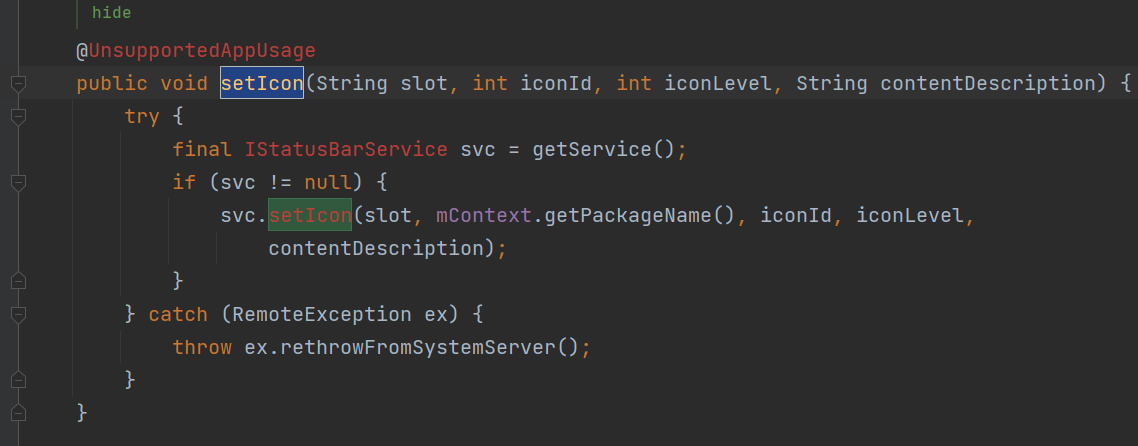
当收到通知后相应的回调接口 onNotificationPosted会被执行，然后使用 NotificationEntryManager对相应的通知进行处理。

1. 状态栏的图标

注意:信号图标和电量和系统时钟是单独的类控制分别是SignalCluster和BatteryController和Clock其他的图标主要是config文件控制

系统显示图标的流程:

1.进入StatusBarManager类找到setIcon方法



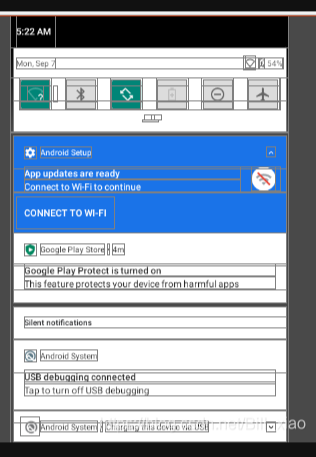
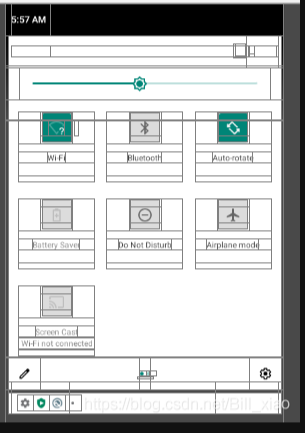
状态栏图标的管理



