1. **systemUI的結構**

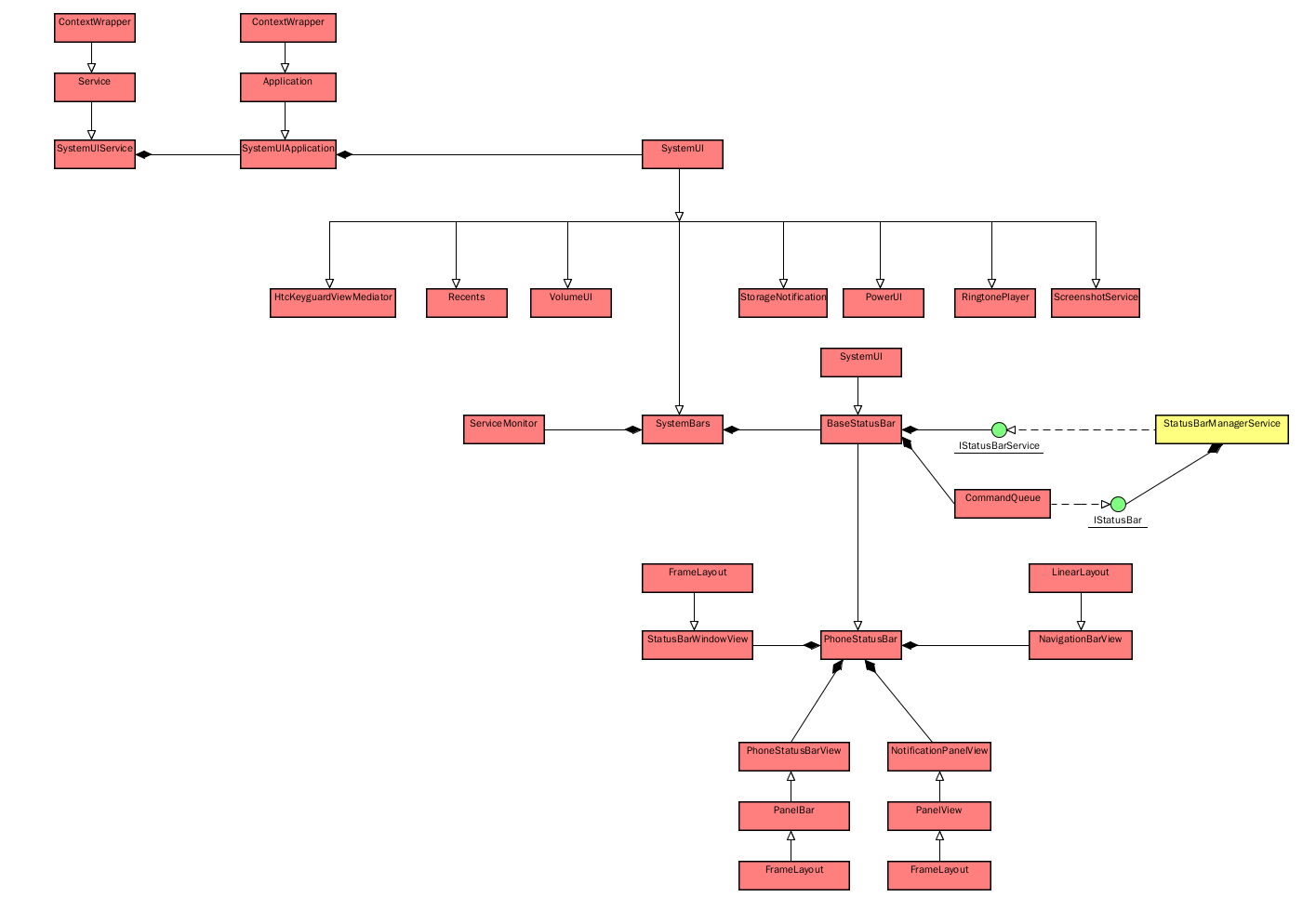
statusBar

navigation Bar

Notification

RecentApp

Wallpaper



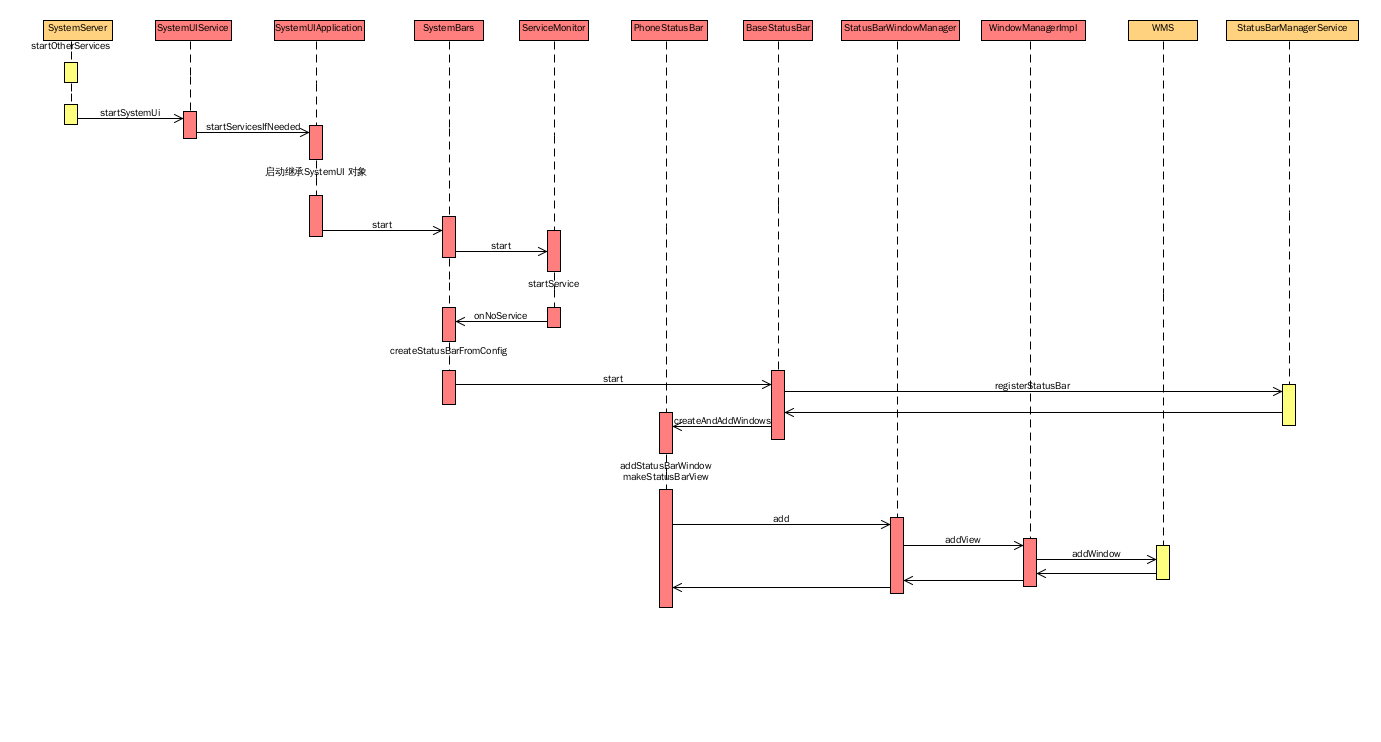
1. **systemUI的啟動**

Service、Application的使用

Application和Activity,Service一样是android框架的一个系统组件，当android程序启动时系统会创建一个 application对象，用来存储系统的一些信息。通常我们是不需要指定一个Application的，这时系统会自动帮我们创建，如果需要创建自己 的Application，也很简单创建一个类继承 Application并在manifest的application标签中进行注册(只需要给Application标签增加个name属性把自己的 Application的名字定入即可)。

android系统会为每个程序运行时创建一个Application类的对象且仅创建一个，所以Application可以说是单例 (singleton)模式的一个类.且application对象的生命周期是整个程序中最长的，它的生命周期就等于这个程序的生命周期。因为它是全局 的单例的，所以在不同的Activity,Service中获得的对象都是同一个对象。所以通过Application来进行一些，数据传递，数据共享 等,数据缓存等操作。

SystemUIApplication 對象中保存繼承SystemUI類的對象數組。



1. **makeStatusBarView 函數詳解和layout結構**

StatusBarWindowView mStatusBarWindow

Inflate layout

setOnTouchListener

PhoneStatusBarView mStatusBarView (系統圖標區域、通知圖標區域、時間)

