一、SystemUi 概述

1.1 简介

SystemU I是一个持续的进程，为系统提供UI，作为Android系统的核心应用，SystemUI负责反馈系统及应用状态并与用户保持大量的交付。

1.2 路径

代码位置在

```
frameworks\base\packages\SystemUI
```

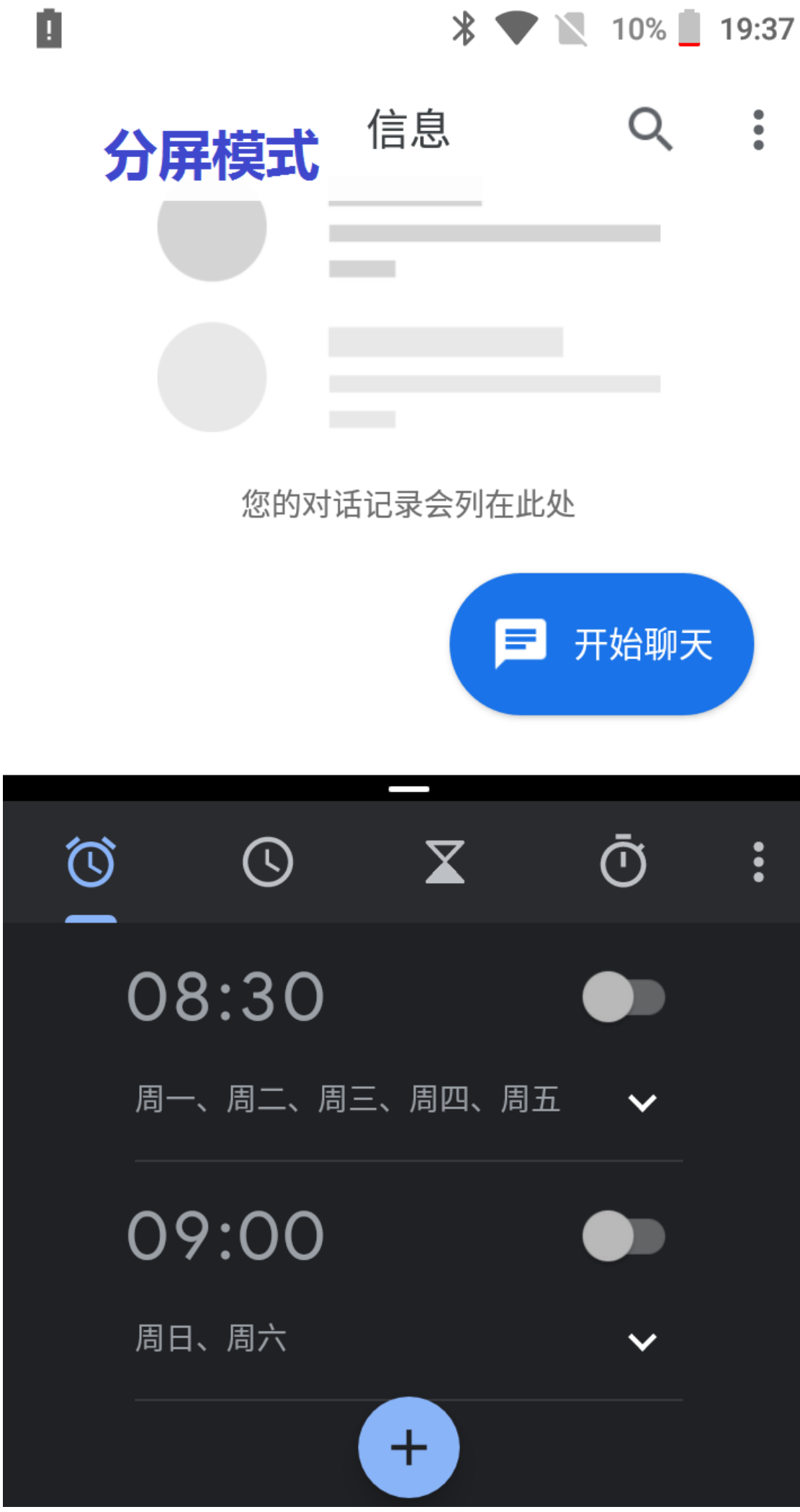
apk安装目录

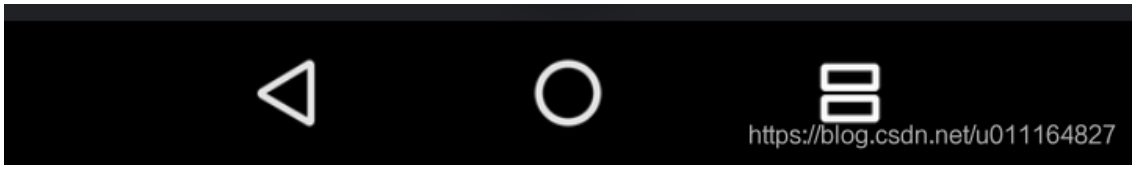
```
system/priv-app/SystemUI
```

不同手机的SystemUI可能有所不同，比如小米手机的安装目录就是system/priv-app/MiuiSystemUI

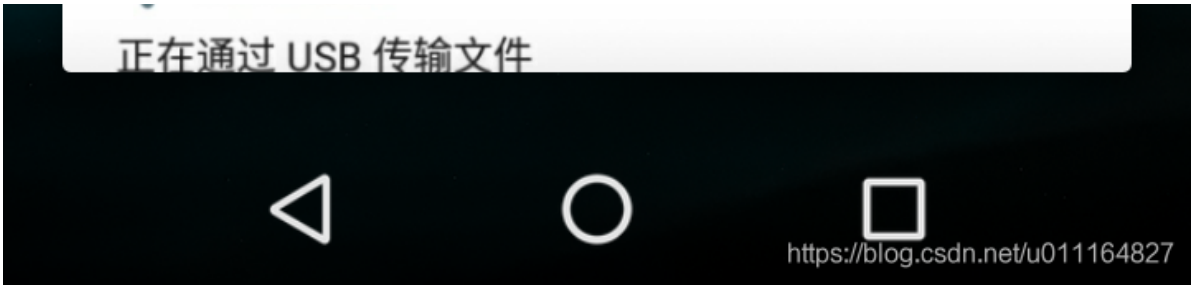
