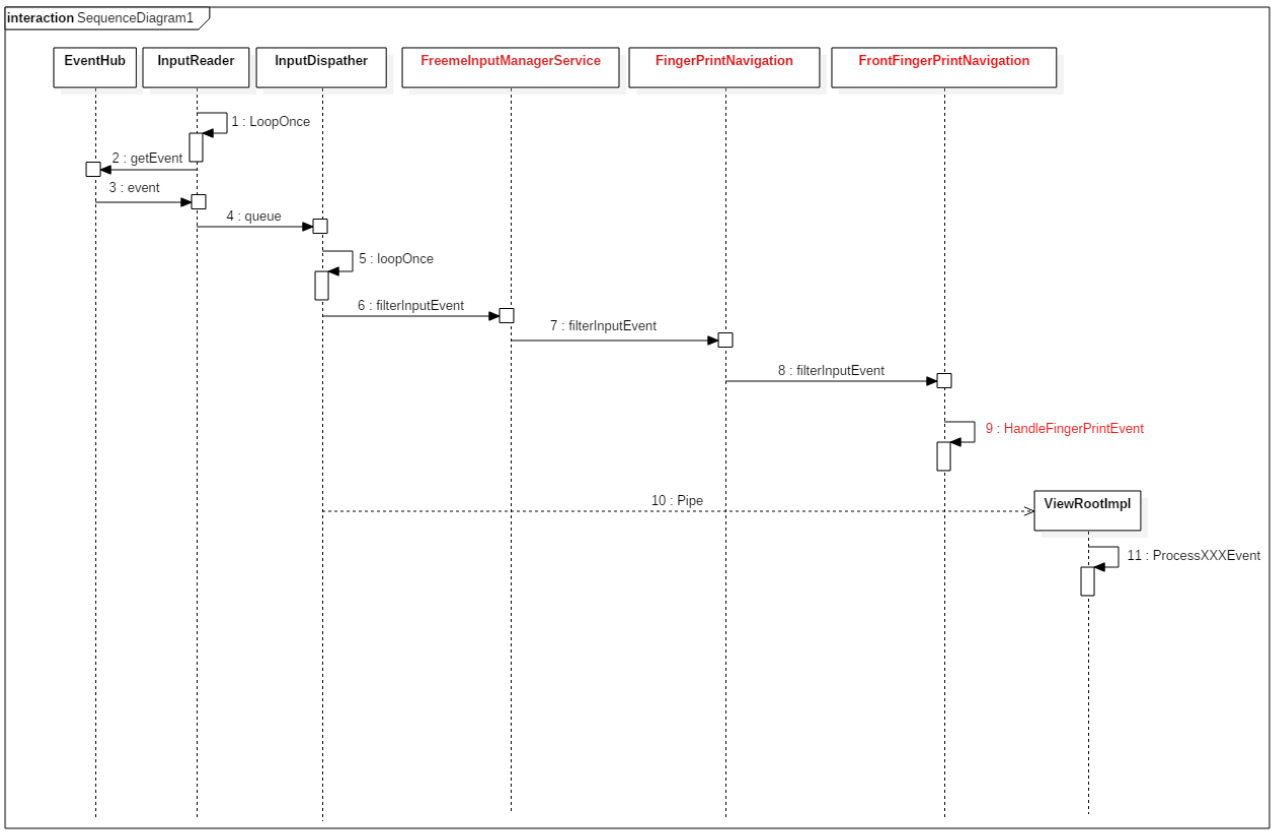
1、导航栏所有修改功能

- NavigationBar Min
- NavigationBar Color Change
- FrontPrint as NavigationBar

2、指纹事件序列



3、指纹报键

- KEYCODE_FINGERPRINT_SINGLE_TAP
- KEYCODE_FINGERPRINT_DOUBLE_TAP
- KEYCODE_FINGERPRINT_LONGPRESS
- KEYCODE_FINGERPRINT_UP
- KEYCODE_FINGERPRINT_DOWN
- KEYCODE_FINGERPRINT_LEFT
- KEYCODE_FINGERPRINT_RIGHT