setBar

setPanelHolder

NotificationPanelView mNotificationPanel

setStatusBar

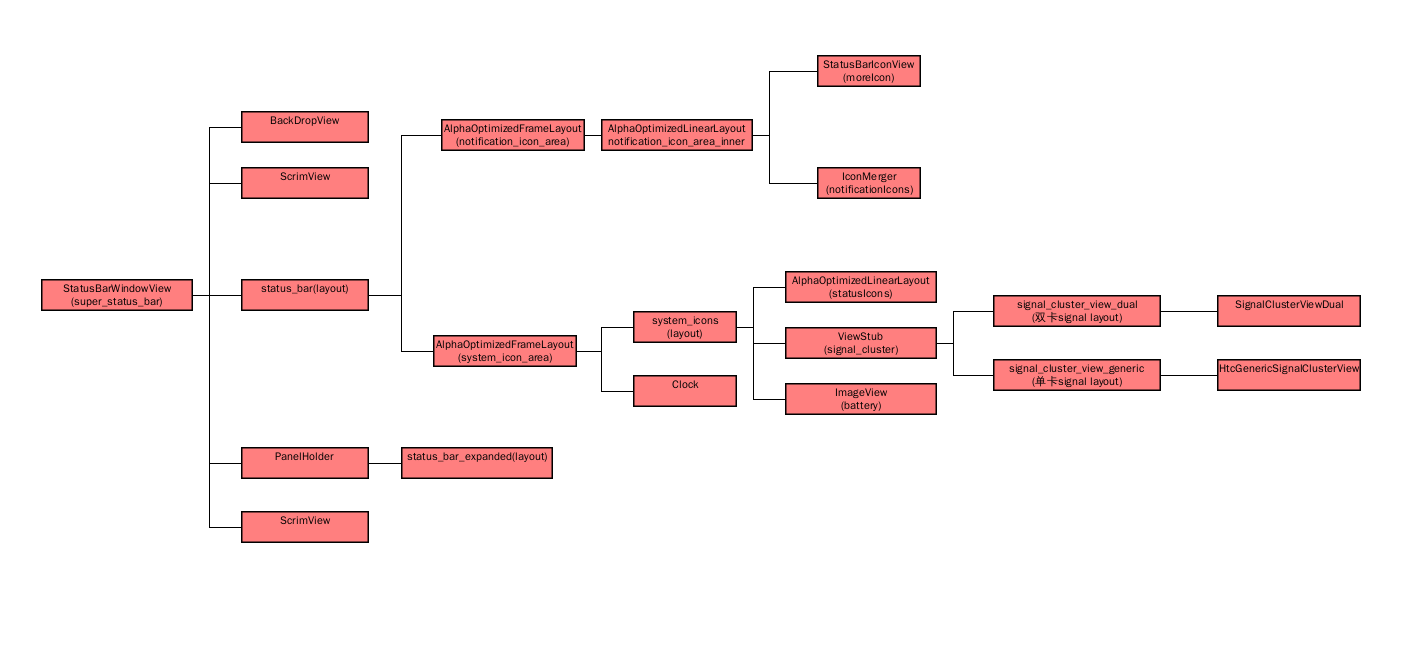
NavigationBarView mNavigationBarView

setDisabledFlags

setBar

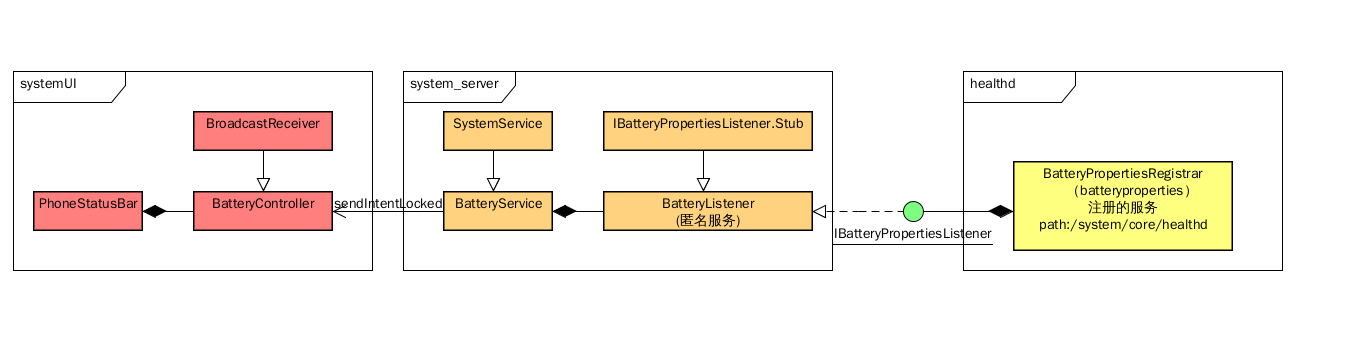
layout 關係：

super\_status\_bar(mStatusBarWindow)->status\_bar(mStatusBarView)-> status\_bar\_expanded(PanelHolder)