1.3 功能划分

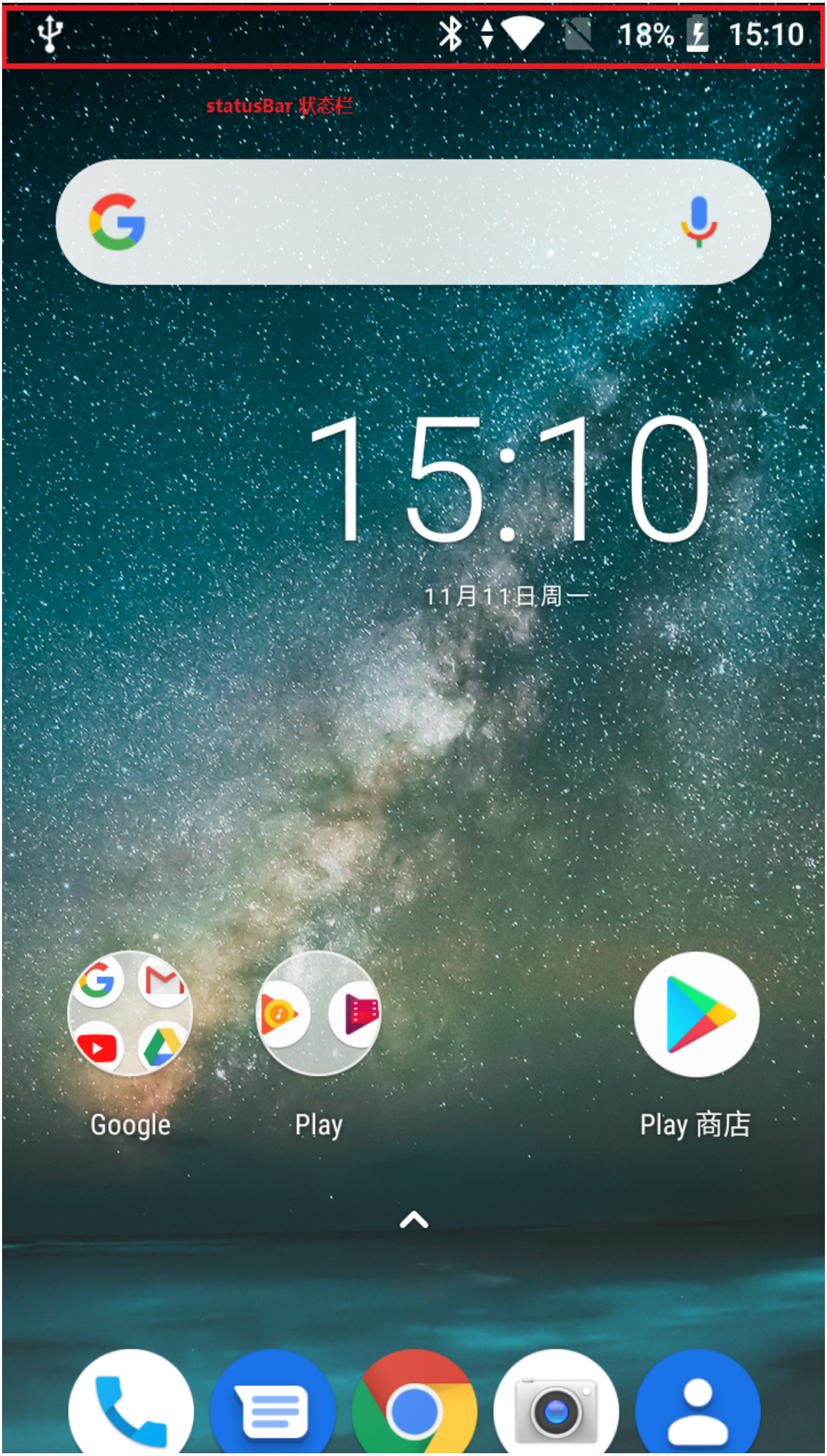
- StatusBar（状态栏）：通知消息提示和状态展示
- NavigationBar（导航栏）：返回，HOME，Recent
- KeyGuard（键盘锁）：锁屏模块
- Recents：近期应用管理，以堆叠的形式展示
- Notification Panel（通知面板）：展示系统或应用的通知内容，提供快速系统设置开关
- Volume：展示或控制音量的变化：媒体、铃音、通知、通话音量
- ScreenShot（截屏）：长按电源键+音量下键后截屏，用以展示截取的屏幕照片/内容
- PowerUI：主要处理和Power相关的事件。
- RingtonePlayer：铃音播放
- StackDivider：控制管理分屏
- PipUI：画中画管理（Android7.0）


