

Unisoc Confidential For hiar

Android 10.0 SystemUI 状态栏及快捷设置介绍

文档版本
发布日期

V1.0
2020-09-30

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档主要适用于 Android 10.0。根据具体流程，介绍 SystemUI 中状态栏和快捷设置功能，分析状态栏和快捷设置中常见的需求和问题。

读者对象


本文档适用于 Android 开发人员。

缩略语

缩略语	英文全名	中文解释
SystemUI	System User Interface	系统用户界面
QS	Quick Settings	快捷设置
WIFI	Wireless Fidelity	无线局域网

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-09-30	第一次正式发布。

关键字

状态栏、快捷设置、SystemUI、StatusBar、QuickSettings、QS。

Unisoc Confidential For Hiar

目 录

1 状态栏.....	1
1.1 介绍.....	1
1.1.1 锁屏状态栏.....	2
1.1.2 解锁状态栏.....	3
1.1.3 下拉状态栏.....	5
1.2 主要控件.....	5
1.2.1 运营商信息.....	5
1.2.2 时间和日期.....	6
1.2.3 电池.....	6
1.2.4 系统图标.....	7
1.2.5 通知图标.....	11
1.2.6 图标容器.....	11
1.3 主要功能.....	13
1.3.1 多图标成点.....	13
1.3.2 图标反色.....	15
1.3.3 StatusBar 服务.....	16
1.3.4 刘海屏.....	17
1.4 状态栏客制化.....	21
1.4.1 新增系统图标.....	21
1.4.2 更改系统图标.....	24
1.5 常见问题分析.....	25
2 快捷设置.....	28
2.1 快捷设置介绍.....	28
2.2 主要控件.....	34
2.2.1 QSFragment.....	34
2.2.2 QSPanel 和 QuickQSPanel.....	35
2.2.3 QSTileView.....	36
2.3 主要功能.....	36
2.3.1 点击事件处理.....	36
2.3.2 亮度调节.....	38
2.4 快捷设置客制化.....	39
2.4.1 新增快捷设置.....	39
2.4.2 添加亮度模式图标.....	42
2.5 常见问题分析.....	43

图目录

图 1-1 状态栏的启动过程	1
图 1-2 锁屏状态下显示的状态栏	2
图 1-3 updateLayoutParamsNoCutout()	3
图 1-4 解锁后显示的状态栏	3
图 1-5 status_bar.xml 布局代码	4
图 1-6 下拉状态栏界面	5
图 1-7 系统图标	7
图 1-8 系统图标的定义代码	7
图 1-9 系统图标关系	8
图 1-10 StatusBarSignalPolicy 控制信号类系统图标	9
图 1-11 PhoneStatusBarPolicy 控制状态类系统图标	9
图 1-12 Mobile 图标更新过程	10
图 1-13 通知图标的创建和更新流程	11
图 1-14 计算图标数量和剩余空间	12
图 1-15 计算第一个无法显示的图标 index	12
图 1-16 通过 firstUnderflowIndex 进行设置	13
图 1-17 状态栏多图标成点	13
图 1-18 状态栏图标初始化过程	14
图 1-19 状态栏图标刷新流程	14
图 1-20 onDarkChanged 方法设置图标颜色	15
图 1-21 启动 StatusBar 服务	16
图 1-22 adb shell cmd statusbar help	16
图 1-23 图片转换后的数据	17
图 1-24 生成矢量图 xml	17
图 1-25 替换 pathData	18
图 1-26 缩小取值	18

图 1-27 调整初始笔画位置	19
图 1-28 刘海布局设置	20
图 1-29 配置 config_statusBarIcons 参数.....	21
图 1-30 图片资源文件	21
图 1-31 添加 ColorInversionController.java.....	22
图 1-32 添加 ColorInversionControllerImpl.java	22
图 1-33 添加状态类图标到 PhoneStatusBarpolicy 中（上）	23
图 1-34 添加状态类图标到 PhoneStatusBarpolicy 中（下）	23
图 1-35 创建 ColorInversionController 对象	24
图 1-36 修改 StatusBarIconview.java	24
图 1-37 修改 StatusBarIconController.java.....	25
图 1-38 核心代码修改	26
图 1-39 WMS 模块修改.....	27
图 2-1 QS 初始化流程	28
图 2-2 QS 场景定义	29
图 2-3 QS 默认界面半展开形态	30
图 2-4 QS 默认场景完全展开形态	31
图 2-5 edit 点击事件处理	32
图 2-6 编辑界面显示效果	32
图 2-7 QSFragment 类图结构	34
图 2-8 QSPanel 和 QuickQSPanel 类图关系.....	35
图 2-9 QSTileView 类图关系	36
图 2-10 QSTile 点击流程.....	37
图 2-11 亮度调节初始化流程	38
图 2-12 滑动调节亮度的流程	39
图 2-13 添加 TileService 服务	40
图 2-14 label 下显示应用名称.....	41
图 2-15 添加 Imageview.....	42

图 2-16 修改 BrightnessMirrorController	42
图 2-17 修改 BrightnessController.....	42
图 2-18 onTuningChanged 初始化代码.....	43
图 2-19 配置 quick_settings_tiles_default	43
图 2-20 快捷项设置	44

Unisoc Confidential For hiar

表目录

表 1-1 keyguard_status_bar.xml 中的子控件	2
表 1-2 status_bar.xml 中的子控件	3

Unisoc Confidential For hiar

1 状态栏

本章节主要介绍状态栏在各个模式下的状态、主要控件、功能以及状态栏的客制化功能。

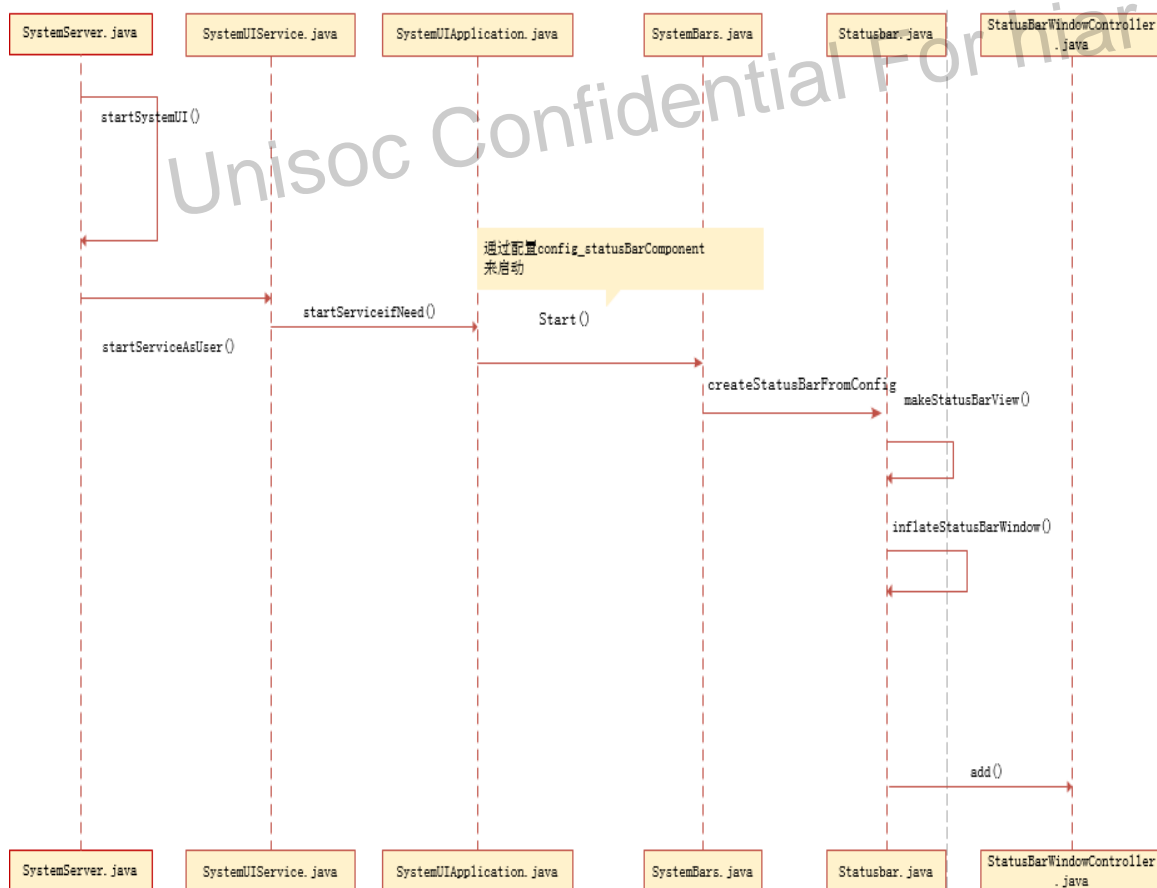
1.1 介绍

状态栏是 SystemUI 中的重要功能之一，用于显示功能图标和手机基本信息。在 Android 10.0 版本中，功能图标和基本信息主要包含：

- 功能图标：SIM 卡信息、通知图标、系统图标、WiFi 图标等。
- 基本信息：电池信息、时间、日期等。

状态栏的启动过程大致如图 1-1 所示。

图1-1 状态栏的启动过程



以上就是整个状态栏启动过程。系统在 StatusBar.java 中获取布局后 add 到 window 中，其中布局的初始在 SuperStatusBarViewFactory.java 中进行，布局名称为 super_status_bar.xml。根布局名称是 StatusBarWindowView，对应的子布局是 status_bar_container，即 statusbar 在解锁后的布局文件，锁屏状态下的布局文件名为 keyguard_status_bar.xml。

1.1.1 锁屏状态栏

只有解锁方式设置为“滑动”或者“其他的安全锁”时才会显示锁屏状态栏。当锁屏方式设置为“无”时状态栏不会显示，只有按下 power 键或者自动息屏才会显示相应的锁屏界面。显示逻辑主要在 NotificationPanelViewController.java 中进行。锁屏状态下显示的状态栏如图 1-2 所示。

图1-2 锁屏状态下显示的状态栏



锁屏状态下的状态栏显示效果分为三部分，分别由三个子控件控制，详细信息如表 1-1 所示。

表1-1 keyguard_status_bar.xml 中的子控件

编号	子控件名称	ID
1	keyguard_status_bar.xml	keyguard_carrier_text
2	keyguard_status_bar.xml	cutout_space_view
3	keyguard_status_bar.xml	system_icons_container

锁屏状态下的状态栏显示效果都在 keyguard_status_bar.xml 布局中，布局路径为

/frameworks/base/packages/SystemUI/res/layout/keyguard_status_bar.xml。

其中 cutout_space_view 控件大小的计算在 updateLayoutParamsNoCutout()方法中进行，如图 1-3 所示。

图1-3 updateLayoutParamsNoCutout()

```
private boolean updateLayoutParamsNoCutout() {
    if (mLayoutState == LAYOUT_NO_CUTOUT) {
        return false;
    }
    mLayoutState = LAYOUT_NO_CUTOUT;

    if (mCutoutSpace != null) {
        mCutoutSpace.setVisibility(View.GONE);
    }

    RelativeLayout.LayoutParams lp = (LayoutParams) mCarrierLabel.getLayoutParams();
    lp.addRule(RelativeLayout.START_OF, R.id.status_icon_area);

    lp = (LayoutParams) mStatusIconArea.getLayoutParams();
    lp.removeRule(RelativeLayout.END_OF);
    lp.width = LayoutParams.WRAP_CONTENT;

    LinearLayout.LayoutParams llp =
        (LinearLayout.LayoutParams) mSystemIconsContainer.getLayoutParams();
    llp.setMarginStart(getResources().getDimensionPixelSize(
        R.dimen.system_icons_super_container_margin_start));
    return true;
}
```

1.1.2 解锁状态栏

解锁后显示的状态栏与锁屏下显示的状态栏不一致，具体的显示效果如图 1-4 所示。

图1-4 解锁后显示的状态栏



解锁状态下的状态栏显示效果分为三部分，分别由三个子控件控制，详细信息如表 1-2 所示。

表1-2 status_bar.xml 中的子控件

编号	子控件名称	ID
1	status_bar.xml 子控件	status_bar_left_side
2	status_bar.xml 子控件	cutout_space_view
3	status_bar.xml 子控件	system_icon_area

解锁状态下的状态栏显示效果都在 status_bar.xml 布局中，status_bar.xml 的根布局名称是 PhoneStatusBarView。布局代码路径为 frameworks/base/packages/SystemUI/res/layout/status_bar.xml，布局代码如图 1-5 所示。

图1-5 status_bar.xml 布局代码

```

<com.android.systemui.statusbar.phone.PhoneStatusBarView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:systemui="http://schemas.android.com/apk/res/com.android.systemui"
    android:layout_width="match_parent"
    android:layout_height="@dimen/status_bar_height"
    android:id="@+id/status_bar"
    android:background="@drawable/system_bar_background"
    android:orientation="vertical"
    android:focusable="false"
    android:descendantFocusability="afterDescendants"
    android:accessibilityPaneTitle="@string/status_bar"
>

    <ImageView
        android:id="@+id/notification_lights_out"
        android:layout_width="@dimen/status_bar_icon_size"
        android:layout_height="match_parent"
        android:paddingStart="@dimen/status_bar_padding_start"
        android:paddingBottom="2dip"
        android:src="@drawable/ic_sysbar_lights_out_dot_small"
        android:scaleType="center"
        android:visibility="gone"
    />

    <LinearLayout android:id="@+id/status_bar_contents"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingStart="@dimen/status_bar_padding_start"
        android:paddingEnd="@dimen/status_bar_padding_end"
        android:orientation="horizontal"
    >

        <ViewStub
            android:id="@+id/operator_name"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout="@layout/operator_name" />

        <FrameLayout
            android:layout_height="match_parent"
            android:layout_width="0dp"
            android:layout_weight="1">

            <include layout="@layout/heads_up_status_bar_layout" />

            <!-- The alpha of the left side is controlled by PhoneStatusBarTransitions, and the
                individual views are controlled by StatusBarManager disable flags DISABLE_CLOCK and
                DISABLE_NOTIFICATION_ICONS, respectively -->
            <LinearLayout
                android:id="@+id/status_bar_left_side"
                android:layout_height="match_parent"
                android:layout_width="match_parent"
                android:clipChildren="false"
                android:orientation="horizontal"
            >

                <com.android.systemui.statusbar.policy.Clock
                    android:id="@+id/clock"
                    android:layout_width="wrap_content"
                    android:layout_height="match_parent"
                    android:textAppearance="@style/TextAppearance.StatusBar.Clock"
                    android:singleLine="true"
                    android:paddingStart="@dimen/status_bar_left_clock_starting_padding"
                    android:paddingEnd="@dimen/status_bar_left_clock_end_padding"
                    android:gravity="center_vertical|start"
                />

                <com.android.systemui.statusbar.AlphaOptimizedFrameLayout
                    android:id="@+id/notification_icon_area"
                    android:layout_width="0dp"
                    android:layout_height="match_parent"
                    android:layout_weight="1"
                    android:orientation="horizontal"
                    android:clipChildren="false"/>

            </LinearLayout>
        </FrameLayout>

        <!-- Space should cover the notch (if it exists) and let other views lay out around it -->
        <android.widget.Space
            android:id="@+id/cutout_space_view"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:gravity="center_horizontal|center_vertical"
        />

        <com.android.keyguard.AlphaOptimizedLinearLayout android:id="@+id/system_icon_area"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:orientation="horizontal"
            android:gravity="center_vertical|end"
        >

            <include layout="@layout/system_icons" />
        </com.android.keyguard.AlphaOptimizedLinearLayout>
    </LinearLayout>

    <ViewStub
        android:id="@+id/emergency_cryptkeeper_text"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_marginLeft="63dp"
        android:layout="@layout/emergency_cryptkeeper_text"
    />
</com.android.systemui.statusbar.phone.PhoneStatusBarView>