监听广播并在状态栏进行更新

1. 下拉通知栏

我们观察下拉的通知栏有两种布局

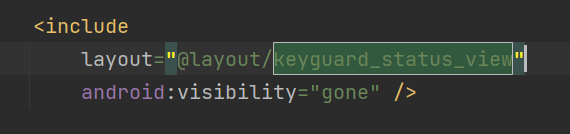


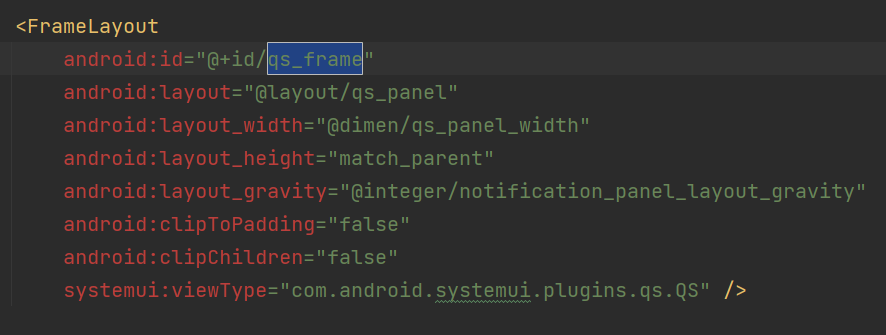
我们只需要观察一下几个

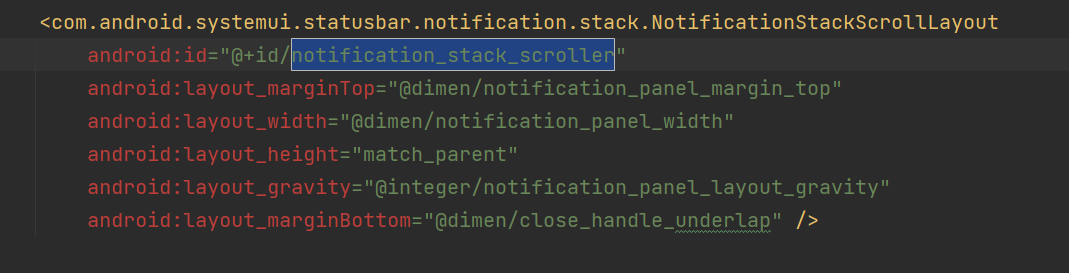
1.@layout/keyguard\_status\_view 这个是锁屏界面的View

2.@+id/qs\_frame QS快捷面板

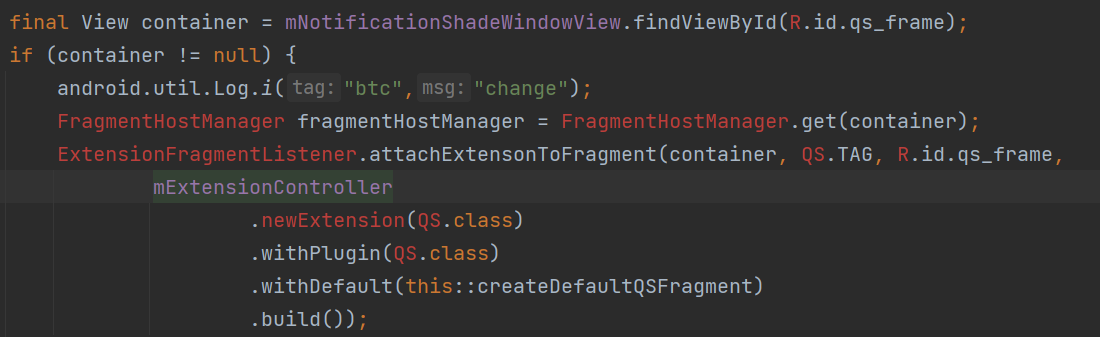
3.@+id/notification\_stack\_scroller短信通知栏



