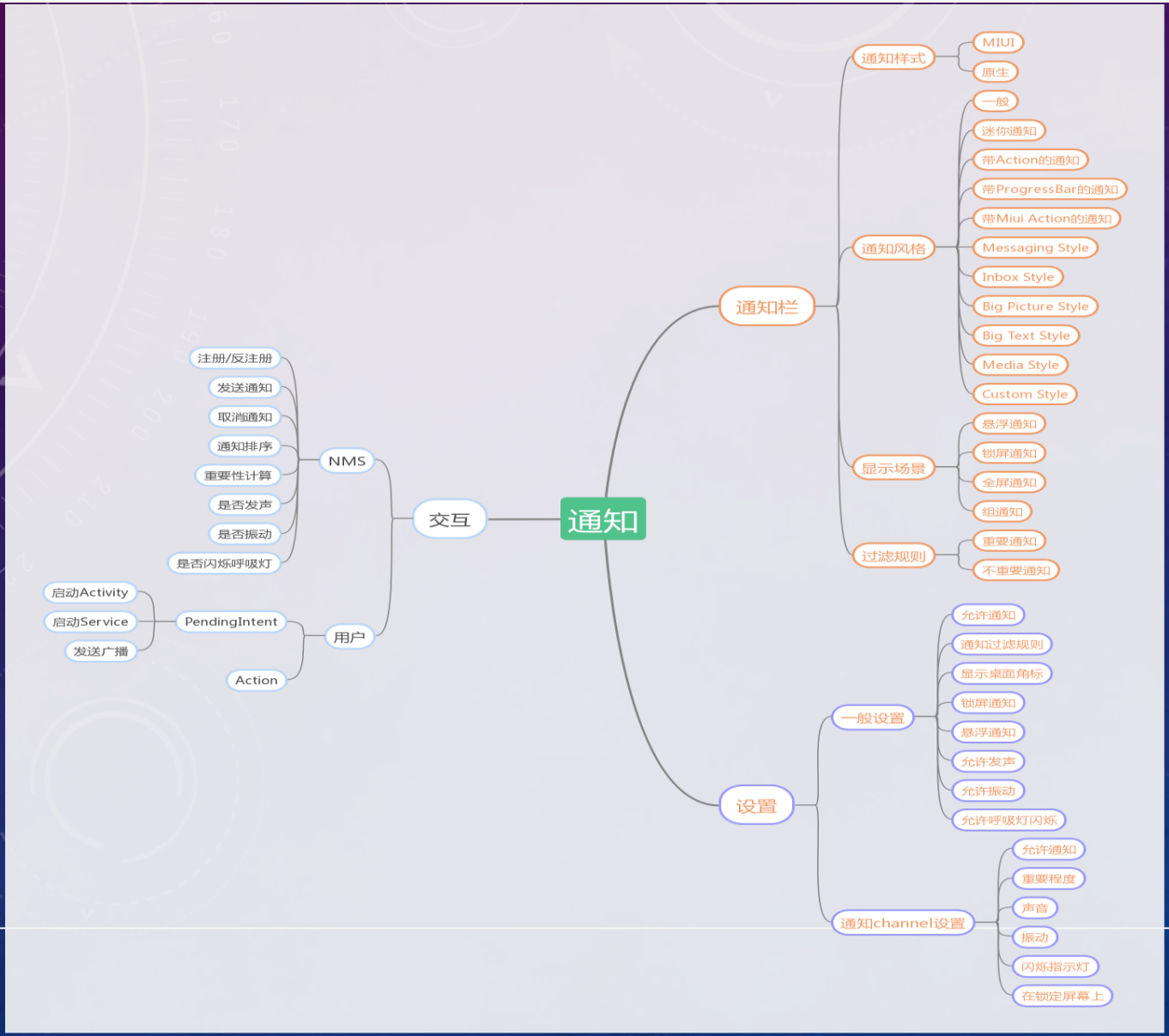
浅析通知机制

通知概览


通知
通知栏
通知样式
MIUI
原生
通知风格
一般
迷你通知
带Action的通知
带ProgressBar的通知
带Miui Action的通知
Messaging Style
Inbox Style
Big Picture Style
Big Text Style
Media Style
Custom Style
显示场景
悬浮通知
锁屏通知
全屏通知
组通知
过滤规则
重要通知
不重要通知
设置
一般设置
允许通知
通知过滤规则
显示桌面角标
锁屏通知
悬浮通知
允许发声
允许振动
允许呼吸灯闪烁
通知channel设置
允许通知
重要程度
声音
振动
闪烁指示灯
在锁定屏幕上
交互
NMS
注册/反注册
发送通知
取消通知
通知排序
重要性计算
是否发声
是否振动
是否闪烁呼吸灯
用户
PendingIntent
启动Activity
启动Service
发送广播
Action