10%
19:37
分屏模式
信息
您的对话记录会列在此处
开始聊天
08:30
周一、周二、周三、周四、周五
09:00
周日、周六

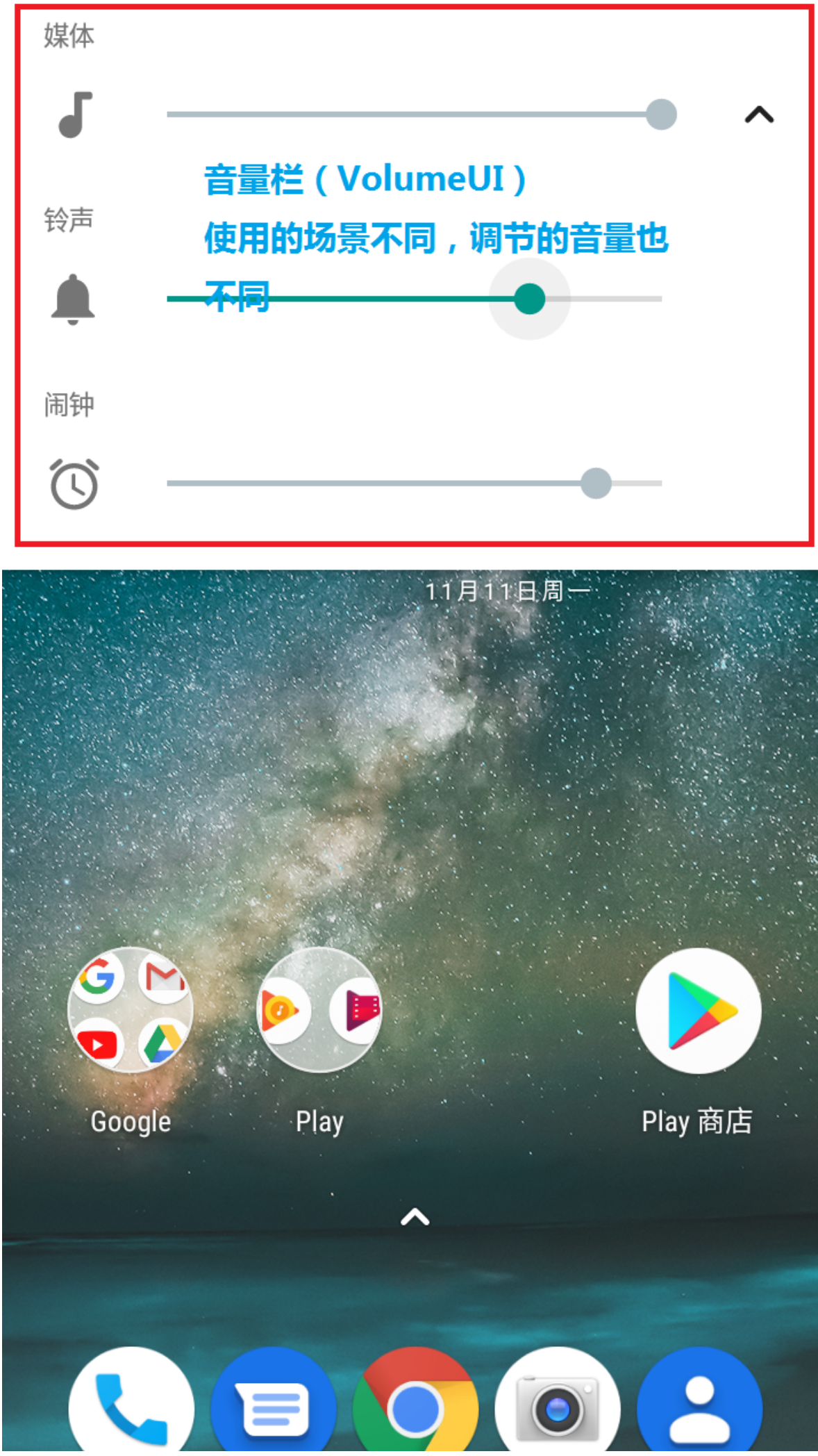

https://blog.csdn.net/u011164827


18% 15:13
设置
开启
相册
国内2.4G
蓝牙
省电模式
Recents近期任务
移动数据
飞行模式
自动旋转
11月11日周一
Android 系统


正在通过 USB 传输文件
https://blog.csdn.net/u011164827

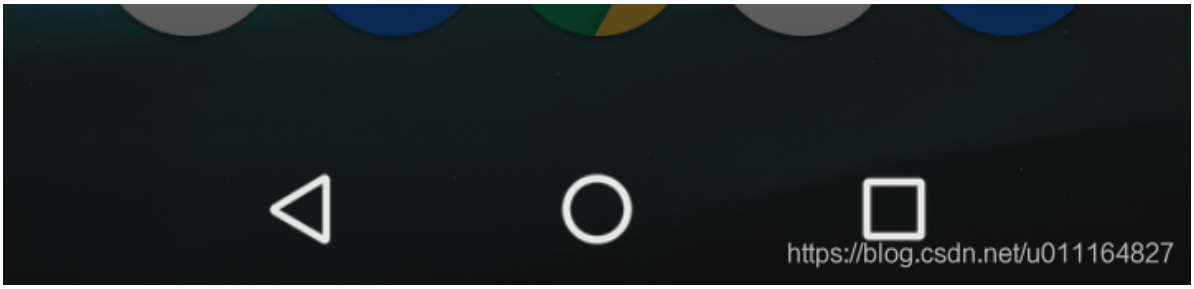

18% 15:10
statusBar 状态栏
15:10
11月11日周一
Google
Play
Play 商店


导航栏
https://blog.csdn.net/u011164827


媒体
音量栏（VolumeUI）
使用的场景不同，调节的音量也不同
铃声
闹钟
11月11日周一
Google
Play
Play 商店


https://blog.csdn.net/u011164827

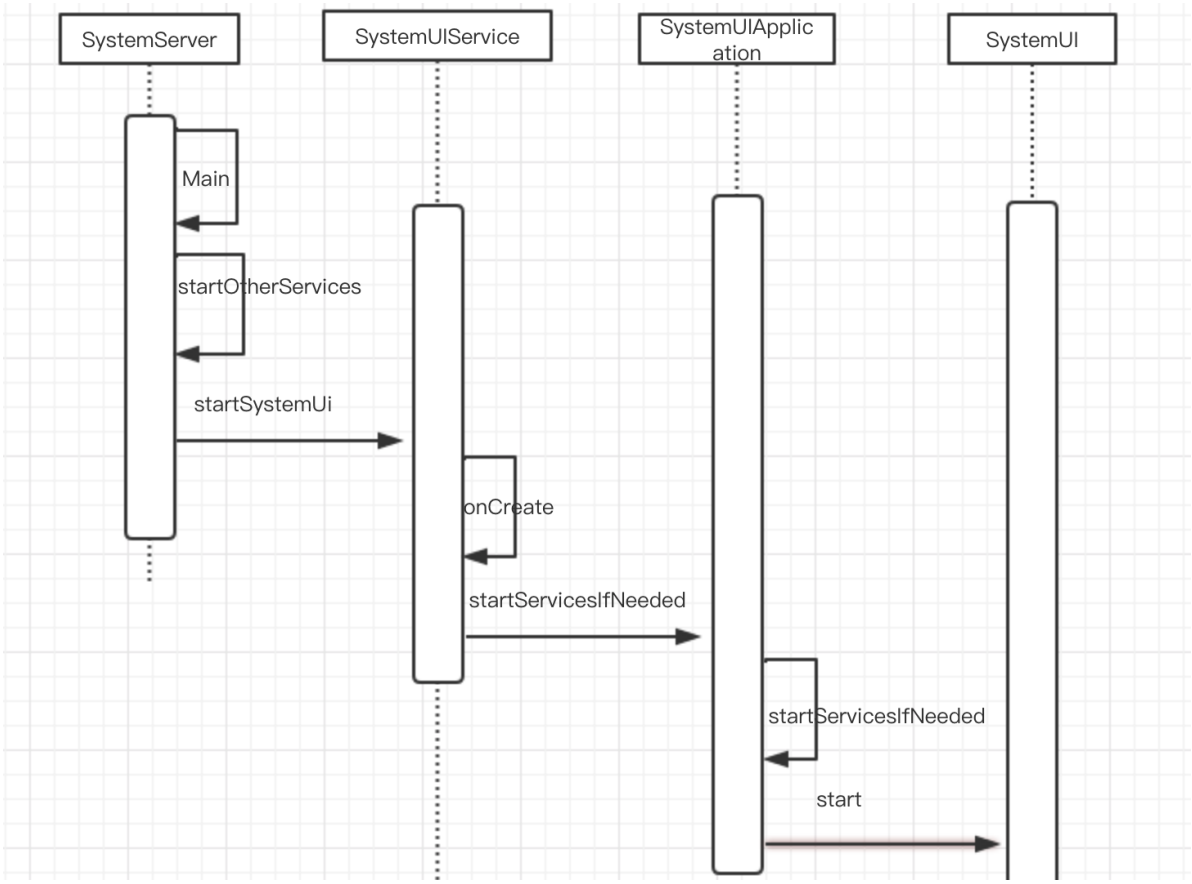

没有 SIM 卡 — 只能拨打紧急呼救电话
18%
15:10
亮度调节
国内2.4G
蓝牙
省电模式
QuickSettings,7.0后可以自定义
移动数据
飞行模式
自动旋转
11月11日周一
QS编辑入口
Android 系统
正在通过 USB 传输文件
点击即可查看更多选项。
Notification 消息通知



二、启动流程

SystemUI的启动是由SystemServer开始。SystemServer由Zygote fork生成的，进程名为system_server，system_server是framework的核心服务。Zygote启动过程中会调用startSystemServer()。