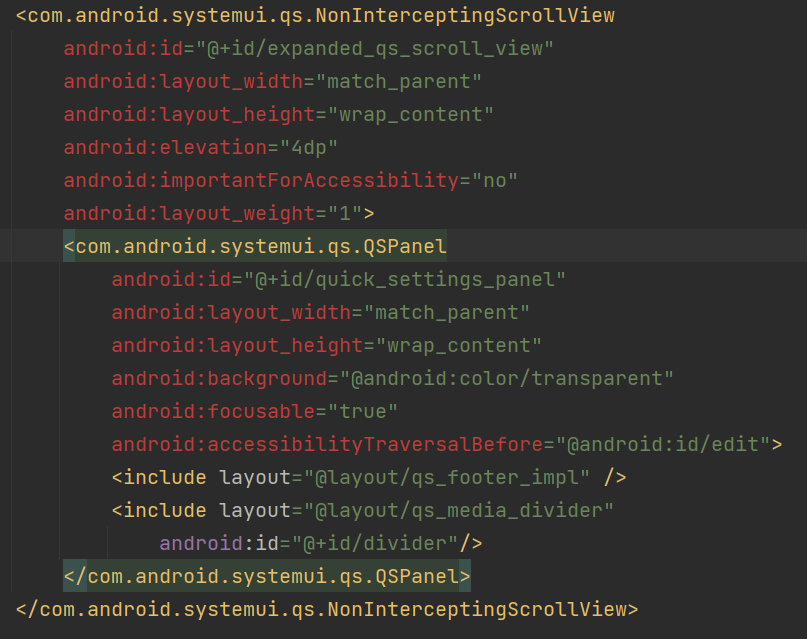




创建下拉菜单在开启初始化的时候就已经创建了



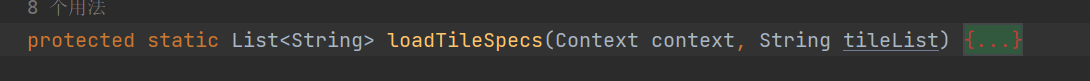
再去qs\_pane.xml文件中查看



@+id/quick\_settings\_panel对应下拉完全的布局

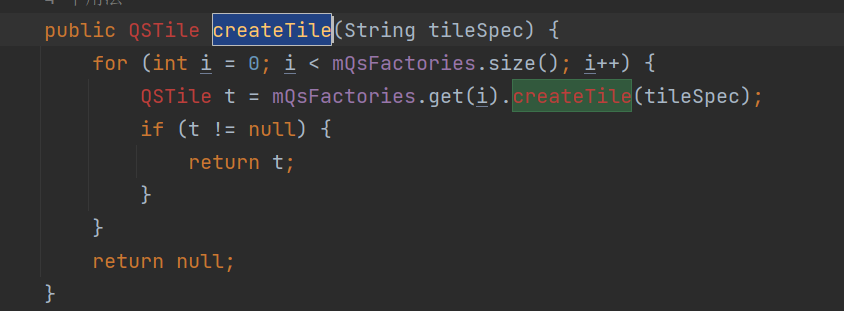
@layout/quick\_status\_bar\_expanded\_header对应下拉未完全的布局

如何加载QS面板的数据在QSTileHost.java类中通过方法 loadTileSpecs 方法将 config.xml 中定义的标签内容 load 进来



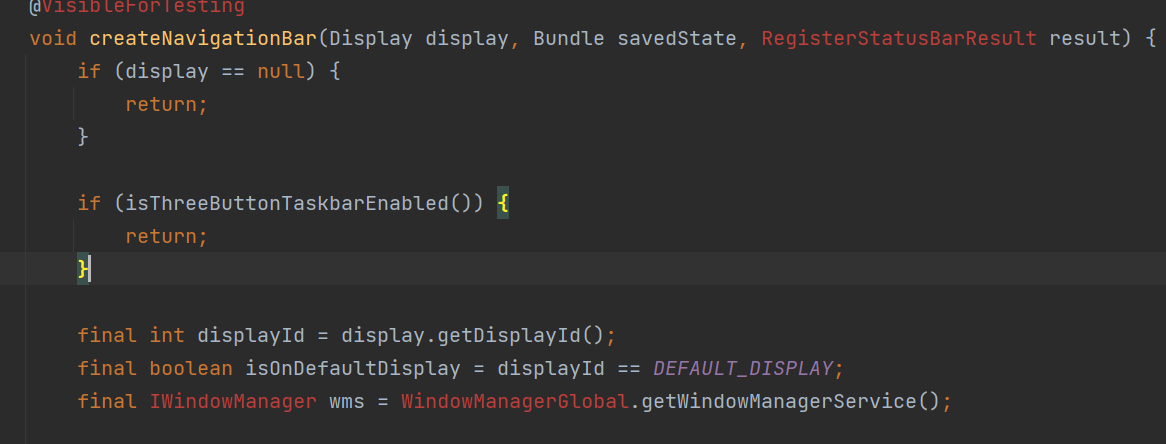
这个方法是用来加载config中的标签内容

再观察createTile方法



这个方法是将 Tile 保存在成员变量的 mTiles 集合中，最后回调 onTilesChanged ()方法，通知 QSPanel.java 对快捷设置选项显示更新