```

当存在活跃通知，且view的flag设置有STEP_UI_FLAG_LOW_PROFILE属性时会显示一个点，提示全屏用于当前存在通知

状态栏显示运营商信息，需设置config_showOperatorNameInStatusBar为true

悬浮式通知布局

时钟图标

通知图标区域

如果是刘海项目，会在updateLayoutForCutout方法里设置区域范围

系统图标区域，主要是包含了system_icons.xml，和锁屏状态栏系统图标区域一样

显示界面信息

1.1.3 下拉状态栏

下拉状态栏是在用户下拉出状态栏时显示的界面，界面显示效果如图 1-6 所示：

图1-6 下拉状态栏界面



下拉状态栏界面显示由两个布局文件控制：

- quick_status_bar_header_system_icons 布局：显示时钟和电池的布局，路径为 `/frameworks/base/packages/SystemUI/res/layout/quick_status_bar_header_system_icons.xml`。
- quick_qs_status_icons 布局：显示日期和系统图标的布局，路径为 `frameworks/base/packages/SystemUI/res/layout/quick_qs_status_icons.xml`。

这两个布局均由 `QuickStatusBarHeader.java` 控制，路径为 `frameworks/base/packages/SystemUI/src/com/android/systemui/qs/QuickStatusBarHeader.java`。

1.2 主要控件

状态栏的主要控件有 6 个：

- 运营商信息
- 时间和日期
- 电池
- 系统图标
- 通知图标
- 图标容器

1.2.1 运营商信息

运营商信息控件有两个：

- `CarrierText`：监听 `CarrierTextController`，封装了 `KeyguardUpdateMonitorCallback` 接口。路径为 `frameworks/base/packages/SystemUI/src/com/android/keyguard/CarrierText.java`。
- `OperatorNameView`：实现了 `KeyguardUpdateMonitorCallback` 接口，用于监听 sim 卡信息的变化并更新运营商信息。路径为：
`frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/OperatorNameView.java`。

说明

这两个控件的区别在于，CarrierText 显示所有状态下的运营商信息，而 OperatorNameView 只显示能正常时的运营商信息。

1.2.2 时间和日期

时间和日期分别由两个不同的控件控制：

- Clock：时间控件，路径为
/frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy/Clock.java。
- DateView：日期控件，路径为
/frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy/DateView.java。

说明

DateView 的日期格式不受 Settings 中的日期格式控制，如需修改可以在 quick_qs_status_icons.xml 中的 com.android.systemui.statusbar.policy.DateView 内添加 systemui:datePattern 属性，然后自定义时间格式。

1.2.3 电池

电池由 BatteryMeterView 控制，路径为

/frameworks/base/packages/SystemUI/src/com/android/systemui/BatteryMeterView.java。此控件平台有相应的客制化实现，主要用于实现充电动画功能，此功能通过 config_battery_animation 属性来控制。

电池百分比显示功能有两种方式控制：

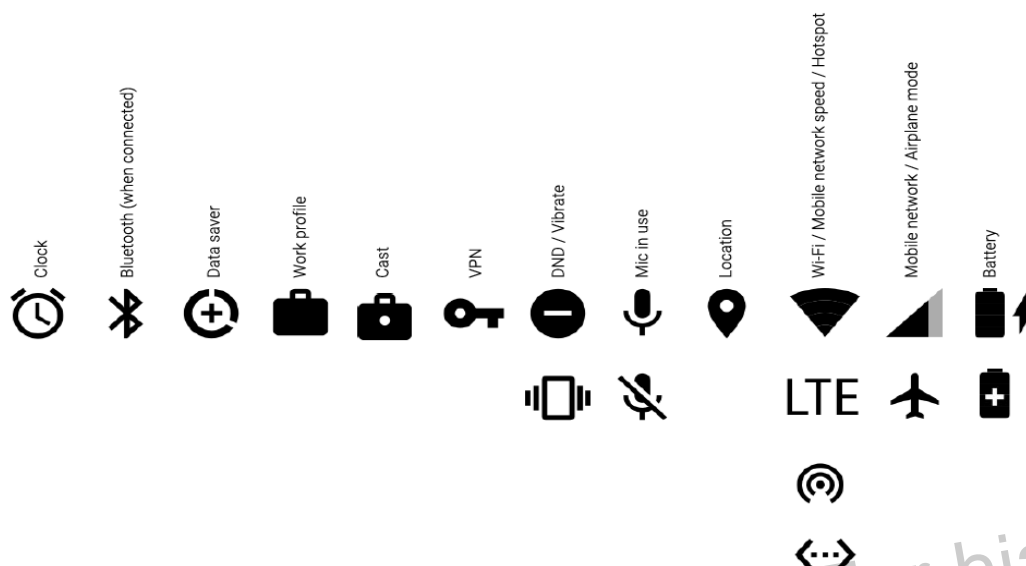
- 调用 setForceShowPercent() 方法来强制显示百分比，这就是锁屏状态栏显示百分比使用的方式。
- 设置 Setting.System 数据库里的 SHOW_BATTERY_PERCENT 值来显示电池百分比，这个是在 Settings 设置显示百分比的方式。

1.2.4 系统图标

设计和定义

Android 10.0 延续了 Android 9.0 中的系统图标设计，系统中包含的图标如图 1-7 所示。

图1-7 系统图标



这些图标都定义在了 `/frameworks/base/core/res/res/values/config.xml` 中的 `config_statusBarIcons` 里，如图 1-8 所示：

图1-8 系统图标的定义代码

```
29 <string-array name="config_statusBarIcons">
30 <item><xliff:g id="id">@string/status_bar_alarm_clock</xliff:g></item>
31 <item><xliff:g id="id">@string/status_bar_rotate</xliff:g></item>
32 <item><xliff:g id="id">@string/status_bar_headset</xliff:g></item>
33 <item><xliff:g id="id">@string/status_bar_data_saver</xliff:g></item>
34 <item><xliff:g id="id">@string/status_bar_ime</xliff:g></item>
35 <item><xliff:g id="id">@string/status_bar_sync_failing</xliff:g></item>
36 <item><xliff:g id="id">@string/status_bar_sync_active</xliff:g></item>
37 <item><xliff:g id="id">@string/status_bar_nfc</xliff:g></item>
38 <item><xliff:g id="id">@string/status_bar_tty</xliff:g></item>
39 <item><xliff:g id="id">@string/status_bar_speakerphone</xliff:g></item>
40 <item><xliff:g id="id">@string/status_bar_cdma_eri</xliff:g></item>
41 <item><xliff:g id="id">@string/status_bar_data_connection</xliff:g></item>
42 <item><xliff:g id="id">@string/status_bar_phone_evdo_signal</xliff:g></item>
43 <item><xliff:g id="id">@string/status_bar_phone_signal</xliff:g></item>
44 <item><xliff:g id="id">@string/status_bar_secure</xliff:g></item>
45 <item><xliff:g id="id">@string/status_bar_managed_profile</xliff:g></item>
46 <item><xliff:g id="id">@string/status_bar_cast</xliff:g></item>
47 <item><xliff:g id="id">@string/status_bar_screen_record</xliff:g></item>
48 <item><xliff:g id="id">@string/status_bar_vpn</xliff:g></item>
49 <item><xliff:g id="id">@string/status_bar_bluetooth</xliff:g></item>
50 <item><xliff:g id="id">@string/status_bar_camera</xliff:g></item>
51 <item><xliff:g id="id">@string/status_bar_microphone</xliff:g></item>
52 <item><xliff:g id="id">@string/status_bar_location</xliff:g></item>
53 <item><xliff:g id="id">@string/status_bar_mute</xliff:g></item>
54 <item><xliff:g id="id">@string/status_bar_volume</xliff:g></item>
55 <item><xliff:g id="id">@string/status_bar_zen</xliff:g></item>
56 <item><xliff:g id="id">@string/status_bar_ethernet</xliff:g></item>
57 <item><xliff:g id="id">@string/status_bar_vowifi</xliff:g></item>
58 <item><xliff:g id="id">@string/status_bar_wifi</xliff:g></item>
59 <item><xliff:g id="id">@string/status_bar_hotspot</xliff:g></item>
60 <item><xliff:g id="id">@string/status_bar_hdvoice</xliff:g></item>
61 <item><xliff:g id="id">@string/status_bar_mobile</xliff:g></item>
62 <item><xliff:g id="id">@string/status_bar_airplane</xliff:g></item>
63 <item><xliff:g id="id">@string/status_bar_battery</xliff:g></item>
64 <item><xliff:g id="id">@string/status_bar_sensors_off</xliff:g></item>
65 </string-array>
```

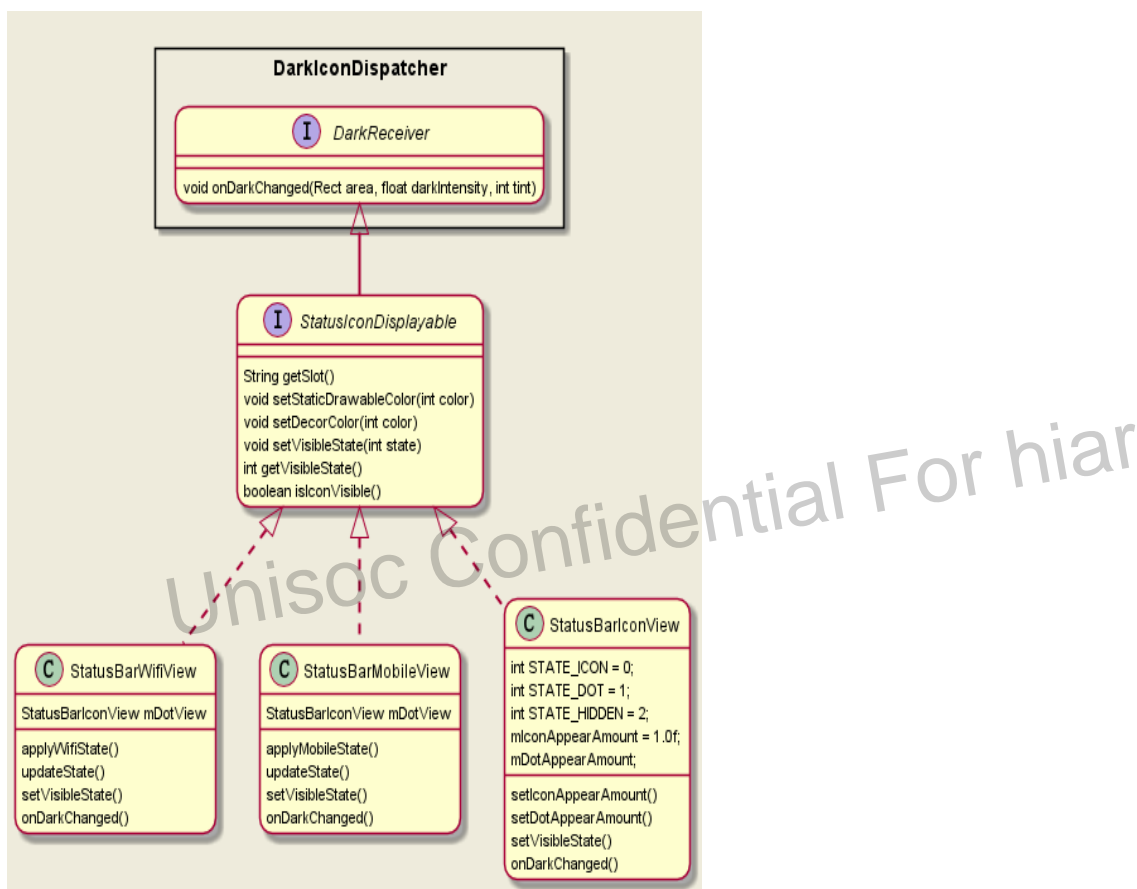

图标控件

Android 10.0 版本里的系统图标控件分为三种：

- StatusBarWifiView：WiFi 图标
- StatusBarMobileView：信号图标
- StatusBarIconView：其他图标，如蓝牙、热点、闹钟等

系统图标关系如图 1-9 所示。

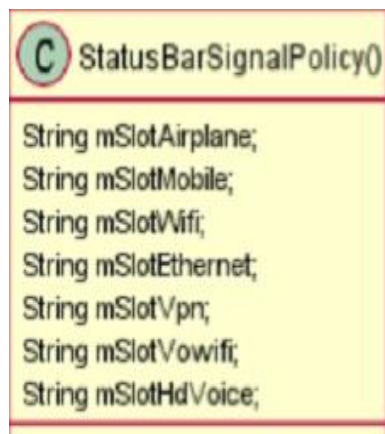
图1-9 系统图标关系



系统图标显示过程不同于其他状态栏图标。系统图标只有在某些功能发现变化时才会显示或者更新状态栏图标。其更新控制集中在 StatusBarSignalPolicy.java 和 PhoneStatusBarPolicy.java 中。