SystemUI的启动是从SystemUI的main方法开始。大致流程如下：



frameworks/base/services/java/com/android/server/SystemServer.java

```
    /**
      * The main entry point from zygote.
      */
```

```
    public static void main(String[] args) {
        new SystemServer().run();
    }


 private void run() {
 /*
  *省略代码
 */
 // Start services.
        try {
            traceBeginAndSlog("StartServices");
            startBootstrapServices();
            startCoreServices();
            startOtherServices(); //systemUI在这个里面启动
            SystemServerInitThreadPool.shutdown();
        } catch (Throwable ex) {
            Slog.e("System", "************************************************");
            Slog.e("System", "************ Failure starting system services",
ex);
            throw ex;
        } finally {
            traceEnd();
        }
}


/**
 * Starts a miscellaneous grab bag of stuff that has yet to be refactored
 * and organized.
 */
private void startOtherServices() {
    ... //省略大概1000行
    mActivityManagerService.systemReady(() -> {
        Slog.i(TAG, "Making services ready");
        ...
        traceBeginAndSlog("StartSystemUI");
        try {
            startSystemUi(context, windowManagerF);
        } catch (Throwable e) {
            reportWtf("starting System UI", e);
        }
        ...
    }
  }


static final void startSystemUi(Context context, WindowManagerService
windowManager) {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName("com.android.systemui",
                "com.android.systemui.SystemUIService"));
    intent.addFlags(Intent.FLAG_DEBUG_TRIAGED_MISSING);
    //Slog.d(TAG, "Starting service: " + intent);
    context.startServiceAsUser(intent, UserHandle.SYSTEM);
    windowManager.onSystemUiStarted();
}
```