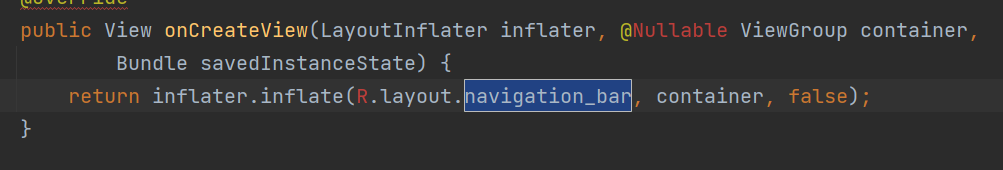
1. 导航栏
2. 创建导航栏



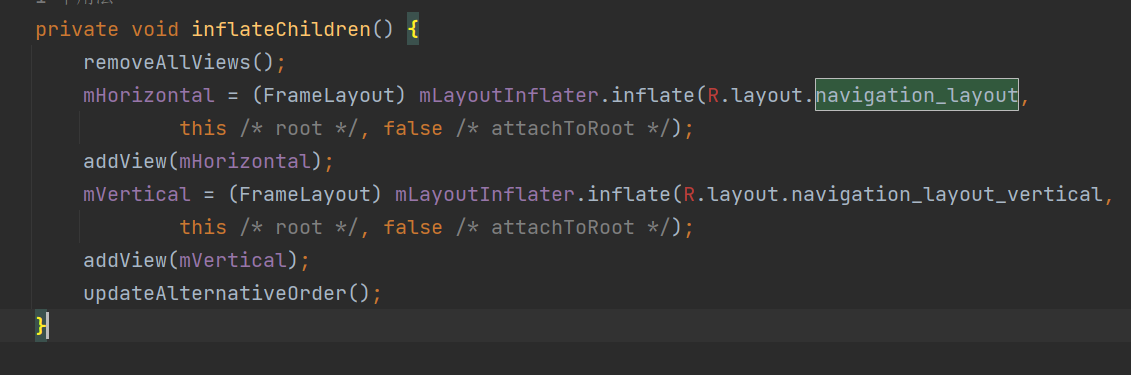
createNavigationBar方法用来创建导航栏

1. 导航栏的布局

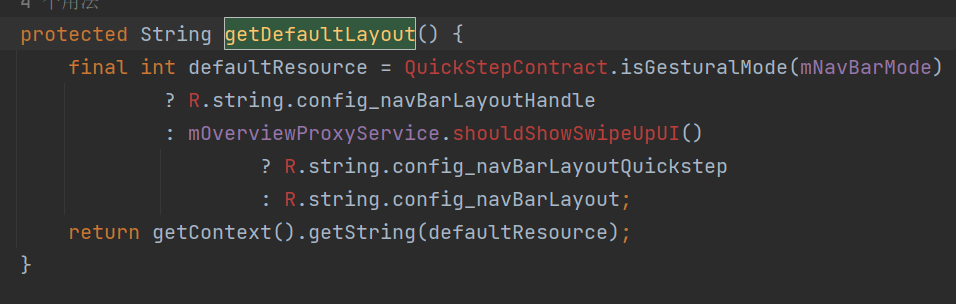
标准布局



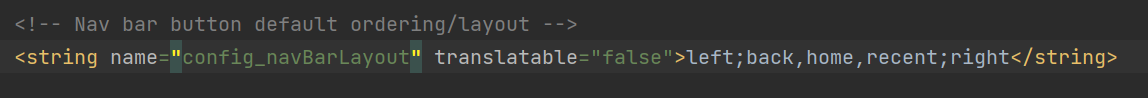
导航栏在0度和90度时候有两种不同的布局



导航栏的控件显示