StatusBarSignalPolicy 主要控制信号类系统图标，如图 1-10 所示。

图1-10 StatusBarSignalPolicy 控制信号类系统图标



PhoneStatusBarPolicy 主要控制状态类系统图标，如图 1-11 所示。

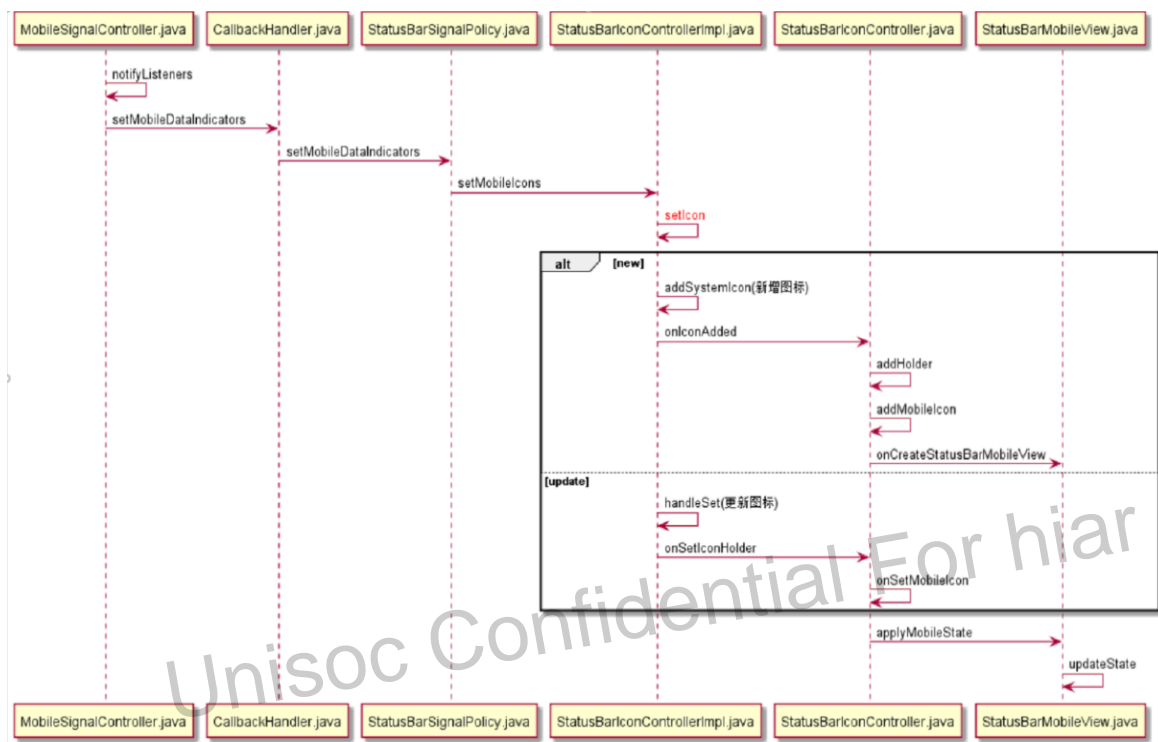
图1-11 PhoneStatusBarPolicy 控制状态类系统图标



显示流程

系统图标的显示流程基本类似，只是功能控制类不同。都是通过各自的 Controller 类监听功能状态，再通过 Callback 接口调 Policy 类，然后通过 IconController 来刷新图标，mobile 图标大致更新过程如图 1-12 所示。

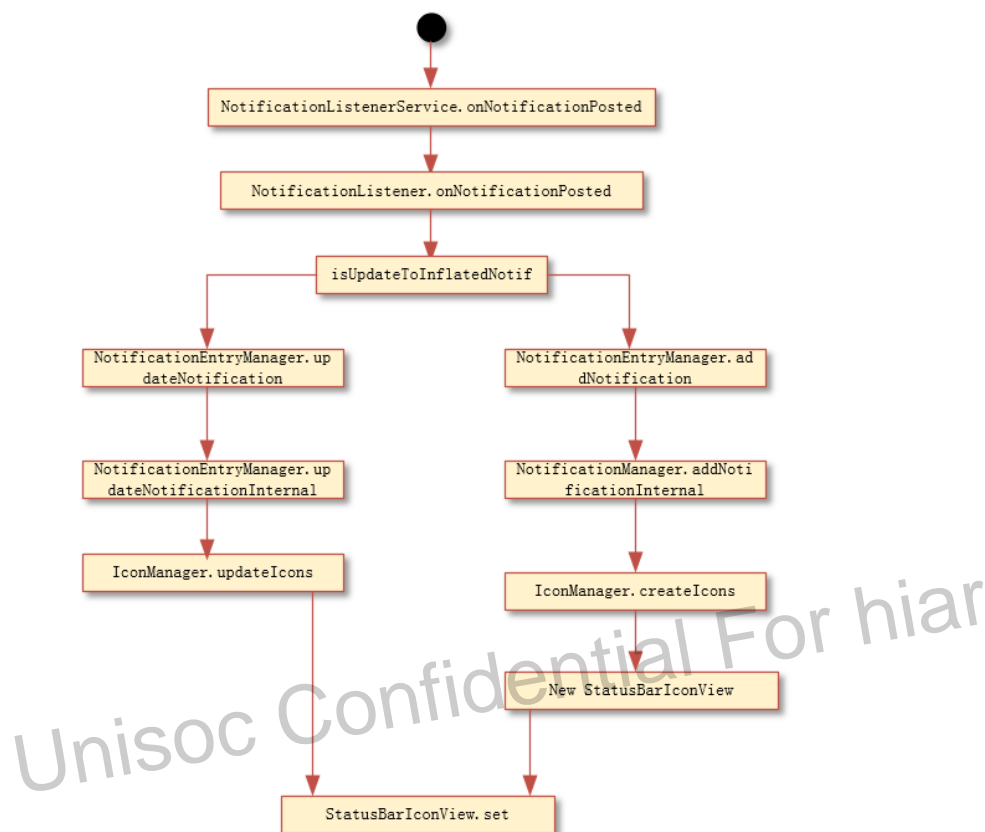
图1-12 Mobile 图标更新过程



1.2.5 通知图标

通知图标也是由 StatusBarIconView 控件控制，其创建和更新流程如图 1-13 所示。

图1-13 通知图标的创建和更新流程



1.2.6 图标容器

Android 10.0 中有两个容器控件：

- StatusIconContainer：系统图标容器控件，是系统图标的父类控件。
- NotificationIconContainer：通知图标容器控件，是通知图标的父类控件。

用户根据当前容器控件的大小来控制子控件的显示状态：

- STATE_ICON：0
- STATE_HIDDEN：1
- STATE_DOT：2

说明

0 代表显示图标，1 代表隐藏图标，2 代表隐藏图标但显示为圆点。

StatusIconContainer 控制子控件显示状态的核心方法是 calculateIconTranslations，详细操作步骤如下。

NotificationIconContainer 显示逻辑与之基本类似。

1. 按从后到前的顺序计算当前需要显示的图标数，并计算剩余的空间 translationx。如图 1-14 所示。

图1-14 计算图标数量和剩余空间

```
/**
 * Layout is happening from end -> start
 */
private void calculateIconTranslations() {
    mLayoutStates.clear();
    float width = getWidth(); //控件宽度
    float translationX = width - getPaddingEnd(); //除去paddingEnd的宽度
    float contentStart = getPaddingStart();
    int childCount = getChildCount();
    // Underflow == don't show content until that index
    if (DEBUG) android.util.Log.d(TAG, "calculateIconTranslations: start=" + translationX
        + " width=" + width + " underflow=" + mNeedsUnderflow);

    // Collect all of the states which want to be visible
    for (int i = childCount - 1; i >= 0; i--) {
        View child = getChildAt(i);
        StatusIconDisplayable iconView = (StatusIconDisplayable) child;
        StatusIconState childState = getViewStateFromChild(child);

        if (!iconView.isIconVisible() || iconView.isIconBlocked()
            || mIgnoredSlots.contains(iconView.getSlot())) {
            childState.visibleState = STATE_HIDDEN;
            if (DEBUG) Log.d(TAG, "skipping child (" + iconView.getSlot() + ") not visible");
            continue;
        }

        // Move translationX to the spot within StatusIconContainer's layout to add the view
        // without cutting off the child view.
        translationX -= getViewTotalWidth(child);
        childState.visibleState = STATE_ICON;
        childState.xTranslation = translationX;
        mLayoutStates.add(0, childState);

        // Shift translationX over by mIconSpacing for the next view.
        translationX -= mIconSpacing;
    }

    // Show either 1-MAX_ICONS icons, or (MAX_ICONS - 1) icons + overflow
    int totalVisible = mLayoutStates.size();
    int maxVisible = totalVisible <= MAX_ICONS ? MAX_ICONS : MAX_ICONS - 1;
}
```

按从后到前的顺序计算当前需要显示的图标数，并计算剩余的空间translationx

2. 通过 state.xTranslation 计算第一个无法显示的图标 index。如图 1-15 所示。

图1-15 计算第一个无法显示的图标 index

```
// Show either 1-MAX_ICONS icons, or (MAX_ICONS - 1) icons + overflow
int totalVisible = mLayoutStates.size();
int maxVisible = totalVisible <= MAX_ICONS ? MAX_ICONS : MAX_ICONS - 1;

mUnderflowStart = 0;
int visible = 0;
int firstUnderflowIndex = -1;
for (int i = totalVisible - 1; i >= 0; i--) {
    StatusIconState state = mLayoutStates.get(i);
    // Allow room for underflow if we found we need it in onMeasure
    if (mNeedsUnderflow && (state.xTranslation < (contentStart + mUnderflowWidth)) ||
        (mShouldRestrictIcons && visible >= maxVisible)) {
        firstUnderflowIndex = i;
        break;
    }
    mUnderflowStart = (int) Math.max(
        contentStart, state.xTranslation - mUnderflowWidth - mIconSpacing);
    visible++;
}
```

通过 state.xTranslation 计算第一个无法显示的图标index

3. 通过 firstUnderflowIndex 将第一个无法显示的图标设置为 STATE_ICON，并将其前面的图标设置为 STATE_HIDDEN。如图 1-16 所示。

图1-16 通过 firstUnderflowIndex 进行设置

```
if (firstUnderflowIndex != -1) {
    int totalDots = 0;
    int dotWidth = mStaticDotDiameter + mDotPadding;
    int dotOffset = mUnderflowStart + mUnderflowWidth - mIconDotFrameWidth;
    for (int i = firstUnderflowIndex; i >= 0; i--) {
        StatusIconState state = mLayoutStates.get(i);
        if (totalDots < MAX_DOTS) {
            state.xTranslation = dotOffset;
            state.visibleState = STATE_DOT;
            dotOffset -= dotWidth;
            totalDots++;
        } else {
            state.visibleState = STATE_HIDDEN;
        }
    }
}
```

通过firstUnderflowIndex将第一个无法显示的图标设置为STATE_ICON，并将其前面的图标设置为STATE_HIDDEN

1.3 主要功能

状态栏的主要功能有：

- 多图标成点
- 图标反色
- StatusBar 服务
- 刘海屏

1.3.1 多图标成点

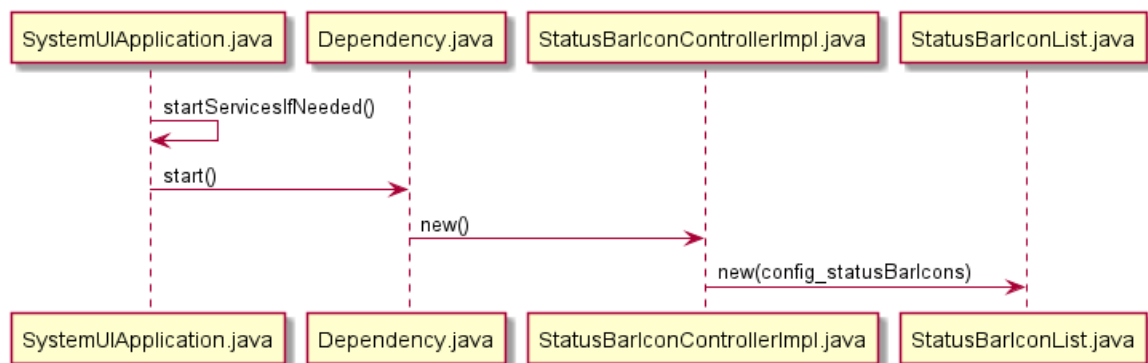
当系统图标区域或者通知图标区域图标过多，区域显示不够用时，状态栏就会将多余的图标隐藏，并显示一个点，如图 1-17 所示。

图1-17 状态栏多图标成点



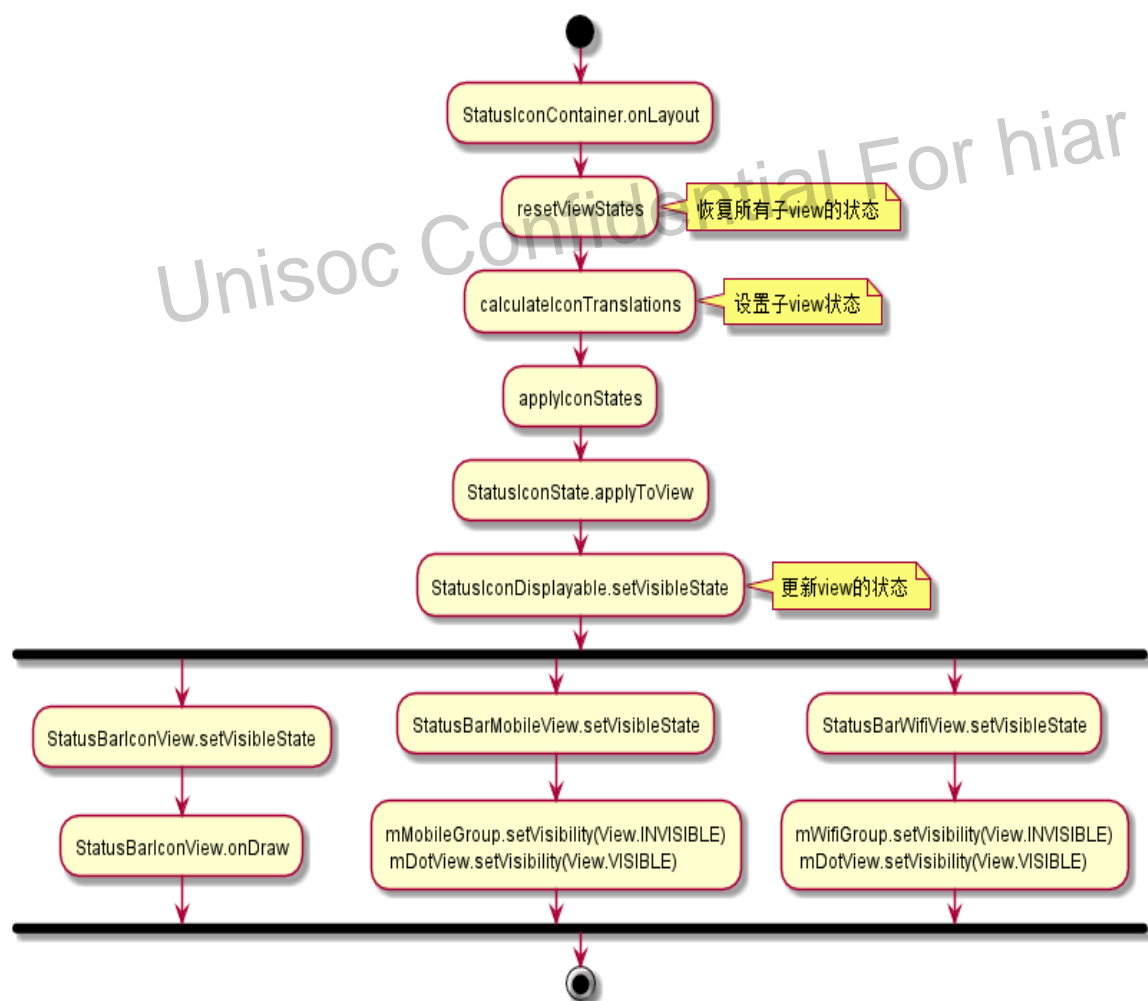
介绍此功能之前先看一下图标的初始化过程，如图 1-18 所示。从中可以看到有个 config_statusBarIcons 参数，该参数定义了所有系统图标由低到高显示的优先级顺序。

图1-18 状态栏图标初始化过程



点图标是通过设置 STATE_DOT 来实现的，在状态更新后会 applyToView，然后在图标类 SetVisivleState 中刷新图标，流程如图 1-19 所示。