4、代码摘录

XXXInputManagerService.java

- 整个类用于输入指纹的管理，整个方案的主要部分

```
private static final boolean FRONT_FINGERPRINT_NAVIGATION =
SystemProperties.getBoolean("ro.config.hw_front_fp_navi", false);
if (HwGuiShou.isFPNavigationInThreeAppsEnabled(this.mInjectCamera, this.mAnswerCall,
this.mStopAlarm) && !FRONT_FINGERPRINT_NAVIGATION) {
    Slog.d("InputManager", "close fingerprint nav");
    System.putInt(this.mResolver, FINGERPRINT_SLIDE_SWITCH, 0);
    this.mGuiShou.setFingerprintSlideSwitchValue(false);
    this.mGuiShou.registerPhoneStateListener();
}

class SettingsObserver extends ContentObserver {
    SettingsObserver(Handler handler) {
        super(handler);
        registerContentObserver(UserHandle.myUserId());
    }

    public void registerContentObserver(int userId) {
        if (!HwGuiShou.isGuiShouEnabled()) {

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_SLIDE_SWITCH), false, this, userId);
        }

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_CAMERA_SWITCH), false, this, userId);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_ANSWER_CALL), false, this, userId);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_SHOW_NOTIFICATION), false, this, userId);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_BACK_TO_HOME), false, this);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_STOP_ALARM), false, this, userId);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_LOCK_DEVICE), false, this);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_GO_BACK), false, this);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_RECENT_APP), false, this);

HwInputManagerService.this.mResolver.registerContentObserver(System.getUriFor(HwInputManagerServic
e.FINGERPRINT_MARKET_DEMO_SWITCH), false, this);

HwInputManagerService.this.mResolver.registerContentObserver(Secure.getUriFor(HwInputManagerServic
e.FINGERPRINT_GALLERY_SLIDE), false, this, userId);
    }
```

```
    public void onChange(boolean selfChange) {
        ……
    }
}

private void initObserver(Handler handler) {
    public void onChange(boolean selfChange) {
        Slog.d("InputManager", "open fingerprint nav");
        System.putInt(HwInputManagerService.this.mResolver,
HwInputManagerService.FINGERPRINT_SLIDE_SWITCH, 1);
        HwInputManagerService.this.mGuiShou.setFingerprintSlideSwitchValue(true);
        return;
    }
    各种行为的监听
    ……
}

boolean filterInputEvent(InputEvent event, int policyFlags) {
    if (this.mFingerprintNavigationFilter.filterInputEvent(event, policyFlags)) {
        return false;
    }
    if (this.mFingerPressNavi == null || this.mFingerPressNavi.filterPressueInputEvent(event)) {
        return super.filterInputEvent(event, policyFlags);
    }
    return false;
}
```

LocalDisplayAdapter.java

- 背光管理，在前置指纹作为导航且TriKey为 1 时，做特殊背光处理

```
private static final boolean FRONT_FINGERPRINT_NAVIGATION =
SystemProperties.getBoolean("ro.config.hw_front_fp_navi", false);
private static final int FRONT_FINGERPRINT_NAVIGATION_TRIKEY =
SystemProperties.getInt("ro.config.hw_front_fp_trikey", 0);

public LocalDisplayDevice(IBinder displayToken, int builtInDisplayId, PhysicalDisplayInfo[]
physicalDisplayInfos, int activeDisplayInfo) {
    if (LocalDisplayAdapter.FRONT_FINGERPRINT_NAVIGATION &&
LocalDisplayAdapter.FRONT_FINGERPRINT_NAVIGATION_TRIKEY == 1) {
        this.mButtonlight = lights.getLight(2);
    } else {
        this.mButtonlight = null;
    }
}

public void registerContentObserver(Context context, Handler handler) {
    if (context != null && FRONT_FINGERPRINT_NAVIGATION && FRONT_FINGERPRINT_NAVIGATION_TRIKEY ==
1) {
        try {
            this.mResolver = context.getContentResolver();
            this.mSettingsObserver = new SettingsObserver(this, handler);
            IntentFilter intentFilter = new IntentFilter();
            intentFilter.addAction("android.intent.action.USER_SWITCHED");
            context.registerReceiver(new UserSwtichReceiver(), intentFilter);
        } catch (Exception exp) {
            Log.e(TAG, "registerContentObserver:" + exp.getMessage());
        }
    }
}
```