1. **status bar上狀態的獲取：**

**1.Battery更新**



BatteryController 註冊三個broadcast的action。

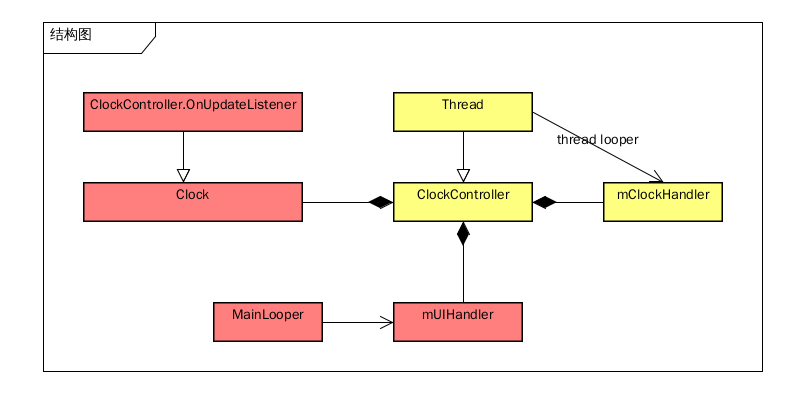
Intent.ACTION\_BATTERY\_CHANGED、

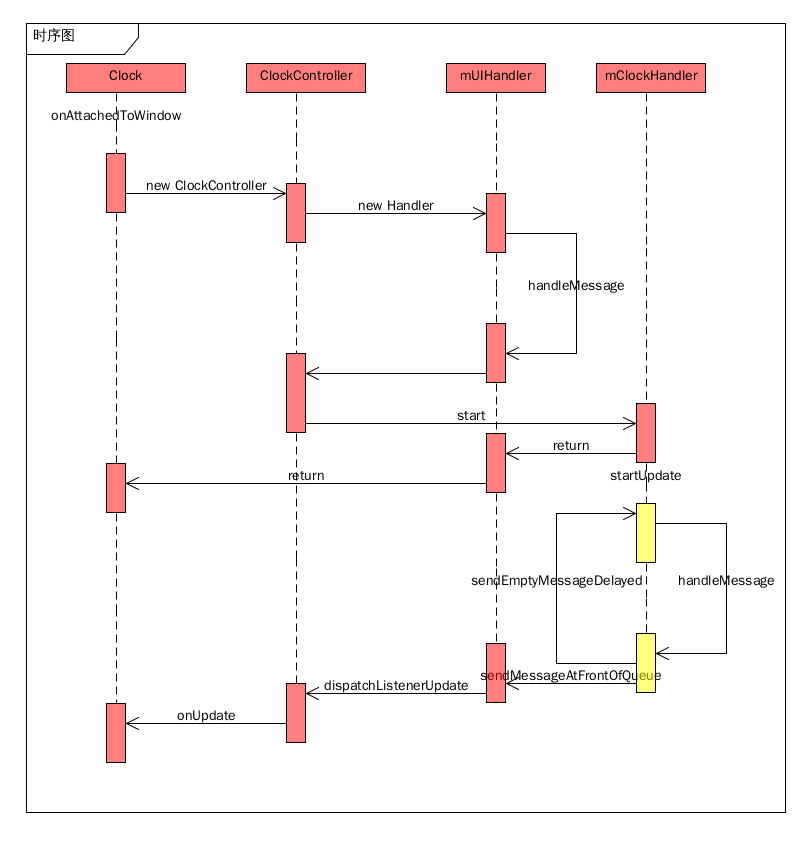
PowerManager.ACTION\_POWER\_SAVE\_MODE\_CHANGED

PowerManager.ACTION\_POWER\_SAVE\_MODE\_CHANGING

接收BatteryService 發生過來跟新電量信息的broadcast ACTION\_BATTERY\_CHANGED ，BatteryService會註冊一個匿名服務到batteryproperties 服務中來更新電量的信息。

**2.時間的更新**





啟動ClockController這個線程，每隔到整分鐘發送一個message到主線程隊列中來更新時間（Clock 控件的實現和使用）。

總結：

1.框架的使用場景

事件源一個（什麼時候更新時間），處理有多個（多個界面的刷新）

比如：

2.為什使用該框架

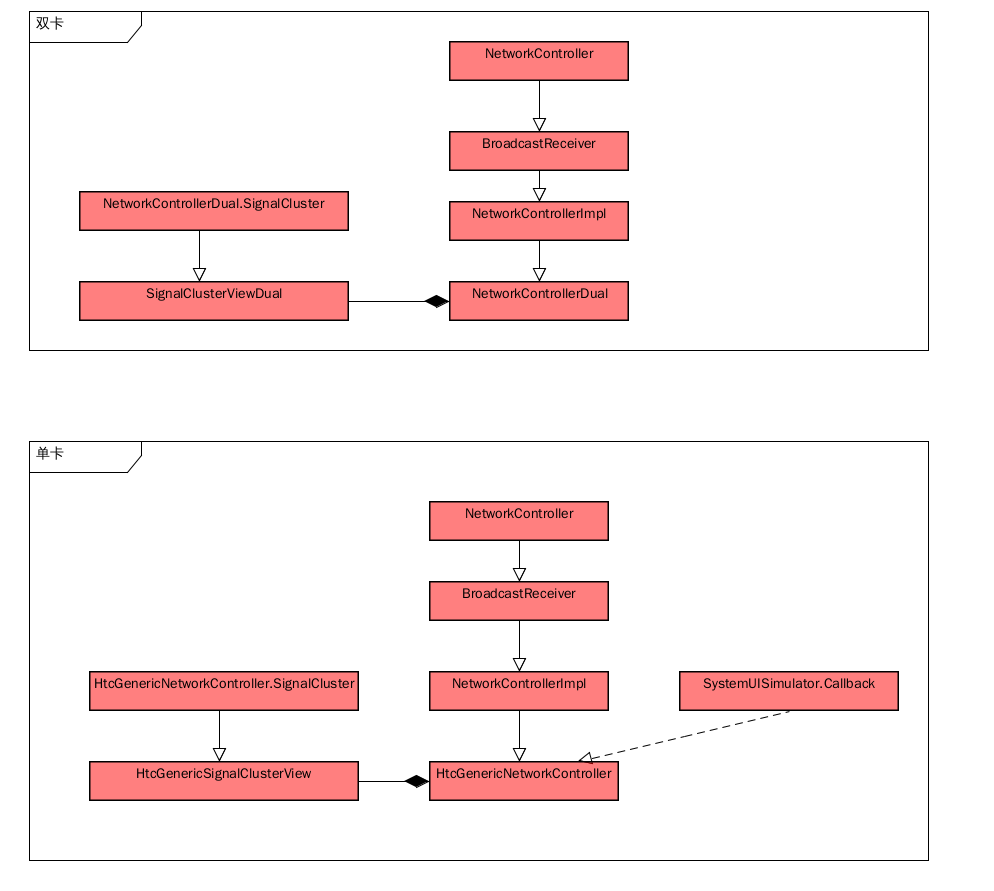
更新事件的觸發不能放在主線程中，Clock組件可以多個地方使用。

**3.手機信號大小更新**

ViewStub的使用和include的區別。

ViewStub是一个轻量级的View，它一个看不见的，不占布局位置，占用资源非常小的控件。可以为ViewStub指定一个布局，在Inflate布局的时候，只有ViewStub会被初始化，然后当ViewStub被设置为可见的时候，或是调用了ViewStub.inflate()的时候，ViewStub所向的布局就会被Inflate和实例化，然后ViewStub的布局属性都会传给它所指向的布局。这样，就可以使用ViewStub来方便的在运行时，要还是不要显示某个布局。