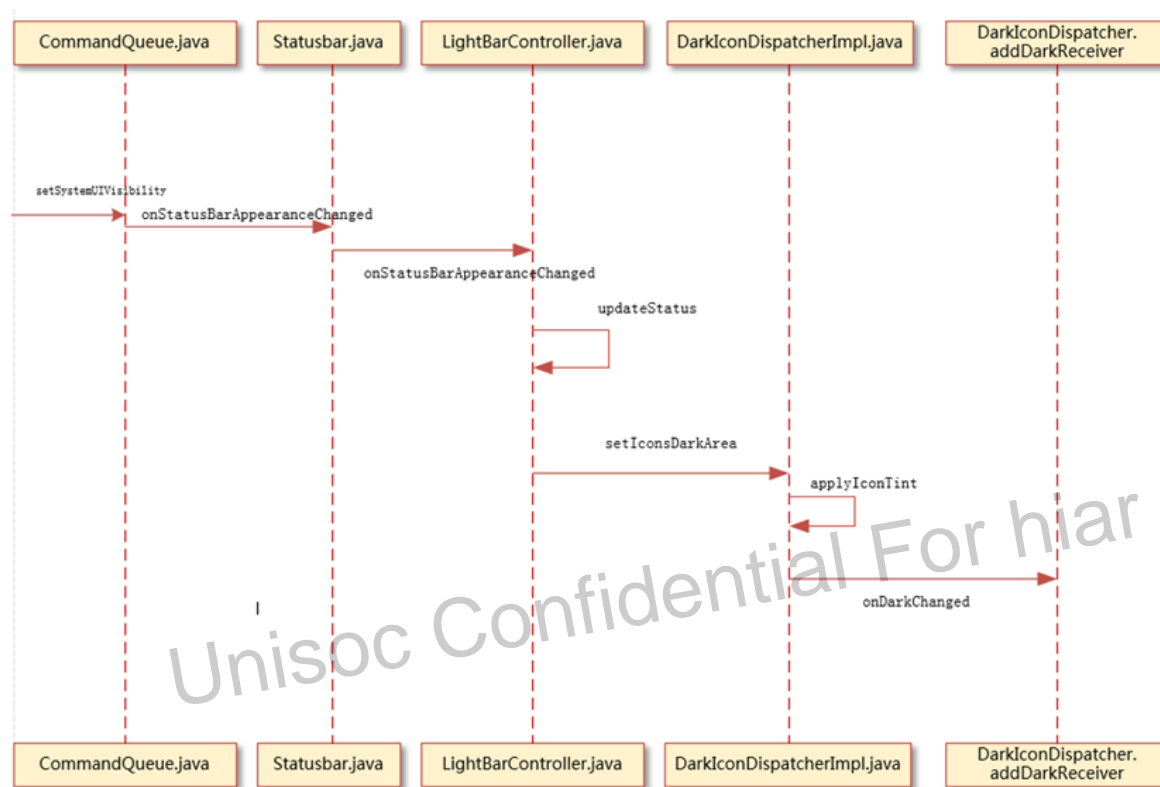
图1-19 状态栏图标刷新流程



1.3.2 图标反色

图标反色指当状态栏背景颜色为浅色时，状态栏图标会显示成深色，背景颜色为深色时，图标颜色会变成浅色。从图 1-9 系统图标关系的类图中可以看到，状态栏所有图标类都实现了 `onDarkChanged` 方法，此方法用于设置图标颜色，具体过程如图 1-20 所示。

图1-20 `onDarkChanged` 方法设置图标颜色



1.3.3 StatusBar 服务

在使用部分应用时，经常发现状态栏不显示部分应用图标，这部分功能是通过 StatusBar 服务实现。该服务是系统服务，并不对外开放，只有系统应用可以直接调用（第三方应用很多是通过反射方式实现）。

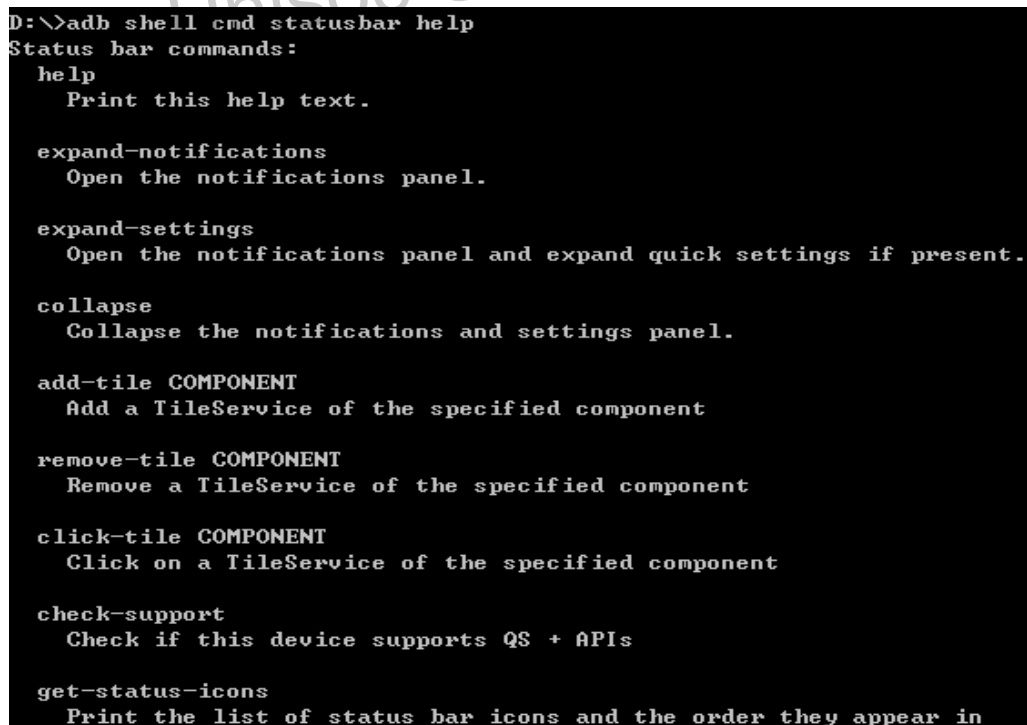
StatusBar 服务在 SysemServer.java 中启动，如图 1-21 所示。

图1-21 启动 StatusBar 服务

```
if (!isWatch) {
    t.traceBegin("StartStatusBarManagerService");
    try {
        statusBar = new StatusBarManagerService(context);
        ServiceManager.addService(Context.STATUS_BAR_SERVICE, statusBar);
    } catch (Throwable e) {
        reportWtf("starting StatusBarManagerService", e);
    }
    t.traceEnd();
}
```

StatusBar 服务实现了很多常用的功能，方便用户对状态栏进行操作。可以在 IStatusBarService.aidl 中了解定义的方法。同时 Android 10.0 提供了 adb 指令对状态栏进行操作，通过 adb shell cmd statusbar help 可以查看当前支持的 adb 功能以及功能说明，如图 1-22 所示。

图1-22 adb shell cmd statusbar help



```
D:\>adb shell cmd statusbar help
Status bar commands:
  help
    Print this help text.

  expand-notifications
    Open the notifications panel.

  expand-settings
    Open the notifications panel and expand quick settings if present.

  collapse
    Collapse the notifications and settings panel.

  add-tile COMPONENT
    Add a TileService of the specified component

  remove-tile COMPONENT
    Remove a TileService of the specified component

  click-tile COMPONENT
    Click on a TileService of the specified component

  check-support
    Check if this device supports QS + APIs

  get-status-icons
    Print the list of status bar icons and the order they appear in
```

1.3.4 刘海屏

刘海功能

该功能是 Android 9.0 上新增的功能，在开发者模式中提供了模拟刘海屏设置，路径为 frameworks/base/core/java/android/view/DisplayCutout.java。通过配置核心变量 config_mainBuiltInDisplayCutout 值来绘制刘海区域。这里以如何把宽为 308px，高为 55px 的 png 格式图片转换到手机上并显示刘海屏效果为例，配置步骤如下。

步骤 1 转换 png 格式图片为 svg 格式图片。利用在线工具 vectorizer 把 png 图转化为 svg 图，并下载。转换出来的数据（用 notepad 即可查看）如图 1-23 所示。

图1-23 图片转换后的数据

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"><svg
version="1.0" xmlns="http://www.w3.org/2000/svg" width="308px" height="55px"
viewBox="0 0 308 55" preserveAspectRatio="xMidYMid meet"><g id="layer101"
fill="#000000" stroke="none"><path d="M369 507 c-80 -33 -124 -62 -163 -109 -
51 -59 -86 -141 -95 -223 -11 -91 -31 -130 -76 -146 -19 -7 -35 -17 -35 -21 0 -4 693 -
8 1541 -8 1024 0 1538 3 1534 10 -3 5 -13 10 -21 10 -28 0 -70 29 -83 57 -7 15 -17
60 -21 99 -17 141 -92 249 -218 311 l-76 38 -1115 2 -1116 3 -56 -23z"/>
</g></svg>
```

步骤 2 用 svg2android-gh-pages 工具把 svg 生成矢量图 xml（Svg 生成 Vector，也可以用 Android Studio 工具生成），内容如图 1-24 所示。

图1-24 生成矢量图 xml

```
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="3080dp"
    android:height="550dp"
    android:viewportWidth="3080"
    android:viewportHeight="550">
    <path
        android:fillColor="#000000"
        android:pathData="M369 507 c-80 -33 -124 -62 -163 -109 -51 -59 -86 -141 -95 -223 -11 -91 -31 -130 -76 -146 -19 -7 -35 -17 -35 -21 0 -4 693 -8 1541 -8 1024 0 1538 3 1534 10 -3 5 -13 10 -21 10 -28 0 -70 29 -83 57 -7 15 -17 60 -21 99 -17 141 -92 249 -218 311 l-76 38 -1115 2 -1116 3 -56 -23z" />
```

步骤 3 取出 pathData 替换到 config_mainBuiltInDisplayCutout，保留 “@dp”，如图 1-25 所示。

图1-25 替换 pathData

```
<string translatable="false" name="config_mainBuiltInDisplayCutout">
    M369 507 c-80 -33 -124 -62 -163 -109 -51 -59 -86 -141 -95 -223 -11 -91 -31 -
130
-76 -146 -19 -7 -35 -17 -35 -21 0 -4 693 -8 1541 -8 1024 0 1538 3 1534 10 -3 5
-13 10 -21 10 -28 0 -70 29 -83 57 -7 15 -17 60 -21 99 -17 141 -92 249 -218 311
l-76 38 -1115 2 -1116 3 -56 -23z
    @dp
</string>
```

说明

“@dp”、“@bottom”、“@right”的使用，代码中有相关解析判断。路径中指定“@dp”以便模拟针对不同设备的刘海屏形状。由于实际的刘海屏具有精确的像素尺寸（不应该跟随 density 变化），因此在定义硬件刘海屏的路径时，请勿使用“@dp”指定符。

步骤 4 把 config_mainBuiltInDisplayCutout 所有取值缩小 20 倍（根据机器的 density 不同，需要缩小的倍数也不同），如图 1-26 所示。

图1-26 缩小取值

```
<string translatable="false" name="config_mainBuiltInDisplayCutout">
M 18.45 25.35
c-4.0 -1.65 -6.2 -3.1 -8.15 -5.45
-2.55 -2.95 -4.3 -7.05 -4.75 -11.15
-0.55 -4.55 -1.55 -6.5 -3.8 -7.3
-0.95 -0.35 -1.75 -0.85 -1.75 -1.05
0.0 -0.2 34.65 -0.4 77.05 -0.4
50.2 0.0 76.9 0.15 76.7 0.5
-0.15 0.25 -0.65 0.5 -1.05 0.5
-1.4 0.0 -3.5 1.45 -4.15 2.85
-0.35 0.75 -0.85 3.0 -1.05 4.95
-0.85 7.05 -4.6 12.45 -10.9 15.55
l-3.8 1.9 -55.75 0.1 -55.8 0.15 -2.8 -1.15z
    @dp
</string>
```

png 图是以 px 为单位 308px * 55px 大小的一张图，转成 Vector xml 后数值会转化为 dp 单位了，且数值放大了 10 倍。因此在使用 dp 单位的数据时应该先缩小 10 倍，而由于最终在机器上是以 px 呈现出来大

小，所以中间涉及到 dp 和 px 的换算，公式为 $px = dip * density / 160$ 。以 density 为 320 举例，要显示对应 px 大小的区域，需要除以 2 才能得到预期效果。

步骤 5 调整初始笔画位置，即 M 指令的 X 值，图中调试时 X 值向左偏移 60（需要根据实际情况调整居中效果），即得到完整的 config，如图 1-27 所示。

图1-27 调整初始笔画位置