SystemUI的run会启动各种重要的服务，在startOtherServices方法中启动SystemUI（）。在startOtherServices()中，通过调用AMS的systemReady()方法通知AMS准备就绪。systemReady()拥有一个名为goingCallback的Runnable实例作为参数，当AMS完成对systemReady()的处理后将会回调这一Runnable的run()方法。startSystemUi这个方法主要是启动com.android.systemui.SystemUIService这个服务，但是需要注意的是，这时候SystemUI还没启动成功，因为startOtherService()方法都还没有执行完毕。所以暂时还不会发送ACTION_BOOT_COMPLETE广播，该广播是在AMS中的finishBooting()中发送的。（该广播在我们SystemUIService中有做监听，用来判断是否完成系统启动。

frameworks/base/packages/SystemUI/src/com/android/systemui/SystemUIService.java

```
    @Override
37    public void onCreate() {
38        super.onCreate();
39        ((SystemUIApplication) getApplication()).startServicesIfNeeded();
40
41        // For debugging RescueParty
42        if (Build.IS_DEBUGGABLE &&
SystemProperties.getBoolean("debug.crash_sysui", false)) {
43            throw new RuntimeException();
44        }
45
46        if (Build.IS_DEBUGGABLE) {
47            // b/71353150 - looking for leaked binder proxies
48            BinderInternal.nSetBinderProxyCountEnabled(true);
49            BinderInternal.nSetBinderProxyCountWatermarks(1000,900);
50            BinderInternal.setBinderProxyCountCallback(
51                    new BinderInternal.BinderProxyLimitListener() {
52                        @Override
53                        public void onLimitReached(int uid) {
54                            Slog.w(SystemUIApplication.TAG,
55                                    "uid " + uid + " sent too many Binder
proxies to uid "
56                                    + Process.myUid());
57                        }
58                    }, Dependency.get(Dependency.MAIN_HANDLER));
59        }
60    }
```

这里面的核心代码就只有一行

```
((SystemUIApplication) getApplication()).startServicesIfNeeded();
```

这里需要说明一下Service和Application的创建先后

```
（1）实例Service；

（2）实例Application；

（3）Application实例执行onCreate方法；

（4）Service实例执行onCrate方法。
```

所以在SystemUI启动过程中，SystemUIApplication.java的onCreate方法先于SystemUIService.java的oncreate方法。

```
   @Override
62    public void onCreate() {
63        super.onCreate();
64        // Set the application theme that is inherited by all services. Note
that setting the
65        // application theme in the manifest does only work for activities.
Keep this in sync with
66        // the theme set there. 设置主题
67        setTheme(R.style.Theme_SystemUI);
68        //android 7.0以后引入，方便定制
69        SystemUIFactory.createFromConfig(this);
70        //判断是系统还是切换到其它用户
71        if (Process.myUserHandle().equals(UserHandle.SYSTEM)) {
72            IntentFilter bootCompletedFilter = new
IntentFilter(Intent.ACTION_BOOT_COMPLETED);
73
bootCompletedFilter.setPriority(IntentFilter.SYSTEM_HIGH_PRIORITY);
74            registerReceiver(new BroadcastReceiver() {
75                @Override
76                public void onReceive(Context context, Intent intent) {
77                    if (mBootCompleted) return;
78
79                    if (DEBUG) Log.v(TAG, "BOOT_COMPLETED received");
80                    unregisterReceiver(this);
81                    mBootCompleted = true;
82                    if (mServicesStarted) { //该变量表示SystemUIService是否已经启
动了
83                        final int N = mServices.length;
84                        for (int i = 0; i < N; i++) {
85                            mServices[i].onBootCompleted();
86                        }
87                    }
88
89
90                }
91            }, bootCompletedFilter);
92
93            IntentFilter localeChangedFilter = new
IntentFilter(Intent.ACTION_LOCALE_CHANGED);
94            registerReceiver(new BroadcastReceiver() {
95                @Override
96                public void onReceive(Context context, Intent intent) {
97                    if
(Intent.ACTION_LOCALE_CHANGED.equals(intent.getAction())) {
98                        if (!mBootCompleted) return;
99                        // Update names of SystemUi notification channels
100                        NotificationChannels.createAll(context);
101                    }
102                }
103            }, localeChangedFilter);
104        } else {
105            // We don't need to startServices for sub-process that is doing
some tasks.
106            // (screenshots, sweetsweetdesserts or tuner ..)
```

```
107             String processName = ActivityThread.currentProcessName();
108             ApplicationInfo info = getApplicationInfo();
109             if (processName != null && processName.startsWith(info.processName
+ ":")) {
110                 return;
111             }
112             // For a secondary user, boot-completed will never be called
because it has already
113             // been broadcasted on startup for the primary SystemUI process.
Instead, for
114             // components which require the SystemUI component to be
initialized per-user, we
115             // start those components now for the current non-system user.
                //非系统用户
116             startSecondaryUserServicesIfNeeded();
117         }
118     }
119
```