框架圖：



更新狀態：

//wifi

WifiManager.RSSI\_CHANGED\_ACTION

Wifi 信號強度的broadcast

WifiManager.WIFI\_STATE\_CHANGED\_ACTION

Wifi 開始鏈接\斷開時發送的broadcast

WifiManager.NETWORK\_STATE\_CHANGED\_ACTION

Wifi 鏈接是否成功的broadcast

//sim state

TelephonyIntents.ACTION\_SIM\_STATE\_CHANGED

更新sim卡的狀態

//airplane

Intent.ACTION\_AIRPLANE\_MODE\_CHANGED

飛行模式的改變

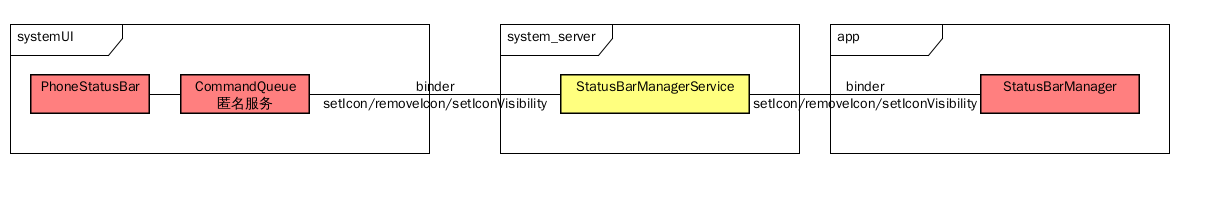
//

TelephonyIntents.ACTION\_SUBINFO\_RECORD\_UPDATED

//

ConnectivityManager.CONNECTIVITY\_ACTION\_IMMEDIATE

**5.statusIcons 圖標的顯示和更新**



PhoneStatusBarPolicy 負責更新：

private static final String SLOT\_SYNC\_ACTIVE = "sync\_active";

private static final String SLOT\_CAST = "cast";

private static final String SLOT\_BLUETOOTH = "bluetooth";

private static final String SLOT\_TTY = "tty";

private static final String SLOT\_ZEN = "zen";

private static final String SLOT\_VOLUME = "volume";

private static final String SLOT\_CDMA\_ERI = "cdma\_eri";

private static final String SLOT\_ALARM\_CLOCK = "alarm\_clock";