LightsService.java

- 背光修改

```
private static final boolean FRONT_FINGERPRINT_NAVIGATION =
SystemProperties.getBoolean("ro.config.hw_front_fp_navi", false);
public void setBrightness(int brightness, int brightnessMode) {
    this.mCurrentBrightness = brightness;
    if (!LightsService.inMirrorLinkBrightnessMode) {
        if (this.mId == 0) {
            LightsService.mLcdBrightness = brightness;
            LightsService.this.sendUpdateaAutoBrightnessDbMsg();
            if (brightness == 0 || (!LightsService.mIsAutoAdjust && LightsService.mRatio >= 1.0d))
{
                LightsService.this.mCurBrightness = brightness;
            } else {
                setLightGradualChange(brightness, brightnessMode, false);
                return;
            }
        }
        synchronized (this) {
            brightness = LightsService.this.mapIntoRealBacklightLevel(brightness);
            int color;
            if (LightsService.this.mIsHighPrecision) {
                color = brightness & 65535;
                if (this.mId == 2 && LightsService.FRONT_FINGERPRINT_NAVIGATION &&
LightsService.FRONT_FINGERPRINT_NAVIGATION_TRIKEY == 1) {
                    color = (color * 255) / 10000;
                    setLightLocked(brightness & 255, 0, 0, 0, brightnessMode);
                    return;
                }
                setLightLocked_10000stage(color, 0, 0, 0, brightnessMode);
            } else {
                color = brightness & 255;
                if (this.mId == 2 && LightsService.FRONT_FINGERPRINT_NAVIGATION &&
LightsService.FRONT_FINGERPRINT_NAVIGATION_TRIKEY == 1) {
                    Slog.i(LightsService.TAG, "Set button brihtness:" + color);
                    setLightLocked(color, 0, 0, 0, brightnessMode);
                    return;
                }
                setLightLocked(color | (((color << 16) | UsbAudioDevice.kAudioDeviceMetaMask) |
(color << 8)), 0, 0, 0, brightnessMode);
            }
        }
    }
}
```

ViewRootImpl.java

- 滑屏过滤，为了优化当手指从屏幕底边向上滑时产生的多余报点

```
private static final boolean FRONT_FINGERPRINT_NAVIGATION =
SystemProperties.getBoolean("ro.config.hw_front_fp_navi", false);
private int processPointerEvent(QueuedInputEvent q) {
    if (!ViewRootImpl.FRONT_FINGERPRINT_NAVIGATION &&
        HwFrameworkFactory.getHwViewRootImpl().filterDecorPointerEvent(ViewRootImpl.this.mContext,
event, action, ViewRootImpl.this.mWindowAttributes, ViewRootImpl.this.mDisplay)) {
        return 1;
    }
}
```

FingerprintUtils.java

- 重写指纹错误提示

```
private static final boolean FRONT_FINGERPRINT_NAVIGATION =
SystemProperties.getBoolean("ro.config.hw_front_fp_navi", false);
public static void vibrateFingerprintErrorHw(Context context) {
    if (context != null) {
        if (FRONT_FINGERPRINT_NAVIGATION) {
            Vibrator vibrator = (Vibrator) context.getSystemService("vibrator");
            if (vibrator != null) {
                SystemVibrator sysVibrator = (SystemVibrator) vibrator;
                if (sysVibrator != null) {
                    sysVibrator.hwVibrate(null, 14);
                }
            }
        } else {
            vibrateFingerprintError(context);
        }
    }
}
```

FingerprintNavigation.java

- 具体的事件行为处理，对接InputManagerService的filterInputEvent

```
public boolean filterInputEvent(InputEvent event, int policyFlags) {
    ……
}
```

FrontFingerprintNavigation.java

- 专用于前置指纹处理，做了各种行为兼容

```
public boolean handleFingerprintEvent(InputEvent event) {
    if (!FrontFingerPrintSettings.FRONT_FINGERPRINT_NAVIGATION || !this.isNormalRunmode) {
        Log.d(TAG, "do not support frontfingerprint");
        return false;
    } else if (this.mSystemUIBackInspector.probe(event)) {
        this.mSystemUIBackInspector.handle(event);
        return true;
    } else if (this.mSystemUIRecentInspector.probe(event)) {
        this.mSystemUIRecentInspector.handle(event);
        return true;
    } else if (this.mSystemUIHomeInspector.probe(event)) {
        this.mSystemUIHomeInspector.handle(event);
        return true;
    } else if (!this.mFingerPrintUpInspector.probe(event)) {
        return false;
    } else {
        this.mFingerPrintUpInspector.handle(event);
        return true;
    }
}

public FrontFingerprintNavigation(Context context) {
    ……
    this.mSystemUIBackInspector = new SystemUIBackInspector();
    this.mSystemUIRecentInspector = new SystemUIRecentInspector();
    this.mSystemUIHomeInspector = new SystemUIHomeInspector();
    this.mFingerPrintUpInspector = new FingerPrintHomeUpInspector();
}
```