在Application的oncreate中会区分系统用户和非系统用户来区分流程。如果是系统用户会接收两个广播Intent.ACTION_BOOT_COMPLETED和Intent.ACTION_LOCALE_CHANGED。
Intent.ACTION_BOOT_COMPLETED是监听开机启动，从android 7.0以后，android提供FBE加密方式（https://source.android.google.cn/security/encryption/file-based），在这种情况下，要等到系统启动并锁屏界面解锁后，在进入到桌面过程中，系统才会发送该广播，所以接收该广播的处理逻辑比较延后，SystemUIService启动完后才接收到该广播，所以startServicesIfNeeded方法会先执行，开机广播只会处理一次，就会注销该广播，以后就不会再接收了。mService[]数组存储的是SystemUI的子服务，当整个系统启动完成后，这里面的每个子服务都会执行onBootCompleted()方法，让各个子服务知道系统启动完成了，开始执行任务。

Intent.ACTION_LOCALE_CHANGED广播是用于监听设备当前区域设置已更改时发出的广播，简单来说就是修改语言时发出的广播（暂时不知道其它动作是否也会发送该广播）。

```
startSecondaryUserServicesIfNeeded
```

只会发生在系统启动之后。

```
    /**
133     * Ensures that all the Secondary user SystemUI services are running. If
they are already
134     * running, this is a no-op. This is needed to conditinally start all the
services, as we only
135     * need to have it in the main process.
136     * <p>This method must only be called from the main thread.</p>
137     */
138    void startSecondaryUserServicesIfNeeded() {
139        String[] names =
140
 getResources().getStringArray(R.array.config_systemUIServiceComponentsPerUser);
//需要启动的服务
141        startServicesIfNeeded(names);
142    }
```

config_systemUIServiceComponentsPerUser在frameworks/base/packages/SystemUI/res/values/config.xml中，需要说明的是，Android9.0才引入这种模式，之前版本都是通过数组直接写在代码中。

```
    <!-- SystemUI Services (per user): The classes of the stuff to start for each
user. This is a subset of the config_systemUIServiceComponents -->
359    <string-array name="config_systemUIServiceComponentsPerUser"
translatable="false">
360        <item>com.android.systemui.Dependency</item> //一种静态依赖项
361        <item>com.android.systemui.util.NotificationChannels</item> //通知
362        <item>com.android.systemui.recents.Recents</item> //多任务
363    </string-array>
```

再看一下

```
  /**
121     * Makes sure that all the SystemUI services are running. If they are
already running, this is a
122     * no-op. This is needed to conditinally start all the services, as we
only need to have it in
123     * the main process.
124     * <p>This method must only be called from the main thread.</p>
125     */
126
127    public void startServicesIfNeeded() {
128        String[] names =
getResources().getStringArray(R.array.config_systemUIServiceComponents);
129        startServicesIfNeeded(names);
130    }
```

需要启动以下服务

```
    <!-- SystemUI Services: The classes of the stuff to start. -->
331    <string-array name="config_systemUIServiceComponents"
translatable="false">
332        <item>com.android.systemui.Dependency</item>
333        <item>com.android.systemui.util.NotificationChannels</item>
334
 <item>com.android.systemui.statusbar.CommandQueue$CommandQueueStart</item>
335        <item>com.android.systemui.keyguard.KeyguardViewMediator</item>
336        <item>com.android.systemui.recents.Recents</item>
337        <item>com.android.systemui.volume.VolumeUI</item>
338        <item>com.android.systemui.stackdivider.Divider</item>
339        <item>com.android.systemui.SystemBars</item>
340        <item>com.android.systemui.usb.StorageNotification</item>
341        <item>com.android.systemui.power.PowerUI</item>
342        <item>com.android.systemui.media.RingtonePlayer</item>
343        <item>com.android.systemui.keyboard.KeyboardUI</item>
344        <item>com.android.systemui.pip.PipUI</item>
345        <item>com.android.systemui.shortcut.ShortcutKeyDispatcher</item>
346        <item>@string/config_systemUIVendorServiceComponent</item>
347        <item>com.android.systemui.util.leak.GarbageMonitor$Service</item>
348        <item>com.android.systemui.LatencyTester</item>
349
 <item>com.android.systemui.globalactions.GlobalActionsComponent</item>
350        <item>com.android.systemui.ScreenDecorations</item>
351        <item>com.android.systemui.fingerprint.FingerprintDialogImpl</item>
352        <item>com.android.systemui.SliceBroadcastRelayHandler</item>
353    </string-array>
```