```
<string translatable="false" name="">
M -60 25.35
c-4.0 -1.65 -6.2 -3.1 -8.15 -5.45
-2.55 -2.95 -4.3 -7.05 -4.75 -11.15
-0.55 -4.55 -1.55 -6.5 -3.8 -7.3
-0.95 -0.35 -1.75 -0.85 -1.75 -1.05
0.0 -0.2 34.65 -0.4 77.05 -0.4
50.2 0.0 76.9 0.15 76.7 0.5
-0.15 0.25 -0.65 0.5 -1.05 0.5
-1.4 0.0 -3.5 1.45 -4.15 2.85
-0.35 0.75 -0.85 3.0 -1.05 4.95
-0.85 7.05 -4.6 12.45 -10.9 15.55
l-3.8 1.9 -55.75 0.1 -55.8 0.15 -2.8 -1.15z
@dp
</string>
```

X 值的计算：

4. 首先要确认机器的 size 和 density（可通过 adbshell wmsize/density 查看设备参数），假设设备 size: 1080x1920, density: 320。
5. 例如将上面的 config 配置完成后，发现刘海偏右，开机 log 中搜索关键字 boundingRect，如 “cutout DisplayCutout{insets=Rect(0, 0 -0, 0) boundingRect=Rect(400, 0 -688, 144)}”。
6. 将设备想象成坐标轴，X 方向的中心点为（540, 0），那么中心左边宽度为 540-400=140px，右边宽度 688-540=148px，右边比左边多 8px，即需要向左再偏移 4px。
7. 根据 density 转化成 dp 单位，即 X 值需要再向左偏移 2dp 即可（有实体刘海，不以 “@dp” 结尾，直接以 px 偏移数值结尾）。

步骤 6 根目录 make DisplayCutoutEmulationTallOverlay 生成 apk，push 后重启即可看到效果。

----结束

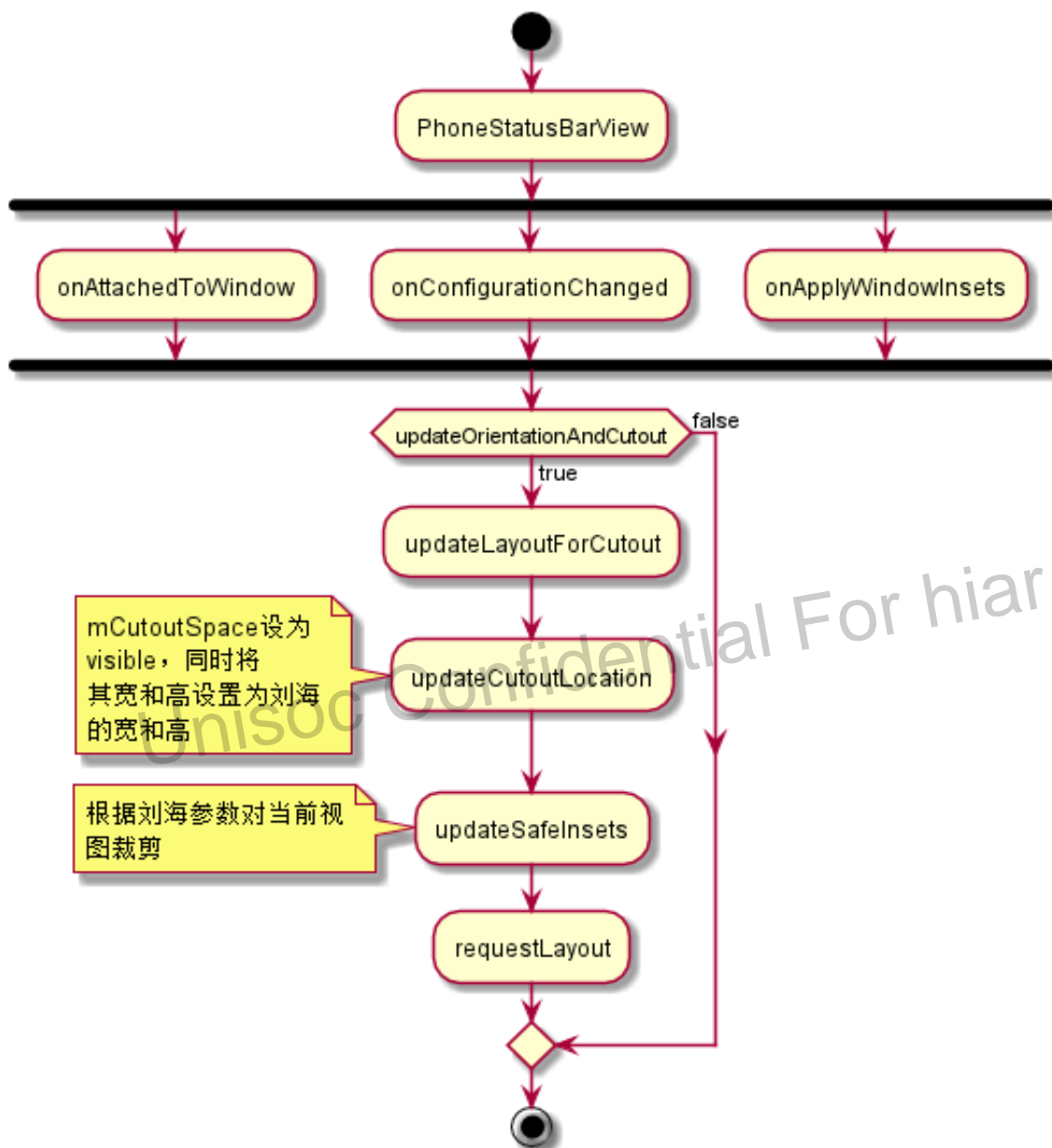
刘海屏显示处理

状态栏场景和锁屏状态栏下会有刘海屏布局，分别对应控件 PhoneStatusBar.java 和

KeyguardStatusBarView.java。PhoneStatusBar 会在 updateCutoutLocation 里对刘海布局 mCutoutSpace 进行

设置，大致流程如图 1-28 所示。其中 `display_cutout_margin_consumption` 参数用以调整刘海的宽度。当刘海外围弧度较大时，图标可以稍微显示到刘海里而不会被遮挡，将状态栏空间使用率最大化。

图1-28 刘海布局设置



`KeyguardStatusBarView` 的逻辑与 `PhoneStatusBarView` 类似，主要是在 `onApplyWindowInsets` 方法中，通过 `updateLayoutConsideringCutout` 来更新刘海屏布局 `mCutoutSpace`。

说明

- `PhoneStatusBarView`: 控制解锁后的状态栏
- `KeyguardStatusBarView`: 控制锁屏下的状态栏

1.4 状态栏客制化

状态栏的客制化功能从新增系统图标和更改系统图标两个方面描述。

1.4.1 新增系统图标

系统图标主要分为两类：

- 状态栏图标
- 信号类图标

首先需要确定添加的图标类别，再开始客制化定制。这里以添加反色状态系统图标为例，步骤如下。

步骤 1 配置 config_statusBarIcons 参数（优先级一般不建议配置太高），如图 1-29 所示。

图1-29 配置 config_statusBarIcons 参数

```
--- a/core/res/res/values/config.xml
+++ b/core/res/res/values/config.xml
@@ -27,6 +27,7 @@
     <!-- Do not translate. Defines the slots for the right-hand side icons. That is to say, the
         icons in the status bar that are not notifications. -->
     <string-array name="config_statusBarIcons">
+
         <item><xliff:g id="id">@string/status_bar_inversion</xliff:g></item>
         <item><xliff:g id="id">@string/status_bar_alarm_clock</xliff:g></item>
         <item><xliff:g id="id">@string/status_bar_rotate</xliff:g></item>
         <item><xliff:g id="id">@string/status_bar_headset</xliff:g></item>
@@ -60,6 +61,7 @@
         <item><xliff:g id="id">@string/status_bar_battery</xliff:g></item>
     </string-array>

+
     <string translatable="false" name="status_bar_inversion">invert_colors</string>
     <string translatable="false" name="status_bar_rotate">rotate</string>
     <string translatable="false" name="status_bar_headset">headset</string>
     <string translatable="false" name="status_bar_data_saver">data_saver</string>
```

步骤 2 添加图片资源文件，如图 1-30 所示。

图1-30 图片资源文件

```
res/drawable/stat_sys_invert_color.xml
```

步骤 3 添加 ColorInversionController.java 和 ColorInversionControllerImpl.java，如图 1-31 和图 1-32 所示。

图1-31 添加 ColorInversionController.java

```
public interface ColorInversionController extends CallbackController<Callback> {  
  
    public interface Callback {  
        void onColorInversion(boolean enable);  
    }  
}
```

图1-32 添加 ColorInversionControllerImpl.java

```
public ColorInversionControllerImpl(Context context) {  
    mContext = context;  
    mContentResolver = mContext.getContentResolver();  
    mContentResolver.registerContentObserver(  
        Secure.getUriFor(Secure.ACCESSIBILITY_DISPLAY_INVERSION_ENABLED), true,  
        mColorInversionModeObserver);  
    onColorInversionModeChanged();  
    if (DEBUG) Log.d(TAG, "new ColorInversionController()");  
}  
  
public void onColorInversionModeChanged(){  
    boolean enable = Secure.getIntForUser(mContentResolver, Secure.ACCESSIBILITY_DISPLAY_INVERSION_ENABLED,  
        0, ActivityManager.getCurrentUser()) != 0;  
    for (ColorInversionController.Callback cb : mCallbacks) {  
        cb.onColorInversionChanged(enable);  
    }  
}  
  
private ContentObserver mColorInversionModeObserver = new ContentObserver(new Handler()) {  
    @Override  
    public void onChange(boolean selfChange) { onColorInversionModeChanged(); }  
}
```


步骤 4 在 Policy 中添加相关图标刷新的逻辑，反色属于状态类图标，故添加到 PhoneStatusBarPolicy 中。
 添加方式如图 1-33 和图 1-34 所示。

图1-33 添加状态类图标到 PhoneStatusBarPolicy 中（上）

```
import com.android.systemui.statusbar.policy.ZenModeController;
+import com.android.systemui.statusbar.policy.ColorInversionController;
import com.android.systemui.util.NotificationChannels;

import java.util.List;
@@ -99,11 +100,13 @@ import java.util.Locale;
*/
public class PhoneStatusBarPolicy implements Callback, Callbacks,
    RotationLockControllerCallback, Listener, LocationChangeCallback,
    ZenModeController.Callback, DeviceProvisionedListener, KeyguardMonitor.Callback {
+    ZenModeController.Callback, DeviceProvisionedListener, KeyguardMonitor.Callback, ColorInversionController.Callback {
    private static final String TAG = "PhoneStatusBarPolicy";
    private static final boolean DEBUG = Log.isLoggable(TAG, Log.DEBUG);

    public static final int LOCATION_STATUS_ICON_ID = R.drawable.stat_sys_location;
+    public static final int Color_INVERT_STATUS_ICON_ID = R.drawable.stat_sys_invert_color;
+
    public static final int NUM_TASKS_FOR_INSTANT_APP_INFO = 5;

    private final String mSlotCast;
@@ -118,6 +121,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    private final String mSlotHeadset;
    private final String mSlotDataSaver;
    private final String mSlotLocation;
+    private final String mSlotColorInversion;

    private final Context mContext;
    private final Handler mHandler = new Handler();
@@ -134,6 +138,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    private final DeviceProvisionedController mProvisionedController;
    private final KeyguardMonitor mKeyguardMonitor;
    private final LocationController mLocationController;
+    private final ColorInversionController mColorInversionController;
    private final ArraySet<Pair<String, Integer>> mCurrentNotifs = new ArraySet<>();
    private final UiOffloadThread mUiOffloadThread = Dependency.get(UiOffloadThread.class);

@@ -167,6 +172,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    mProvisionedController = Dependency.get(DeviceProvisionedController.class);
    mKeyguardMonitor = Dependency.get(KeyguardMonitor.class);
    mLocationController = Dependency.get(LocationController.class);
+    mColorInversionController = Dependency.get(ColorInversionController.class);

    mSlotCast = context.getString(com.android.internal.R.string.status_bar_cast);
```

图1-34 添加状态类图标到 PhoneStatusBarPolicy 中（下）

```
@@ -167,6 +172,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    mProvisionedController = Dependency.get(DeviceProvisionedController.class);
    mKeyguardMonitor = Dependency.get(KeyguardMonitor.class);
    mLocationController = Dependency.get(LocationController.class);
+    mColorInversionController = Dependency.get(ColorInversionController.class);

    mSlotCast = context.getString(com.android.internal.R.string.status_bar_cast);
    mSlotHotspot = context.getString(com.android.internal.R.string.status_bar_hotspot);
@@ -181,6 +187,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    mSlotHeadset = context.getString(com.android.internal.R.string.status_bar_headset);
    mSlotDataSaver = context.getString(com.android.internal.R.string.status_bar_data_saver);
    mSlotLocation = context.getString(com.android.internal.R.string.status_bar_location);
+    mSlotColorInversion = context.getString(com.android.internal.R.string.status_bar_inversion);

    // listen for broadcasts
    IntentFilter filter = new IntentFilter();
@@ -249,6 +256,7 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    mDataSaver.addCallback(this);
    mKeyguardMonitor.addCallback(this);
    mLocationController.addCallback(this);
+    mColorInversionController.addCallback(this);

    SysUIServiceProvider.getComponent(mContext, CommandQueue.class).addCallbacks(this);
    ActivityManagerWrapper.getInstance().registerTaskStackListener(mTaskListener);
@@ -300,6 +308,17 @@ public class PhoneStatusBarPolicy implements Callback, Callbacks,
    updateLocation();
}

+ @Override
+ public void onColorInversionChanged(boolean enable) {
+     Log.i(TAG, "onColorInversionChanged enable-->" + enable);
+     if (enable) {
+         mIconController.setIcon(mSlotColorInversion, Color_INVERT_STATUS_ICON_ID,
+             mContext.getString(R.string.accessibility_color_invert_active));
+     } else {
+         mIconController.removeAllIconsForSlot(mSlotColorInversion);
+     }
+ }
+ }
```


说明

图 1-33 和图 1-34 均属于添加状态类图标到 PhoneStatusBarPolicy 中的流程。由于流程较长，故分上下。

步骤 5 在 Dependency 中创建 ColorInversionController 对象，如图 1-35 所示。

图1-35 创建 ColorInversionController 对象

```
--- a/packages/SystemUI/src/com/android/systemui/Dependency.java
+++ b/packages/SystemUI/src/com/android/systemui/Dependency.java
@@ -65,6 +65,8 @@ import com.android.systemui.statusbar.policy.BluetoothController;
import com.android.systemui.statusbar.policy.BluetoothControllerImpl;
import com.android.systemui.statusbar.policy.CastController;
import com.android.systemui.statusbar.policy.CastControllerImpl;
+import com.android.systemui.statusbar.policy.ColorInversionController;
+import com.android.systemui.statusbar.policy.ColorInversionControllerImpl;
import com.android.systemui.statusbar.policy.ConfigurationController;
import com.android.systemui.statusbar.policy.DarkIconDispatcher;
import com.android.systemui.statusbar.policy.DataSaverController;
@@ -179,6 +181,9 @@ public class Dependency extends SystemUI {
    mProviders.put(LocationController.class, () ->
        new LocationControllerImpl(mContext, getDependency(BG_LOOPER)));

+    mProviders.put(ColorInversionController.class, () ->
+        new ColorInversionControllerImpl(mContext));
+
    mProviders.put(RotationLockController.class, () ->
        new RotationLockControllerImpl(mContext));
```

说明

修改完后建议测试一下 CTS。

---结束

1.4.2 更改系统图标

系统图标有自动反色的功能，因此不论提供的图标资源是彩色或是黑白色，都会被强制转变颜色。如果需要保持图标原有颜色，需要进行相应的修改，这里以客制化飞行模式图标为例，步骤如下。

步骤 1 修改飞行模式图标资源，文件路径为

/frameworks/base/packages/SystemUI/res/drawable/stat_sys_airplane_mode.xml。

步骤 2 修改对应的视图文件，飞行模式图标的视图文件是 StatusBarIconview.java，如图 1-36 所示。

图1-36 修改 StatusBarIconview.java

```
--- a/packages/SystemUI/src/com/android/systemui/statusbar/StatusBarIconView.java
+++ b/packages/SystemUI/src/com/android/systemui/statusbar/StatusBarIconView.java
@@ -889,6 +889,10 @@ public class StatusBarIconView extends AnimatedImageView implements StatusIconDi

    @Override
    public void onDarkChanged(Rect area, float darkIntensity, int tint) {
+        if("airplane".equals(mSlot)){
+            Log.i(TAG, "skip airplane");
+            return;
+        }
        int areaTint = getTint(area, this, tint);
        ColorStateList color = ColorStateList.valueOf(areaTint);
        setImageTintList(color);
```

步骤 3 由于下拉状态栏和锁屏状态栏对 StatusIcons 做了特殊处理，故将飞行模式图标添加到 TintedIconManager 时需要修改 StatusBarIconController.java，修改如图 1-37 所示。

图1-37 修改 StatusBarIconController.java