AlarmManager.ACTION\_NEXT\_ALARM\_CLOCK\_CHANGED

Intent.ACTION\_SYNC\_STATE\_CHANGED

BluetoothAdapter.ACTION\_STATE\_CHANGED

AudioManager.RINGER\_MODE\_CHANGED\_ACTION

TelecomManager.ACTION\_CURRENT\_TTY\_MODE\_CHANGED

Intent.ACTION\_USER\_SWITCHED

com.android.nfc

**6.NotificationIconArea 圖標的更新**

知識點：Parcelable（Notification）：

動畫：

AsyncTask：

使用：

繼承AsyncTask類。

實現doInBackground 函數。（輔助線程）

調用execute函數。

AsyncTask 線程和主線程通信：

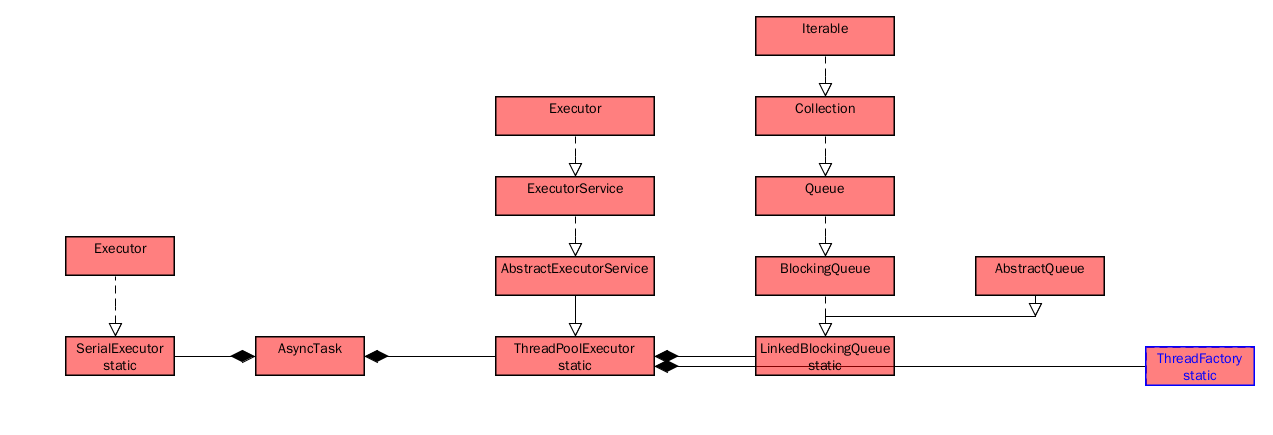
實現onProgressUpdate 函數。（主線程）

在doInBackground 函數中調用publishProgress函數。

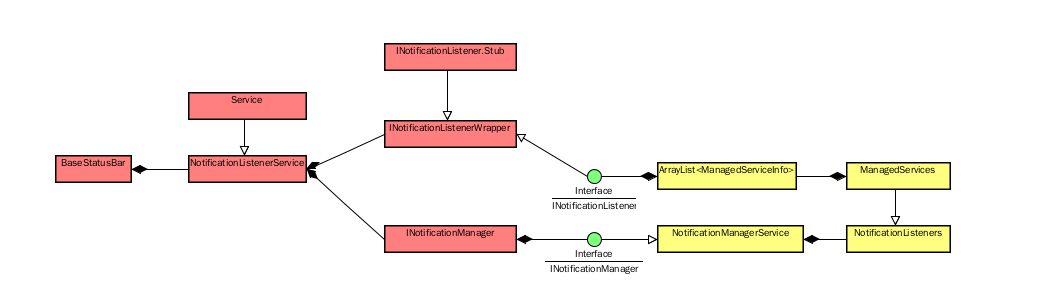
原理：

1.addWorker

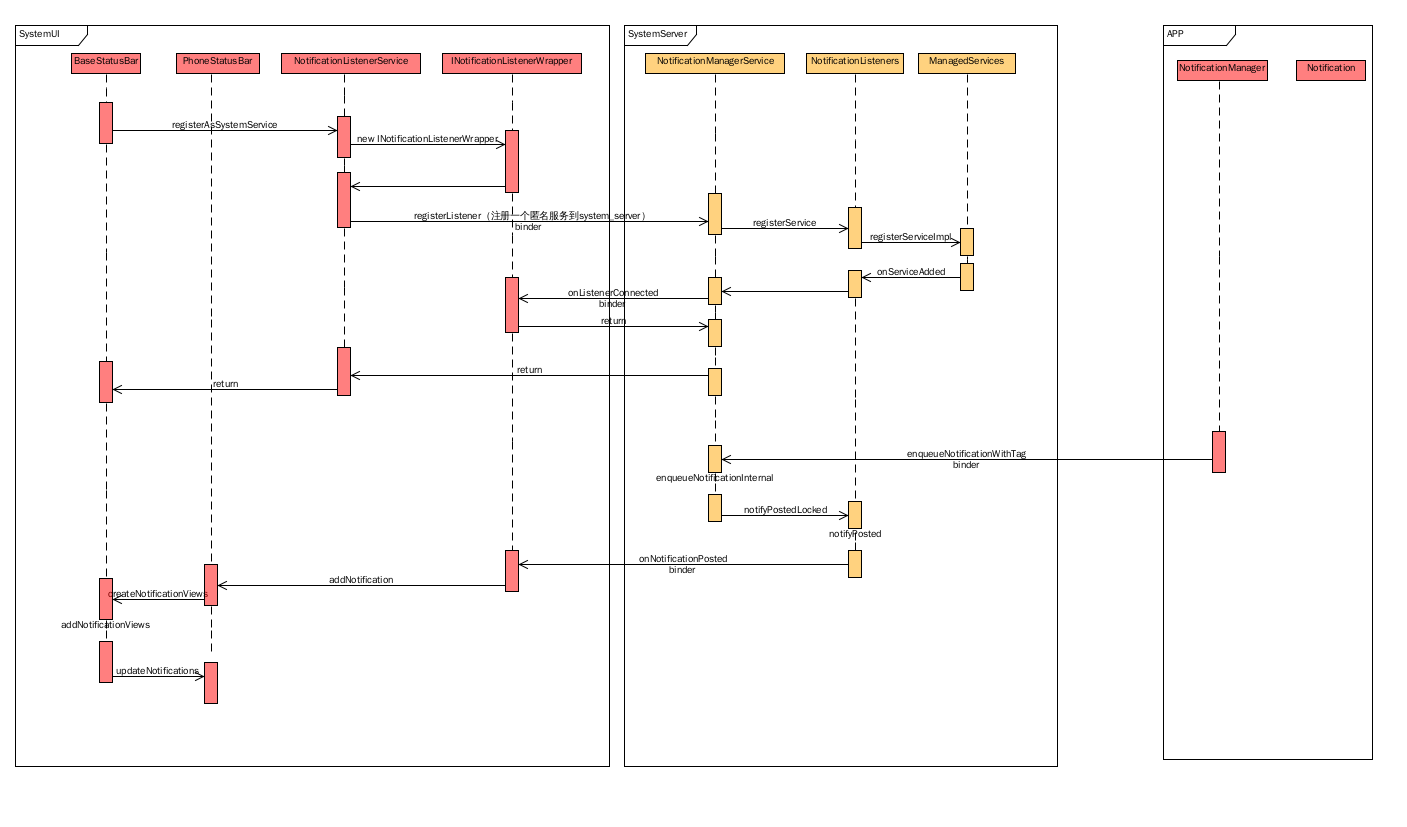
2.ThreadPoolExecutor.runWorker



Notification 結構圖

****

Notification 初始化和app notification流程圖

****

**7. PanelHolder**

知識點：

**1.下拉動畫**

**Cloneable：**

Java的淺拷貝和深拷貝

Implements Cloneable

實現clone()函數。

Object obj1 = Object obj2；

obj1 == obj2; //true

obj1.equals(obj2); //true

Object obj2 = obj1.clone();