最终是在

```
private void startServicesIfNeeded(String[] services) {
145         if (mServicesStarted) { //如果SystemUI已经启动就返回
146             return;
147         }
148         mServices = new SystemUI[services.length];
149 
150         if (!mBootCompleted) {
151             // check to see if maybe it was already completed long before we
began
152             // see ActivityManagerService.finishBooting()
153             if ("1".equals(SystemProperties.get("sys.boot_completed"))) {
154                 mBootCompleted = true;
155                 if (DEBUG) Log.v(TAG, "BOOT_COMPLETED was already sent");
156             }
157         }
158 
159         Log.v(TAG, "Starting SystemUI services for user " +
160                 Process.myUserHandle().getIdentifier() + ".");
161         TimingsTraceLog log = new TimingsTraceLog("SystemUIBootTiming",
162                 Trace.TRACE_TAG_APP);
163         log.traceBegin("StartServices");
164         final int N = services.length;
165         for (int i = 0; i < N; i++) {
166             String clsName = services[i];
167             if (DEBUG) Log.d(TAG, "loading: " + clsName);
168             log.traceBegin("StartServices" + clsName);
169             long ti = System.currentTimeMillis();
170             Class cls;
171             try {
172                 cls = Class.forName(clsName);
173                 mServices[i] = (SystemUI) cls.newInstance();
174             } catch(ClassNotFoundException ex){
175                 throw new RuntimeException(ex);
176             } catch (IllegalAccessException ex) {
177                 throw new RuntimeException(ex);
178             } catch (InstantiationException ex) {
179                 throw new RuntimeException(ex);
180             }
181 
182             mServices[i].mContext = this;
183             mServices[i].mComponents = mComponents;
184             if (DEBUG) Log.d(TAG, "running: " + mServices[i]);
185             mServices[i].start(); //启动子服务
186             log.traceEnd();
187 
188             // Warn if initialization of component takes too long
189             ti = System.currentTimeMillis() - ti;
190             if (ti > 1000) {
191                 Log.w(TAG, "Initialization of " + cls.getName() + " took " +
ti + " ms");
192             }
193             if (mBootCompleted) { //如果已经启动完成
194                 mServices[i].onBootCompleted();
195             }
196         }
```

```
197         log.traceEnd();
198         Dependency.get(PluginManager.class).addPluginListener(
199                 new PluginListener<OverlayPlugin>() {
200                     private ArraySet<OverlayPlugin> mOverlays;
201 
202                     @Override
203                     public void onPluginConnected(OverlayPlugin plugin,
Context pluginContext) {
204                         StatusBar statusBar = getComponent(StatusBar.class);
205                         if (statusBar != null) {
206                             plugin.setup(statusBar.getStatusBarWindow(),
207                                     statusBar.getNavigationBarView());
208                         }
209                         // Lazy init.
210                         if (mOverlays == null) mOverlays = new ArraySet<>();
211                         if (plugin.holdStatusBarOpen()) {
212                             mOverlays.add(plugin);
213 
 Dependency.get(StatusBarWindowManager.class).setStateListener(b ->
214                                     mOverlays.forEach(o ->
o.setCollapseDesired(b)));
215 
 Dependency.get(StatusBarWindowManager.class).setForcePluginOpen(
216                                     mOverlays.size() != 0);
217 
218                         }
219                     }
220 
221                     @Override
222                     public void onPluginDisconnected(OverlayPlugin plugin) {
223                         mOverlays.remove(plugin);
224 
 Dependency.get(StatusBarWindowManager.class).setForcePluginOpen(
225                                 mOverlays.size() != 0);
226                     }
227                 }, OverlayPlugin.class, true /* Allow multiple plugins */);
228 
229         mServicesStarted = true;
230     }
```

这里主要通过反射的方式将前面的各个子服务实例化，并执行对这些对象中的start方法，来启动这些服务。

至此，SystemUI的启动分析完成。