```
--- a/packages/SystemUI/src/com/android/systemui/statusbar/phone/StatusBarIconController.java
+++ b/packages/SystemUI/src/com/android/systemui/statusbar/phone/StatusBarIconController.java
@@ -173,6 +173,9 @@ public interface StatusBarIconController {
     protected void onIconAdded(int index, String slot, boolean blocked,
                               StatusBarIconHolder holder) {
         StatusIconDisplayable view = addHolder(index, slot, blocked, holder);
+        if ("airplane".equals(slot)) {
+            return;
+        }
         view.setStaticDrawableColor(mColor);
         view.setDecorColor(mColor);
     }
@@ -183,6 +186,9 @@ public interface StatusBarIconController {
     View child = mGroup.getChildAt(i);
     if (child instanceof StatusIconDisplayable) {
         StatusIconDisplayable icon = (StatusIconDisplayable) child;
+        if ("airplane".equals(icon.getSlot())) {
+            return;
+        }
         icon.setStaticDrawableColor(mColor);
         icon.setDecorColor(mColor);
     }
 }
```

说明

需要注意的是，修改完成后建议测试一下 CTS。

1.5 常见问题分析

去掉 Volte 图标

【问题分析】

状态栏图标显示区域有限，特别是当出现尺寸较大的 Volte 图标时，会导致其他系统图标无法显示。这种情况在刘海屏项目中经常出现。

【解决方法】

由于状态栏图标控件最大尺寸不会超过手机宽度的一半，而下拉状态栏明显大于一半。故以手机宽度的 1/2 为分界点，当宽度大于分界点时允许显示 Volte，小于时则不显示。核心代码修改如图 1-38 所示。

图1-38 核心代码修改

```

@@ -216,12 +217,22 @@ public class StatusIconContainer extends AlphaOptimizedLinearLayout {
    if (DEBUG) android.util.Log.d(TAG, "calculateIconTranslations: start=" + translationX
        + " width=" + width + " underflow=" + mNeedsUnderflow);

    // Collect all of the states which want to be visible
    for (int i = childCount - 1; i >= 0; i--) {
        View child = getChildAt(i);
        StatusIconDisplayable iconView = (StatusIconDisplayable) child;
        StatusIconState childState = getViewStateFromChild(child);

+        if (iconView instanceof StatusBarMobileView) {
+            StatusBarMobileView mobileView = (StatusBarMobileView) iconView;
+            if (translationX < 360) { //此值可设置为手机宽度的一般修改
+                mobileView.updateVolte(false);
+            } else {
+                mobileView.updateVolte(true);
+            }
+        }

        if (!iconView.isIconVisible() || iconView.isIconBlocked()) {
            childState.visibleState = STATE_HIDDEN;
            if (DEBUG) Log.d(TAG, "skipping child (" + iconView.getSlot() + ") not visible");

```

说明

结合 StatusBarContainer 图标显示的逻辑，可以在 calculateIconTranslation 里判断当前空间的宽度大小。

通知图标显示不全

【问题分析】

当刘海屏很宽时，左侧通知图标会出现显示不全的情况。这种现象主要出现在刘海屏项目中。当只有一个通知图标时，通知图标不会自动转换成点图标。

【解决方法】

分析 log 发现当前情况下 firstOverflowIndex 一直为-1。根据代码 firstOverflowIndex=noOverflowAfter&&!forceOverflow? i-1: i 可得知，noOverflowAfter 为 true 而 forceOverflow 为 false，导致为-1。故只需要将代码改为 firstOverflowIndex=i 即可。

禁止进入状态栏

【问题分析】

此类情况一般是由于客户需要实现在特定的应用界面内强制显示成全屏，并且不可下拉状态栏的功能。

【解决方法】

上述功能主要由 WMS 模块进行控制，很多应用（如工程测试应用）都需要用到该功能，WMS 模块修改方式如图 1-39 所示。

图1-39 WMS 模块修改

```
diff --git a/core/java/android/view/WindowManagerGlobal.java b/core/java/android/view/WindowManagerGlobal.java
index 3b3d930..829002b 100644
--- a/core/java/android/view/WindowManagerGlobal.java
+++ b/core/java/android/view/WindowManagerGlobal.java
@@ -347,6 +347,11 @@ public final class WindowManagerGlobal {
    }

    root = new SprdViewRootImpl(view.getContext(), display);

+   if ( wparams != null && (null != wparams.packageName) && wparams.packageName.startsWith("your app packageName")) {
+       wparams.flags |= WindowManager.LayoutParams.FLAG_FULLSCREEN;
+   }

    view.setLayoutParams(wparams);

diff --git a/services/core/java/com/android/server/policy/PhoneWindowManager.java b/services/core/java/com/android/server/policy/PhoneWindowManager.java
index a14c634..2014330 100644
--- a/services/core/java/com/android/server/policy/PhoneWindowManager.java
+++ b/services/core/java/com/android/server/policy/PhoneWindowManager.java
@@ -2082,6 +2082,11 @@ public class PhoneWindowManager extends AbsPhoneWindowManager implements WindowM
    @Override
    public void onSwipeFromTop() {
        if (mStatusBar != null) {
+           WindowManager.LayoutParams wparams = mFocusedWindow != null ? mFocusedWindow.getAttrs() : null;
+           if ( wparams != null && (null != wparams.packageName) && wparams.packageName.startsWith("your app packageName")) {
+               return;
+           }

            requestTransientBars(mStatusBar);
        }
    }
}
```

Unisoc Confidential For hiar

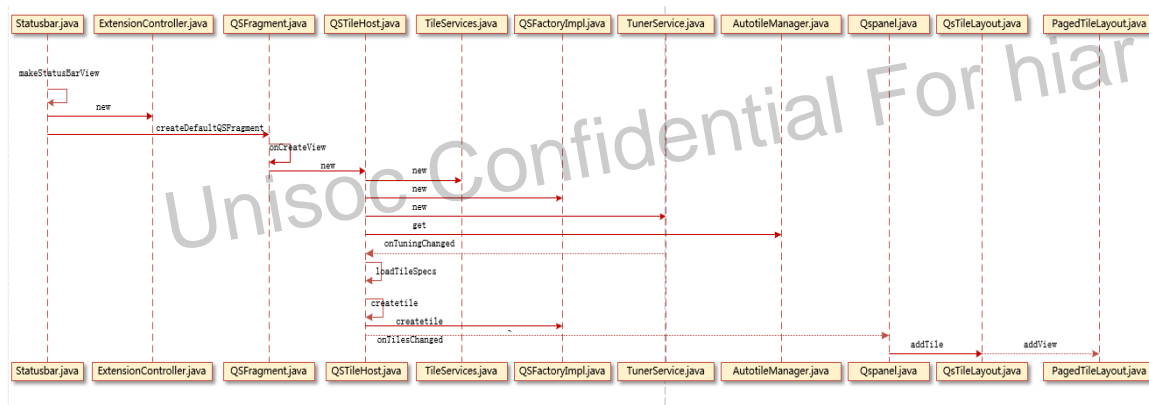
2 快捷设置

快捷设置（QS）功能是 SystemUI 的另一项重要功能，快捷设置的整个布局文件是 frameworks/base/packages/SystemUI/res/layout/qs_panel.xml。

2.1 快捷设置介绍

快捷设置主要提供常用功能的设置，比如开启和关闭 WiFi、蓝牙等。用户只要在任意界面下拉状态栏就可以调出此界面进行功能设置。快捷设置在 NotificationPanelView 中，其布局是 status_bar_expanded.xml 里的 qs_frame，初始化流程如图 2-1 所示。

图2-1 QS 初始化流程



从图 2-1 可以看到，构造快捷设置在 QSTileHost 中进行。快捷设置的显示主要分三个场景：

- 默认界面
- 编辑界面
- 详情界面

说明

关于详情界面，只有部分功能有该界面（如省电模式）。

QS 场景定义

每个 QS 场景都在 qs_panel 中有定义，其代码如图 2-2 所示。

图2-2 QS 场景定义

```

<com.android.systemui.qs.QSContainerImpl
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/quick_settings_container"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:clipToPadding="false"
    android:clipChildren="false" >

    <!-- Main QS background -->
    <View
        android:id="@+id/quick_settings_background"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:elevation="4dp"
        android:background="@drawable/qs_background_primary" />

    <!-- Black part behind the status bar -->
    <View
        android:id="@+id/quick_settings_status_bar_background"
        android:layout_width="match_parent"
        android:layout_height="@*android:dimen/quick_qs_offset_height"
        android:clipToPadding="false"
        android:clipChildren="false"
        android:background="#ff000000" />

    <!-- Gradient view behind QS -->
    <View
        android:id="@+id/quick settings gradient view"
        android:layout_width="match_parent"
        android:layout_height="126dp"
        android:layout_marginTop="@*android:dimen/quick_qs_offset_height"
        android:clipToPadding="false"
        android:clipChildren="false"
        android:background="@drawable/qs_bg_gradient" />

    <com.android.systemui.qs.QSPanel
        android:id="@+id/quick_settings_panel"
        android:layout_marginTop="@*android:dimen/quick_qs_offset_height"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="@dimen/qs_footer_height"
        android:elevation="4dp"
        android:background="@android:color/transparent"
        android:focusable="true"
        android:accessibilityTraversalBefore="@id/qs_carrier_text"
    />

    <include layout="@layout/quick_status_bar_expanded_header" />
    <include layout="@layout/qs_footer_impl" />
    <include android:id="@+id/qs_detail" layout="@layout/qs_detail" />
    <include android:id="@+id/qs_customize" layout="@layout/qs_customize_panel"
        android:visibility="gone" />

```

QSPanel的背景颜色

QS快捷设置区域

顶部黑色状态栏区域

底部区域

快捷设置详情界面

快捷设置客制化界面

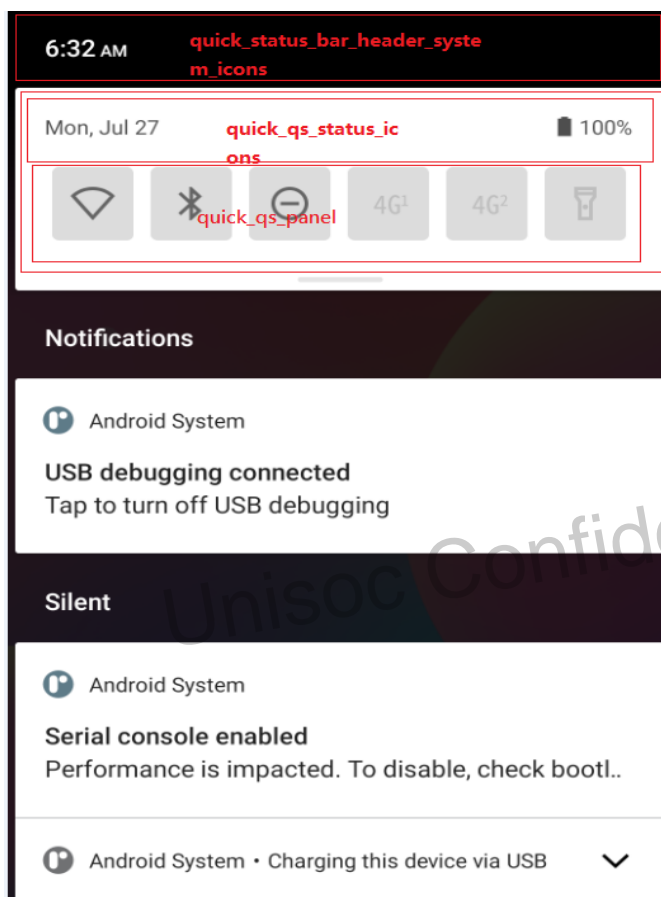
默认界面

默认界面有两种形态：

- 半展开状态，布局文件是
/frameworks/base/packages/SystemUI/res/layout/quick_status_bar_expanded_header.xml。
- 完全展开形态，布局文件是/frameworks/base/packages/SystemUI/res/layout/qs_panel.xml。

半展开形态显示效果如图 2-3 所示。

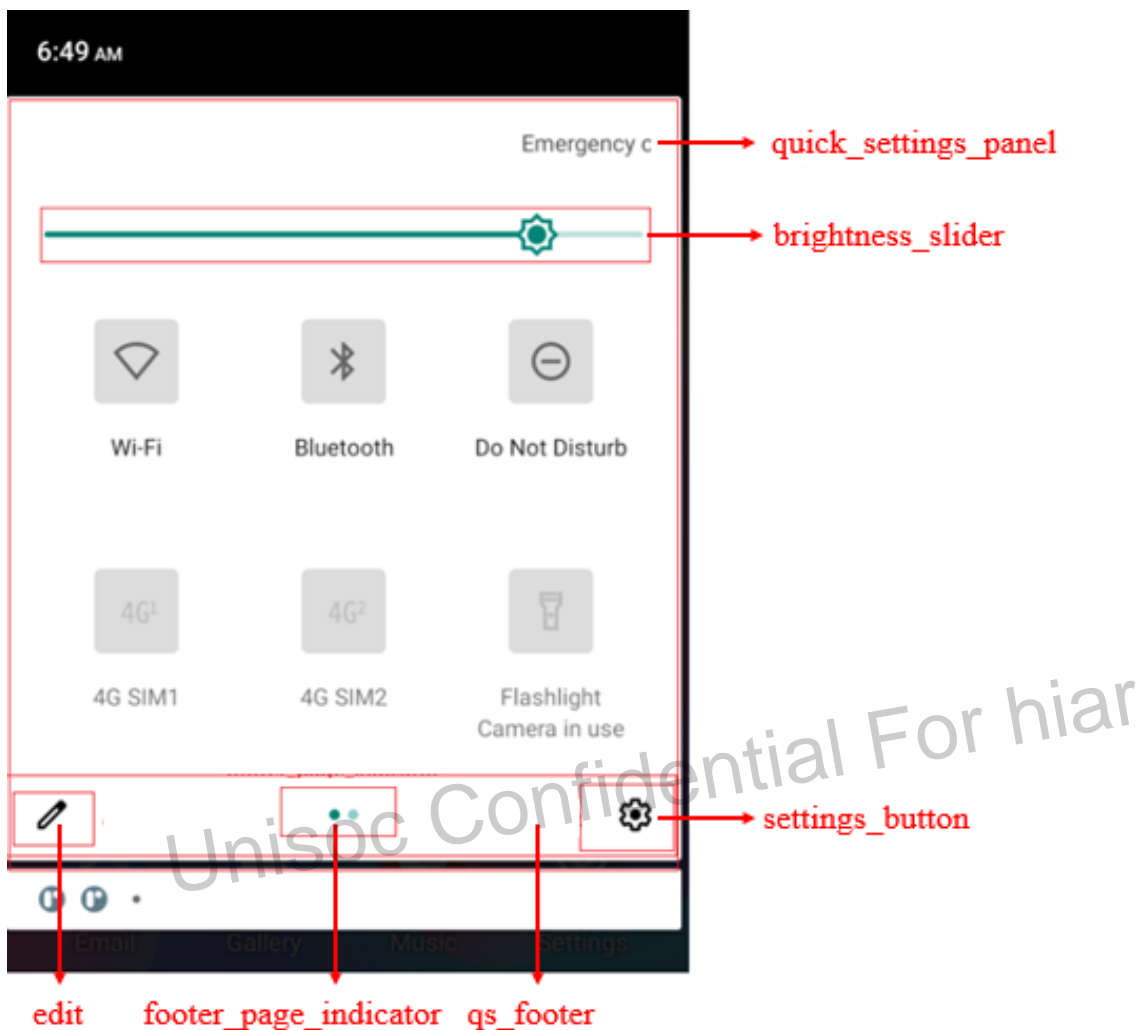
图2-3 QS 默认界面半展开形态



其中 QuickQSPanel 继承于 QSPanel。

完全展开形态显示效果如图 2-4 所示。

图2-4 QS 默认场景完全展开形态



默认场景完全展开形态的主要区域是 `quick_settings_panle`，其中包含亮度调节显示 `SeekBar`（此处的亮度调节只是显示功能），对应的布局文件是

`/frameworks/base/packages/SystemUI/res/layout/quick_settings_brightness_dialog.xml`。QuickSettings 的 `viewPage` 对应的布局文件是 `/frameworks/base/packages/SystemUI/res/layout/qs_paged_tile_layout.xml`。

这两个布局都是通过动态添加的，并非在 `qs_panel.xml` 里直接引用。通过单击 `qs_footer_impl` 中的 `edit` 按钮，进入快捷设置编辑界面，`edit` 点击事件的处理是在 `QsFooterImpl.java` 中进行，如图 2-5 所示。

图2-5 edit 点击事件处理