通知样式

MIUI:

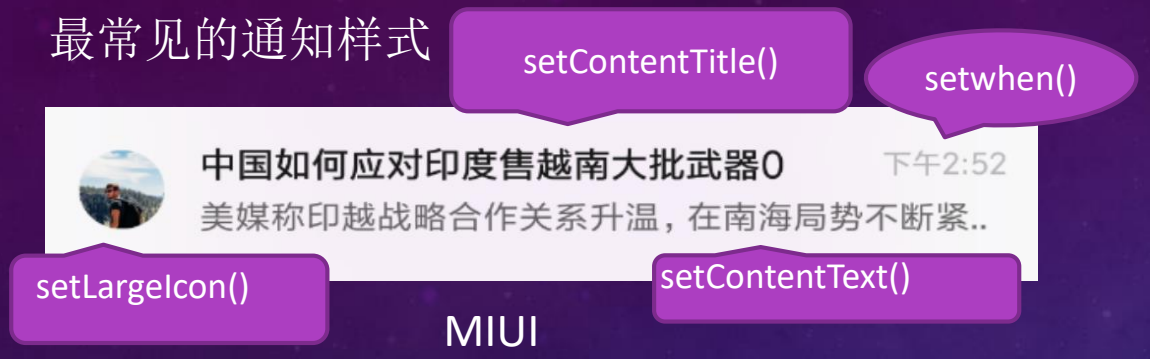

21:12
12月24日 周二
2.3K/s 38%
今日:65.5MB 本月:8.59GB 设置套餐 >
@金州勇士 20:29
斯佩尔曼逼真假动作 http://t...(视频)>>
耗电Battle 21:08
不服来战
没错，我就是白胡子老头 20:23
我礼物准备好了，你袜子呢！
【包邮通知】小米包邮清单来喽！ 20:19
电动牙刷99元3支包邮 >
不重要通知

原生：

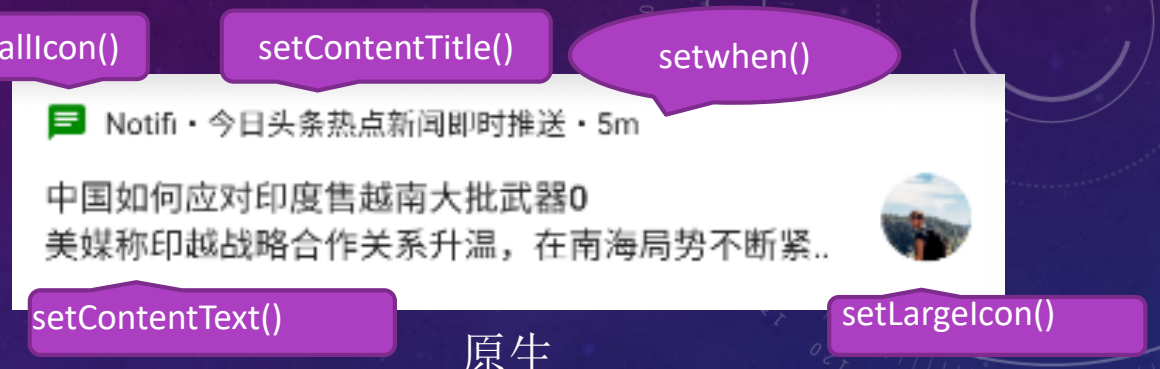

21:12
12月24日 周二
0.0K/s 38%
今日:65.5MB 本月:8.59GB 设置套餐 >
微博 · 42 分钟
@金州勇士
斯佩尔曼逼真假动作 http://t...(视频)>>
小米社区 · 3 分钟
耗电Battle
不服来战
QQ音乐 · 49 分钟
没错，我就是白胡子老头
我礼物准备好了，你袜子呢！
小米商城 · 53 分钟
【包邮通知】小米包邮清单来喽！
电动牙刷99元3支包邮 >
不重要通知