Obj2 == obj1; //false

Obj2.equals(obj1); // true;

**動畫的種類：**

**View 動畫：**

**Frame 動畫：**

**屬性 動畫：**

**2. touch 事件在view樹中的傳遞流程**

at com.android.systemui.statusbar.phone.PanelBar.onTouchEvent(PanelBar.java:102)

atcom.android.systemui.statusbar.phone.PhoneStatusBarView.onTouchEvent(PhoneStatusBarView.java:144)

at android.view.View.dispatchTouchEvent(View.java:8538)

at android.view.ViewGroup.dispatchTransformedTouchEvent(ViewGroup.java:2541)

at android.view.ViewGroup.dispatchTouchEvent(ViewGroup.java:2220)

at android.view.ViewGroup.dispatchTransformedTouchEvent(ViewGroup.java:2547)

at android.view.ViewGroup.dispatchTouchEvent(ViewGroup.java:2240)

at android.view.View.dispatchPointerEvent(View.java:8760)

at android.view.ViewRootImpl$ViewPostImeInputStage.processPointerEvent(ViewRootImpl.java:4700)

at android.view.ViewRootImpl$ViewPostImeInputStage.onProcess(ViewRootImpl.java:4548)

at android.view.ViewRootImpl$InputStage.deliver(ViewRootImpl.java:4067)

at android.view.ViewRootImpl$InputStage.onDeliverToNext(ViewRootImpl.java:4126)

at android.view.ViewRootImpl$InputStage.forward(ViewRootImpl.java:4092)

at android.view.ViewRootImpl$AsyncInputStage.forward(ViewRootImpl.java:4205)

at android.view.ViewRootImpl$InputStage.apply(ViewRootImpl.java:4100)

at android.view.ViewRootImpl$AsyncInputStage.apply(ViewRootImpl.java:4262)

at android.view.ViewRootImpl$InputStage.deliver(ViewRootImpl.java:4072)

at android.view.ViewRootImpl$InputStage.onDeliverToNext(ViewRootImpl.java:4126)

at android.view.ViewRootImpl$InputStage.forward(ViewRootImpl.java:4092)

at android.view.ViewRootImpl$InputStage.apply(ViewRootImpl.java:4100)

at android.view.ViewRootImpl$InputStage.deliver(ViewRootImpl.java:4072)

at android.view.ViewRootImpl.deliverInputEvent(ViewRootImpl.java:6539)

at android.view.ViewRootImpl.doProcessInputEvents(ViewRootImpl.java:6489)

at android.view.ViewRootImpl.enqueueInputEvent(ViewRootImpl.java:6435)

at android.view.ViewRootImpl$WindowInputEventReceiver.onInputEvent(ViewRootImpl.java:6701)

at android.view.InputEventReceiver.dispatchInputEvent(InputEventReceiver.java:212)

at android.os.MessageQueue.nativePollOnce(Native Method)

at android.os.MessageQueue.next(MessageQueue.java:143)

at android.os.Looper.loop(Looper.java:155)

at android.app.ActivityThread.main(ActivityThread.java:5894)

at java.lang.reflect.Method.invoke(Native Method)

at java.lang.reflect.Method.invoke(Method.java:372)

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:1005)

at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:800)