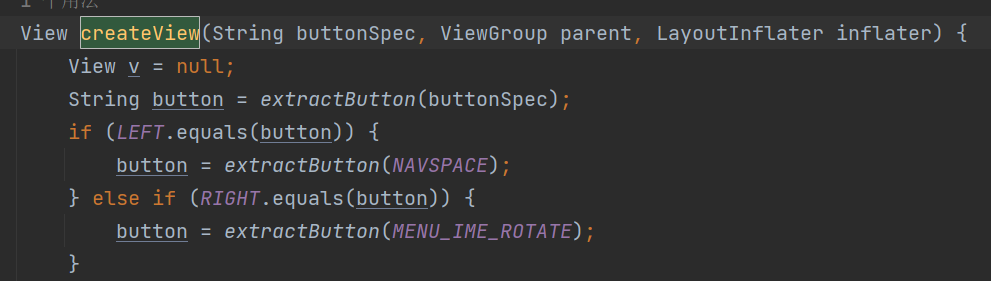


观察控件文件



分别是左,返回,桌面,最近菜单,右

三个按钮的布局分别是home.xml back.xml recent\_app.xml

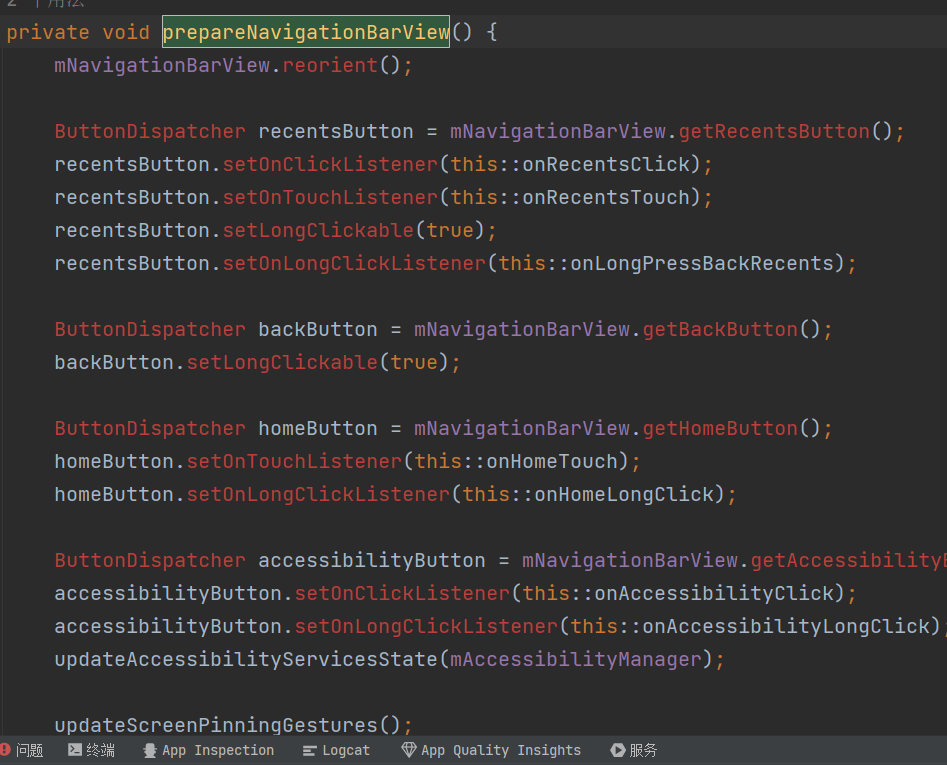


createView方法主要是创建对应的布局文件，并且添加到导航栏中

3.导航栏的触发事件

NavigationBarFragment类

观察prepareNavigationBarView()方法



这个方法设置了底部三个虚拟按键的短触事件和长按事件