```

134     @Override
135     protected void onFinishInflate() {
136         super.onFinishInflate();
137         mEdit = findViewById(android.R.id.edit);
138         mEdit.setOnClickListener(view ->
139             mActivityStarter.postQSRRunnableDismissingKeyguard(() ->
140                 mQsPanel.showEdit(view));
141     }

```

- postQSRRunnableDismissingKeyguard: 先解锁，如果是安全锁需要输入安全密码解锁。
- showEdit(view): 显示客制化快捷设置界面。

编辑界面

编辑界面显示效果如图 2-6 所示。

图2-6 编辑界面显示效果



编辑界面对应的布局文件是/packages/SystemUI/res/layout/qs_customize_panel.xml QSCustomizer.java。主要通过 RecyclerView 来显示各个数据，可以看到大致分为三个区域：

- 上方是现有的 tile。
- 中间是可设置的 tile。
- 下方是第三方 tile。

详情界面

详情界面的主要布局文件为/frameworks/base/packages/SystemUI/res/layout/qs_detail.xml。对应的 java 文件是/frameworks/base/packages/SystemUI/src/com/android/systemui/qs/QSDetail.java。此界面的显示和功能强相关。

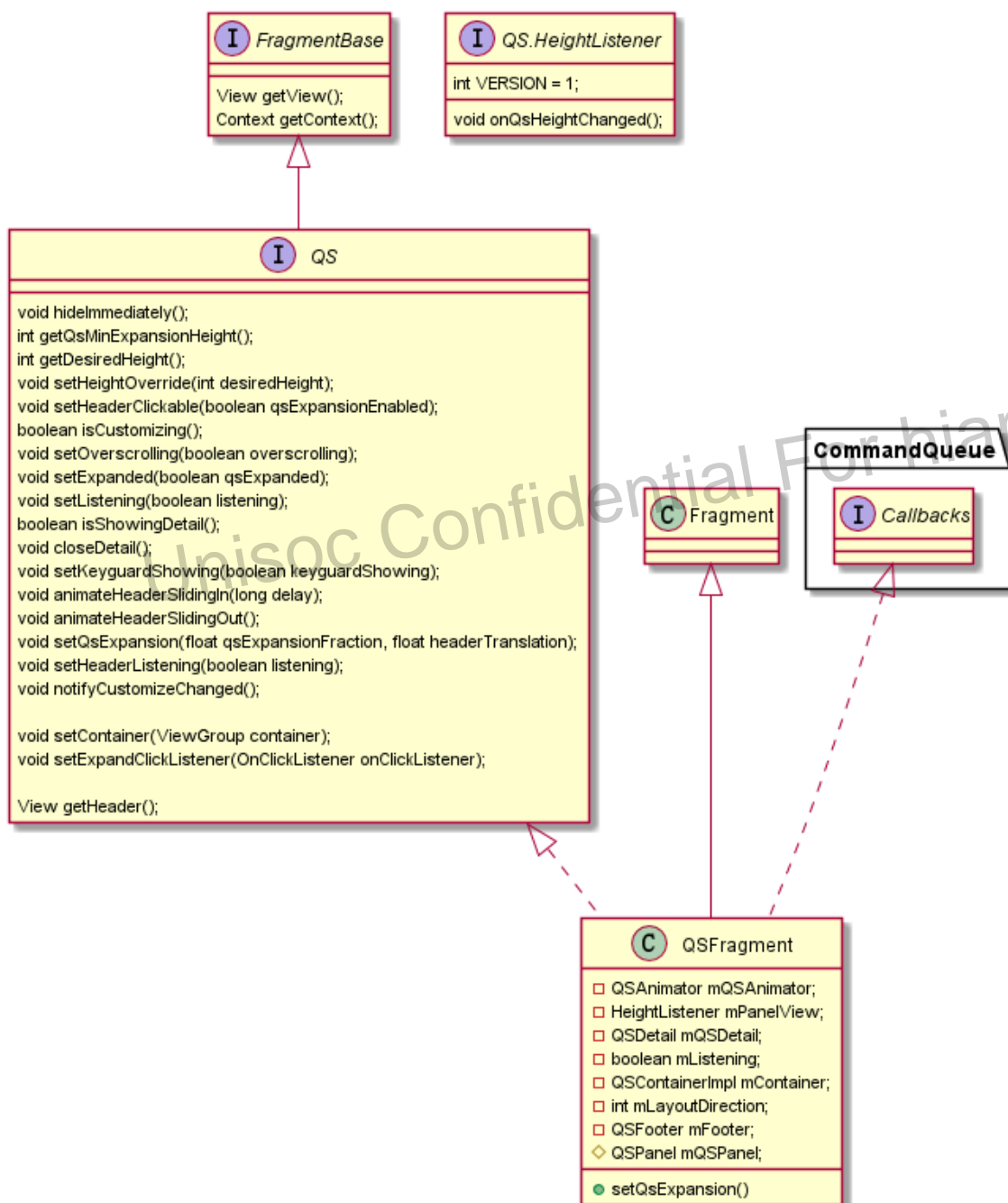
Unisoc Confidential For hiar

2.2 主要控件

2.2.1 QSFragment

QSFragment 控件是整个快捷设置的最大视图，主要用于控制各快捷设置界面的动画效果以及场景显示的切换，其类图结构主要如图 2-7 所示。

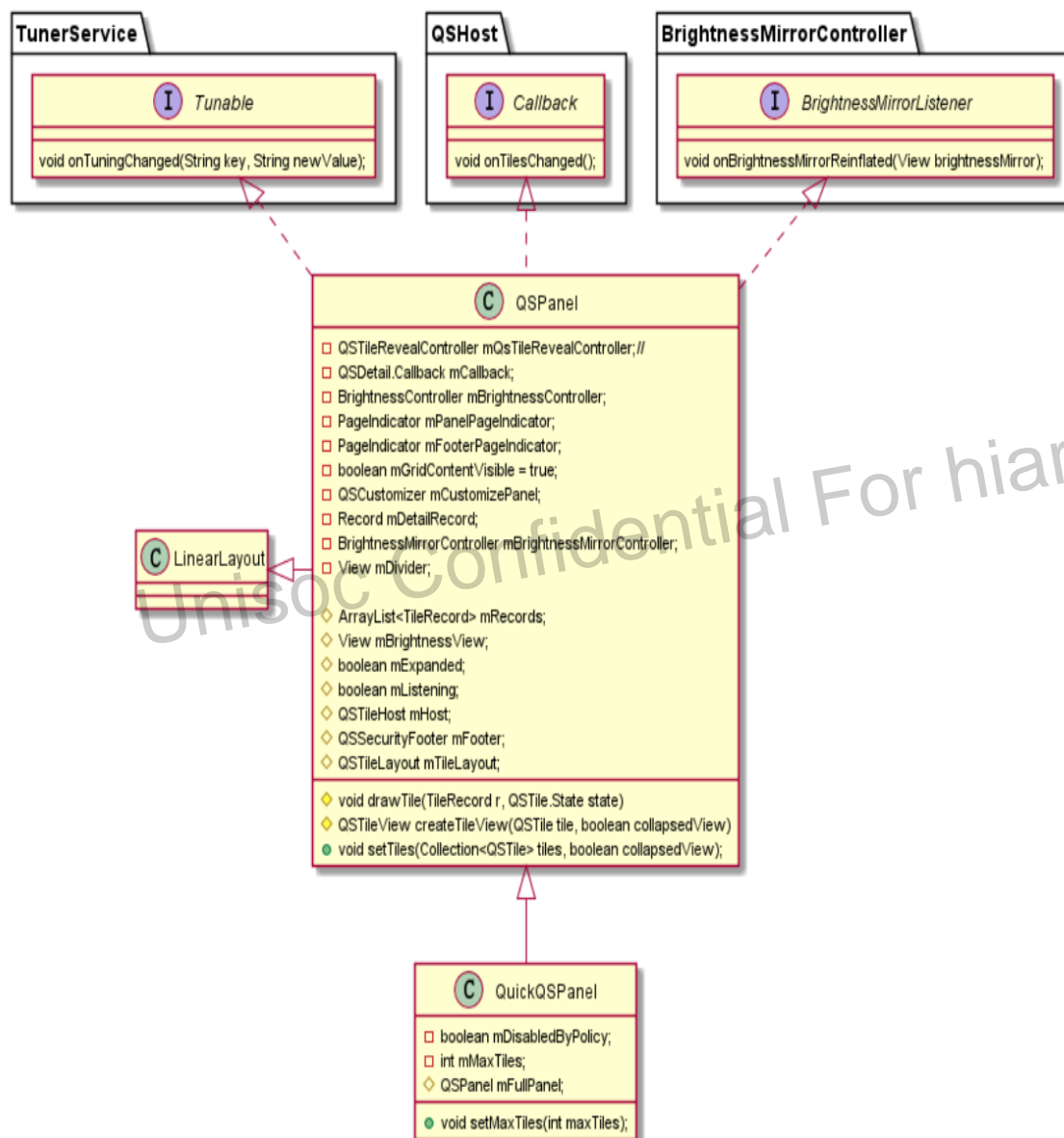
图2-7 QSFragment 类图结构



2.2.2 QSPanel 和 QuickQSPanel

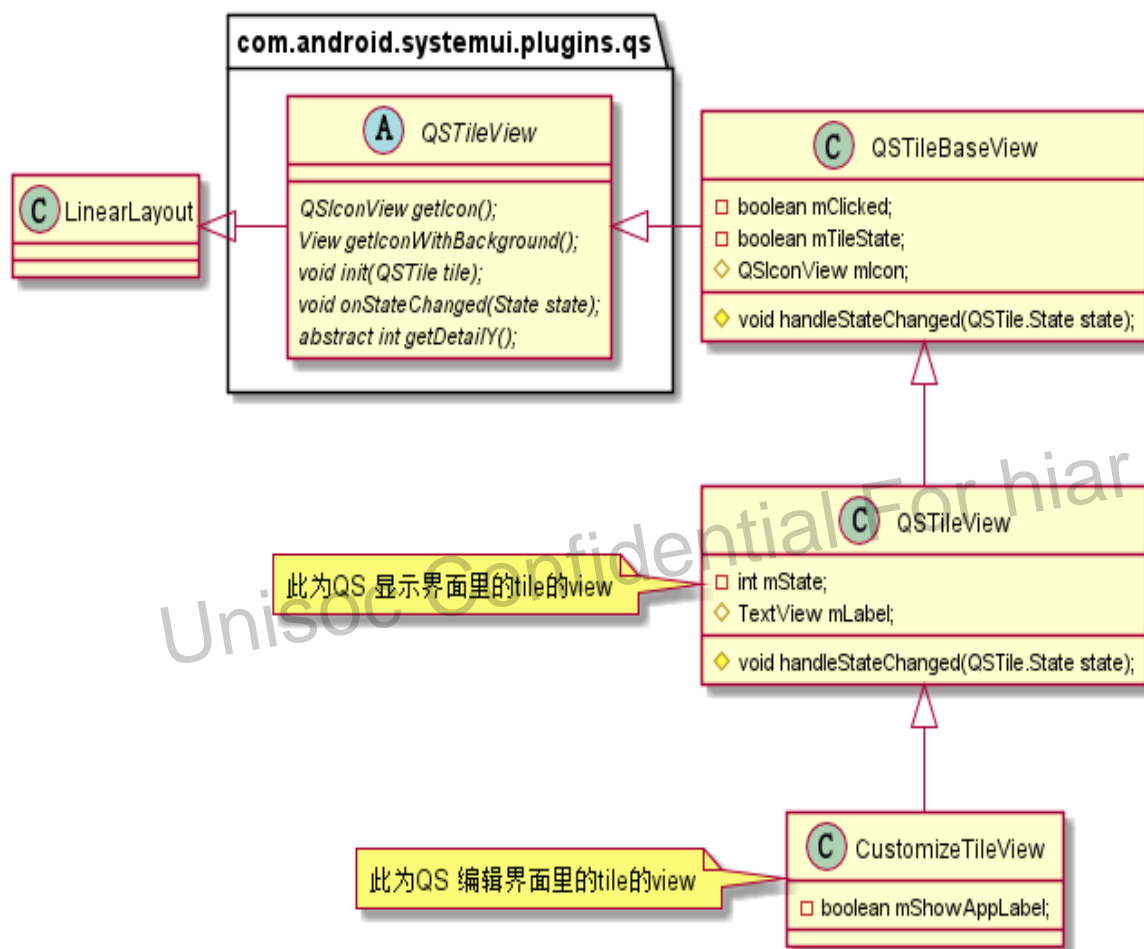
QSPanel 和 QuickQSPanel 都是快捷设置的设置区域控件，主要控制 tile 的显示。其中 QuickQSPanel 继承于 QSPanel，相对于 QSPanel，QuickQSPanel 对 tile 的显示个数做了限制（sDefaultMaxTiles = 6）。其类图关系如图 2-8 所示。

图2-8 QSPanel 和 QuickQSPanel 类图关系



QSTileView 控件是快捷设置里最重要的控件之一。该控件继承于 LinearLayout，是快捷设置界面控制开关的视图，其类图关系如图 2-9 所示。主要包含一个图标（mIcon）和一个标题（mLabel），并通过 handleStateChanged 来更新视图的状态。

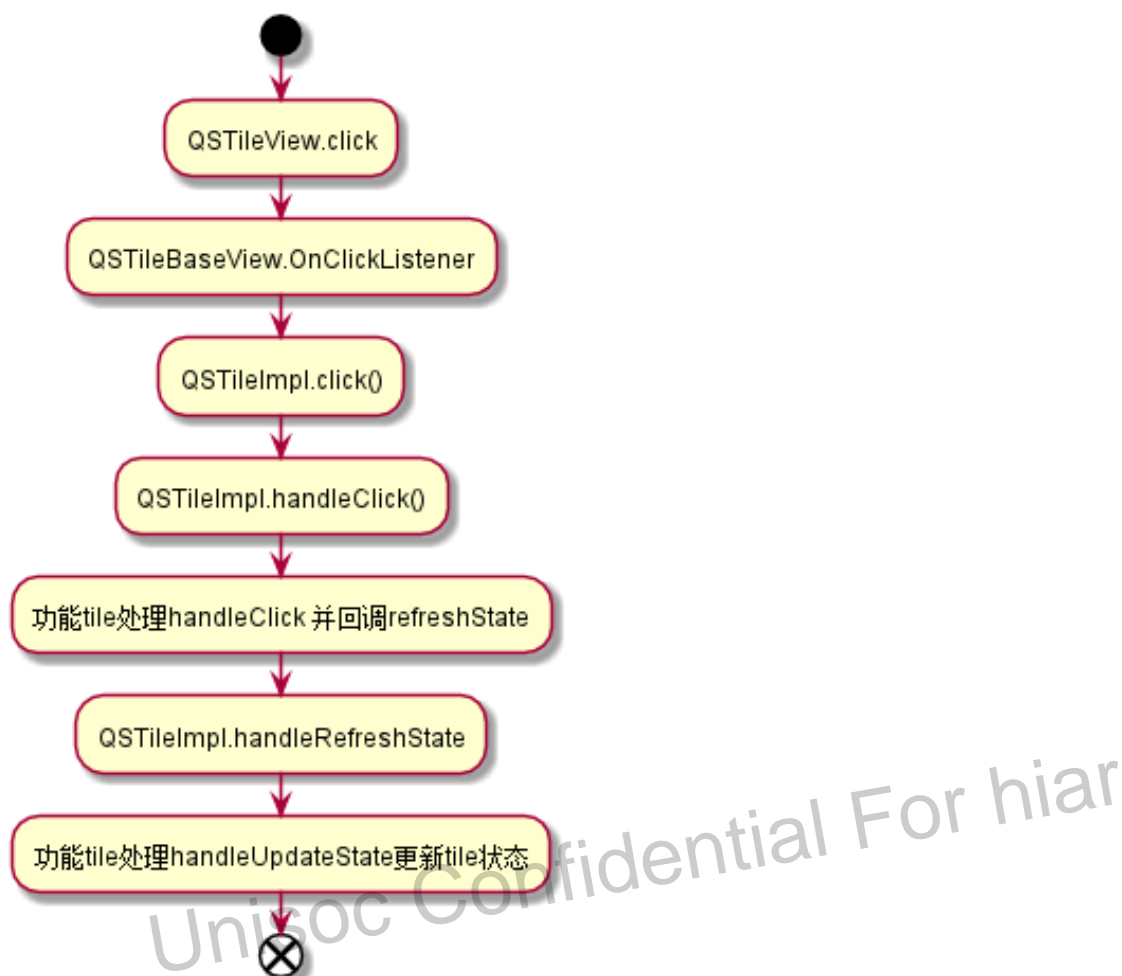
图2-9 QSTileView 类图关系



快捷设置的主要功能包含点击流程和亮度调节。

每个快捷设置按钮都是一个 `QSTileView`，因此 `Tile` 的点击事件都是在 `QSTileView` 中监听，点击流程大致如图 2-10 所示。

图2-10 QSTile 点击流程



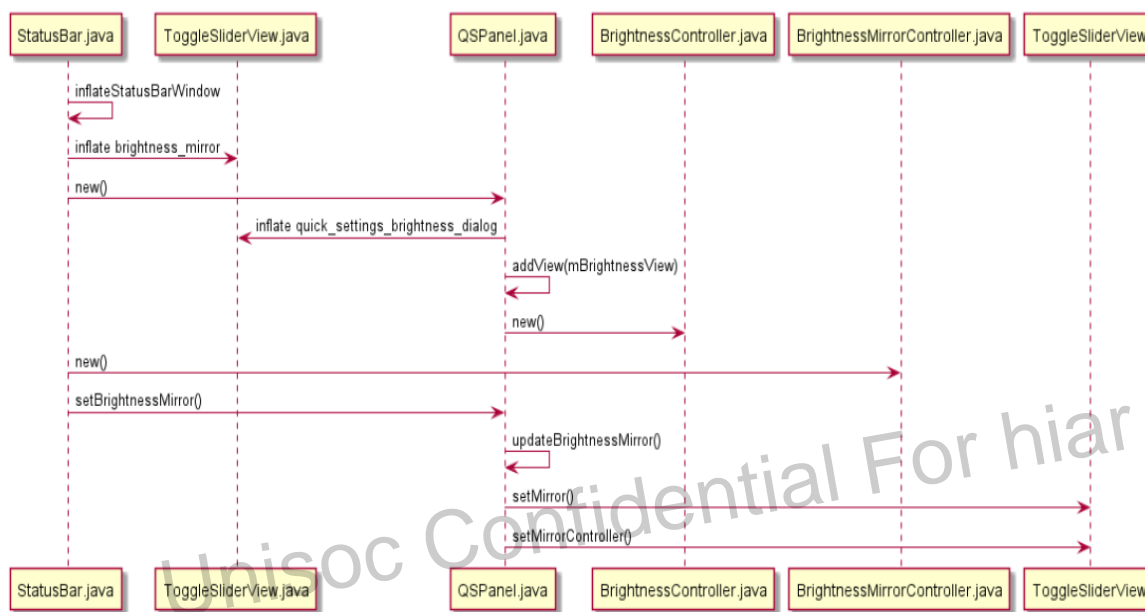
2.3.2 亮度调节

亮度调节由两个相同的 View 一起实现，让亮度调节有更好的预览效果。

- 滑动调节亮度时，整个下拉状态栏的透明度会设置为 0，然后将 `super_notification_shade` 里的 `brightness_mirror` 设为可见。
- 亮度调节结束后，下拉状态栏透明度会设置为 255，`brightness_mirror` 再次隐藏。

亮度调节初始化流程如图 2-11 所示。

图2-11 亮度调节初始化流程



下拉状态栏完全展开状态显示的是 QSPanel 中的 view（id:brightness_slider）。

- 下拉状态栏亮度调节的主要布局文件是
/frameworks/base/packages/SystemUI/res/layout/quick_settings_brightness_dialog.xml。
- 进度条布局文件是
/frameworks/base/packages/SystemUI/src/com/android/systemui/settings/ToggleSliderView.java。

说明

当前 view 通过 addview 的形式添加。

在亮度调节过程中，`onStartTrackingTouch`、`onProgressChanged`、`onStopTrackingTouch` 每次都会执行两遍。一次是在 `panleview` 中，一次是在 `StatusBarWindowView` 中。滑动调节亮度的流程如图 2-12 所示。

图2-12 滑动调节亮度的流程



2.4 快捷设置客制化

快捷设置的客制化从新增快捷设置和添加亮度模式图标两个方面进行描述。

2.4.1 新增快捷设置

新增快捷设置有两种实现方式：

- 在 SystemUI 里实现。
- 在第三方应用里实现。

在 SystemUI 里实现

SystemUI 中新增快捷设置步骤如下：

- 步骤 1 增加*Tile.java、*Controller.java、*ControllerImpl.java 以及其他所需的资源文件。其中*Tile.java 继承 QSTileImpl<TState>，*Controller.java 继承 CallbackController<Callback>，*ControllerImpl.java 实现*Controller.java。
- 步骤 2 在 QSTileImpl.java 的 createTile 方法中添加 new *Tile()。
- 步骤 3 在 frameworks\base\packages\SystemUI\res\values\config.xml 里的 quick_settings_tiles_stock 中添加功能字段“*”。如需默认显示，需要同时在 quick_settings_tiles_default 里添加功能字段“*”。

说明

在 quick_settings_tiles_default 里添加功能字段“*”时，请注意项目是否有做 overlay。

---结束

在第三方应用里实现

在 Android 7.0 及以上版本中，Google 提供了 API 用于第三方应用添加 Tile。但此 Tile 只能添加在编辑界面中，由用户选择是否添加到下拉界面。

实现方式非常简单，只要继承 `android.service.quicksettings.TileService`，然后在 `AndroidManifest.xml` 中注册 Service 即可。

说明

Service 这里指的是 Tileservice。

【示例】

添加 `TileService` 服务，如图 2-13 所示。

图2-13 添加 `TileService` 服务