FingerPressNavigation.java

- 按压式指纹的具体事件行为处理，对接InputManagerService的filterInputEvent

```
public boolean filterPressueInputEvent(InputEvent event) {
}
```

NavigationBarPolicy.java

- 导航栏策略管理，大部分用于控制导航栏缩小

PhoneWindowManager.java

- 分屏模式Fix

```
private static boolean bHasFrontFp = SystemProperties.getBoolean("ro.config.hw_front_fp_navi",
false);
private final class UpdateRotationRunnable implements Runnable {
    private final int mRotation;

    public UpdateRotationRunnable(int rotation) {
        this.mRotation = rotation;
    }

    public void run() {
        PhoneWindowManager.this.mPowerManagerInternal.powerHint(2, 0);
        if (PhoneWindowManager.this.mWindowManagerInternal.isDockedDividerResizing()) {

PhoneWindowManager.this.mWindowManagerInternal.setDockedStackDividerRotation(this.mRotation);
        } else {
            Slog.i(PhoneWindowManager.TAG, "MyOrientationListener: updateRotation.");
            PhoneWindowManager.this.updateRotation(false);
        }
        if (PhoneWindowManager.bHasFrontFp) {
            PhoneWindowManager.this.updateSplitScreenView();
        }
    }
}
```

FingerprintManagerEx.java

- 指纹本身校准

FrontFingerPrintSettings.java

- 存儲配置值，是否使用導航欄還是使用前置指纹

```
public static synchronized boolean isNaviBarEnabled(ContentResolver resolver) {
    if (FRONT_FINGERPRINT_NAVIGATION) {
        ……
    } else {
        return true;
    }
}
```

XXXGeneralManager.java

- 配置硬件支持情况

```
private static String TRIKEY_PATH = "/proc/device-tree/huawei_touch_key";
public boolean isSupportTrikey() {
    if (!this.mInitTrikey) {
        File file = new File(TRIKEY_PATH);
        this.mIsSupportTrikey = file.exists() ? file.isDirectory() : false;
        this.mInitTrikey = true;
        Slog.i(TAG, "isSupportTrikey is:" + String.valueOf(this.mIsSupportTrikey));
    }
    return this.mIsSupportTrikey;
}
```

P10支持情况

- 不支持TriKey
- ro.config.hw_front_fp_navi 配置为true
- ro.config.hw_front_fp_trikey 配置为 0

InputMethodManagerService.java

- 输入法管理，以是否启用导航栏进行区分

```
private void updateSystemUiLocked(IBinder token, int vis, int backDisposition) {
    try {
        boolean isEnableNavBar = System.getIntForUser(this.mContext.getContentResolver(),
"enable_navbar", getNaviBarEnabledDefValue(), -2) != 0;
        if (!(this.mNotificationManager == null || (this.mIWindowManager.hasNavigationBar() &&
isEnableNavBar))) {
            this.mNotificationManager.notifyAsUser(null, 17040399,
this.mImeSwitcherNotification.build(), UserHandle.ALL);
            this.mNotificationShown = true;
        }
    } catch (RemoteException e) {
    }
}
```

KeyEvent.java

```
public static final int FINGERPRINT_APP_MAX_KEYCODE = 600;
public static final int FINGERPRINT_DOUBLE_TAP = 501;
public static final int FINGERPRINT_SINGLE_TAP = 601;
public static final int FLAG_FROM_FINGERPRINT = 2048;
public static final int KEYCODE_FINGERPRINT_DOWN = 512;
public static final int KEYCODE_FINGERPRINT_LEFT = 513;
public static final int KEYCODE_FINGERPRINT_LONGPRESS = 502;
public static final int KEYCODE_FINGERPRINT_RIGHT = 514;
public static final int KEYCODE_FINGERPRINT_UP = 511;
```