**（1）.boolean isTransformedTouchPointInView(float x, float y, View child,**

**PointF outLocalPoint)；**

**（2）.boolean pointInView(float localX, float localY)；**

**（3）.down 在哪個view之上，touch event 就給這個view處理，不管move 區域是否在這個view之內。**

**3.layout類型**

**FrameLayout**

**LinearLayout**

**RelativeLayout**

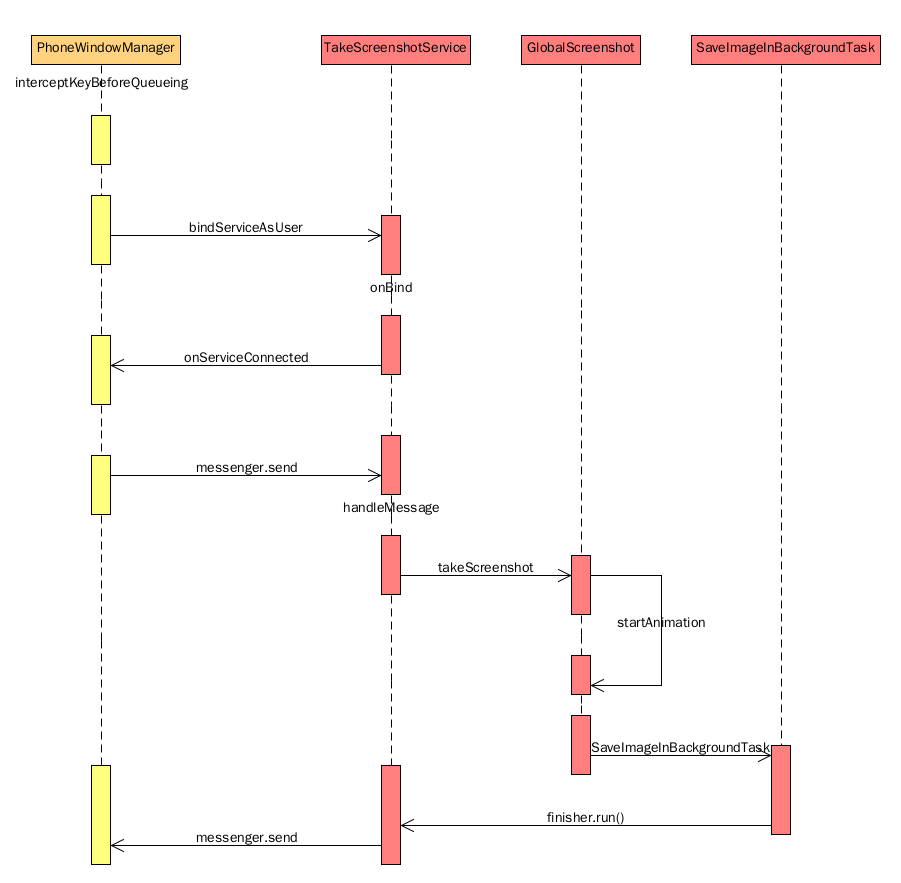
**TableLayout**

**8. screenshot**

**知識點：**

**Service的使用**

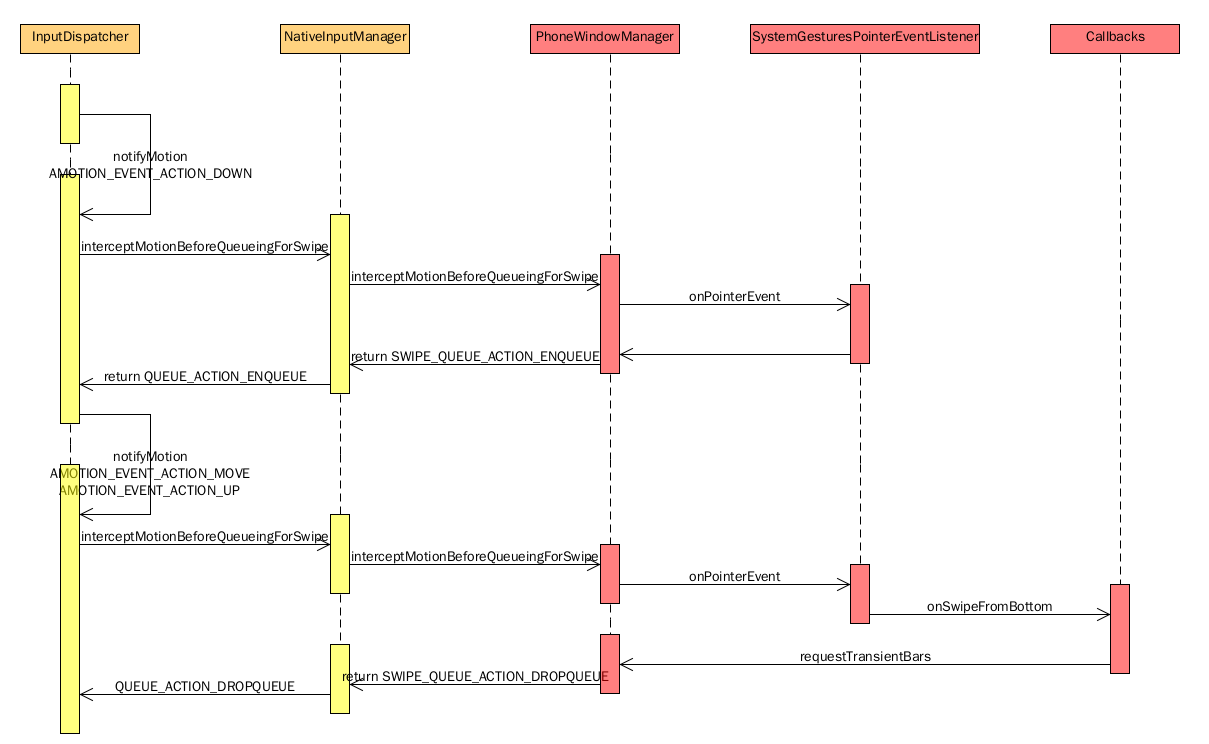
**進程中消息機制Messenger**

****

總結：很好的一個進程之間異步消息處理機制。（Messager + bindService）

Bindservice 啟動一個進程，同時在進程兩端建立Messenger（包含處理事件的handler），然後將兩個Messenger分別傳送到其他進程，兩個進程都可以向其它進程的主消息隊列中發送消息。

**9. navigation bar Display**

****

在InputDispatcher中对AMOTION\_EVENT\_ACTION\_DOWN或者mNeedSwipeChecking为true的Motion event进行处理的时候，回调到PhoneWindowManager中检查条件：

1.ACTION\_DOWN

2.navigation bar is prepare for swiping

3.in the correct region

4.not in lockscreen

如果条件满足告诉InputDispatcher该Down事件需要被拦截，InputDispatcher记录，当有move action要notify给InputDispatcher前检查该action是否是已经拦截的事件序列中的，如果是继续拦截该事件抛送给PhoneWindowManager进行navigation bar的显示的操作，PhoneWindowManager显示完之后通知InputDispatcher不需要再拦截该序列的事件。