```
<service
    android:name=".TestTile"
    android:label="@string/test"
    android:icon="@drawable/tile_icon_test"
    android:permission="android.permission.BIND_QUICK_SETTINGS_TILE"
    android:enabled="true">
    <intent-filter>
        <action android:name="android.service.quicksettings.action.QS_TILE" />
    </intent-filter>
</service>
```

说明

- 一定要添加 `permission` 和 `filter`。
- `label` 是下拉状态栏显示的名称。
- `icon` 是下拉状态栏使用的图标。

在状态栏编辑界面中，label 下面会显示应用的名称，如图 2-14 所示。

图2-14 label 下显示应用名称

```
package com.example.unisoc.test;

import android.service.quicksettings.TileService;
import android.util.Log;

public class TestTile extends TileService {

    private static final String TAG = "TestTile";
    private boolean mClick = false;

    @Override
    public void onStartListening() {
        super.onStartListening();
    }

    @Override
    public void onStopListening() {
        super.onStopListening();
    }

    @Override
    public void onClick() {
        super.onClick();
        mClick = !mClick;
        Log.i(TAG, msg: "mClick-->"+mClick);
    }
}
```

可以看到 java 代码实现非常简单，相关功能只要在 onClick 中实现即可。

- 如果功能要求比较复杂，建议使用“在 SystemUI 里实现”方案。
- 如果功能相对简单的话，建议使用“在第三方应用里实现”方案。

📖 说明

注意 TileService 里没有提供 onLongClick 接口，长按默认跳到第三方应用信息界面。

2.4.2 添加亮度模式图标

添加亮度模式图标时，需要在 quick_settings_brightness_dialog.xml 中添加对应的 Imageview，如图 2-15 所示。

图2-15 添加 Imageview

```
<ImageView
    android:id="@+id/brightness_icon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:layout_marginEnd="8dp"
    android:src="@drawable/ic_qs_brightness_auto_off"
    android:contentDescription="@null"
    android:visibility="gone" />
```

只需要在代码里根据要求将亮度模式图标显示出来即可，可以在 BrightnessMirrorController 和 BrightnessController 对图标进行更新。修改 BrightnessMirrorController 主要是为了在 showMirror 时查询当前亮度模式，并显示图标，如图 2-16 所示。

图2-16 修改 BrightnessMirrorController

```
public void showMirror() {
    ImageView mIcon = mBrightnessMirror.findViewById(R.id.brightness_icon);
    mBrightnessMirror.setVisibility(View.VISIBLE);
    if (mIcon != null) {
        boolean automatic = Settings.System.getIntForUser(mContext.getContentResolver(),
            Settings.System.SCREEN_BRIGHTNESS_MODE,
            Settings.System.SCREEN_BRIGHTNESS_MODE_MANUAL,
            UserHandle.USER_CURRENT) == Settings.System.SCREEN_BRIGHTNESS_MODE_AUTOMATIC;
        mIcon.setVisibility(View.VISIBLE);
        mIcon.setImageResource(automatic ?
            com.android.systemui.R.drawable.ic_qs_brightness_auto_on :
            com.android.systemui.R.drawable.ic_qs_brightness_auto_off);
    }
    mVisibilityCallback.accept(true);
    mNotificationPanel.setPanelAlpha(0, true /* animate */);
}
```

对于 BrightnessController 的修改，主要是在 updateIcon 时获取当前模式，然后将更新 icon 即可，如图 2-17 所示。

图2-17 修改 BrightnessController

```
private void updateIcon(boolean automatic) {
    automatic = Settings.System.getIntForUser(mContext.getContentResolver(),
        Settings.System.SCREEN_BRIGHTNESS_MODE,
        Settings.System.SCREEN_BRIGHTNESS_MODE_MANUAL,
        UserHandle.USER_CURRENT) == Settings.System.SCREEN_BRIGHTNESS_MODE_AUTOMATIC;
    if (mIcon != null) {
        mIcon.setVisibility(View.VISIBLE);
        mIcon.setImageResource(automatic && SHOW_AUTOMATIC_ICON ?
            com.android.systemui.R.drawable.ic_qs_brightness_auto_on :
            com.android.systemui.R.drawable.ic_qs_brightness_auto_off);
    }
}
```

2.5 常见问题分析

去掉下拉菜单中的快捷小组件

【问题分析】

这个问题处理相对简单，一般这类功能代码中都会提供配置参数来处理。这里主要介绍如何去寻找此配置参数。

【解决方法】

通过 2.1 快捷设置的介绍，快捷设置组件的初始化在 QSTileHost 中通过 TunerService 的 onTuningChanged 触发。只要找到 onTuningChanged，检查初始化代码，如图 2-18 所示。

图2-18 onTuningChanged 初始化代码

```
@Override
public void onTuningChanged(String key, String newValue) {
    if (!TILES_SETTING.equals(key)) {
        return;
    }
    if (DEBUG) Log.d(TAG, "Recreating tiles");
    if (newValue == null && UserManager.isDeviceInDemoMode(mContext)) {
        newValue = mContext.getResources().getString(R.string.quick_settings_tiles_retail_mode);
    }
    final List<String> tileSpecs = loadTileSpecs(mContext, newValue, false);
    int currentUser = UserManager.getCurrentUser();
    if (tileSpecs.equals(mTileSpecs) && currentUser == mCurrentUser) return;
    mTiles.entrySet().stream().filter(tile -> !tileSpecs.contains(tile.getKey())).forEach(

protected List<String> loadTileSpecs(Context context, String tileList, boolean is_special_request) {
    final Resources res = context.getResources();
    String defaultTileList = res.getString(R.string.quick_settings_tiles_default);
    if (tileList == null) {
        tileList = res.getString(R.string.quick_settings_tiles);
        if (DEBUG) Log.d(TAG, "Loaded tile specs from config: " + tileList);
    } else {
        if (DEBUG) Log.d(TAG, "Loaded tile specs from setting: " + tileList);
    }
}
```

发现 LoadTileSpecs 会去读 R.string.quick_settings_tiles_default，此处就是配置快捷设置项的地方。配置内容如图 2-19 所示。

图2-19 配置 quick_settings_tiles_default

```
<!-- The default tiles to display in QuickSettings -->
<string name="quick_settings_tiles_default" translatable="false">
    wifi,bt,dnd,volte1,volte2,vowifi,lte1,lte2,flashlight,rotation,battery,cell,airplane,cast,screenrecord
</string>
```

快捷设置半展开状态下的快捷设置项个数，以及快捷设置完全展开状态下快捷设置项的行列数等，都可以通过上述方法进行配置。图 2-20 所示。

图2-20 快捷项设置

```
<!-- The maximum number of tiles in the QuickQSPanel -->
<integer name="quick_qs_panel_max_columns">6</integer>

<!-- The number of columns in the QuickSettings -->
<integer name="quick_settings_num_columns">3</integer>

<!-- The number of rows in the QuickSettings -->
<integer name="quick_settings_max_rows">3</integer>

<!-- The number of columns that the top level tiles span in the QuickSettings -->
<integer name="quick_settings_user_time_settings_tile_span">1</integer>
```

Unisoc Confidential For hiar