普通通知样式

最常见的通知样式



MIUI



原生

创建方式：

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        // Set the intent that will fire when the user taps the notification
        .setContentIntent(pendingIntent)
        .setAutoCancel(true);
```

迷你通知样式

条件：channel importance < IMPORTANCE_LOW

MIUI

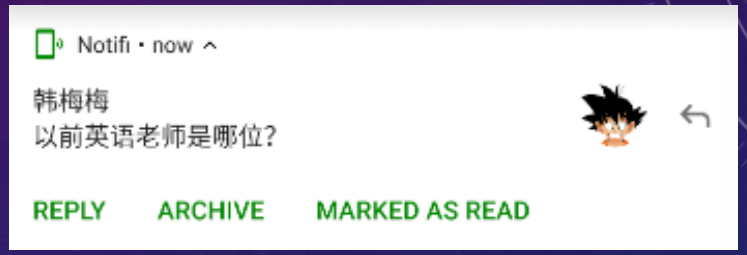
原生

创建方式：

```
NotificationChannel mChannel = new NotificationChannel(channelId, channelName, NotificationManager.IMPORTANCE_MIN);
```

带ACTION的通知

MIUI

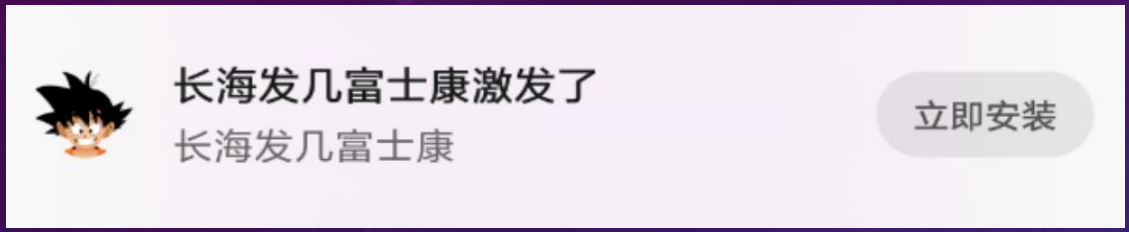


原生

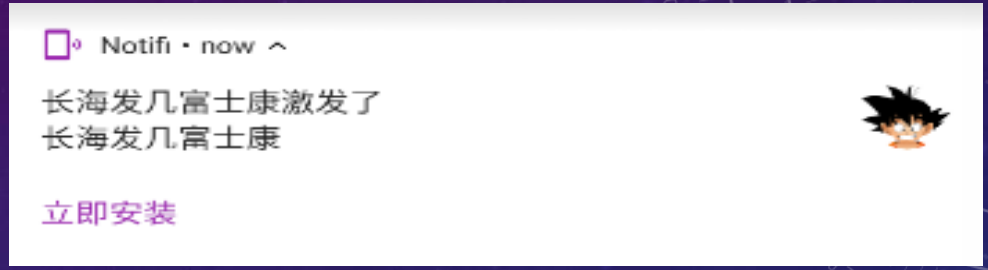
创建方式:

```
Notification.Builder.addAction(replyAction);
```

带MIUI ACTION的通知



MIUI

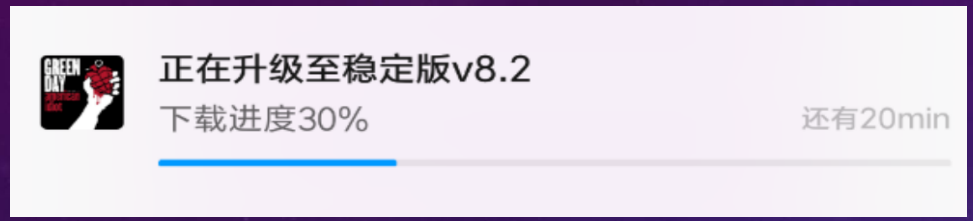


原生

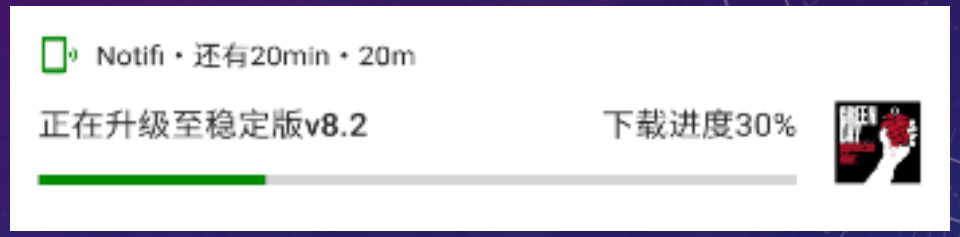
创建方式：

```
Bundle bundle = new Bundle();
bundle.putBoolean("miui.showAction", true);
Notification.Builder.addAction(R.drawable.ic_phonelink_ring_primary_24dp, "立即安装", pendingIntent).setExtras(bundle);
```

带PROGRESS BAR的通知



MIUI

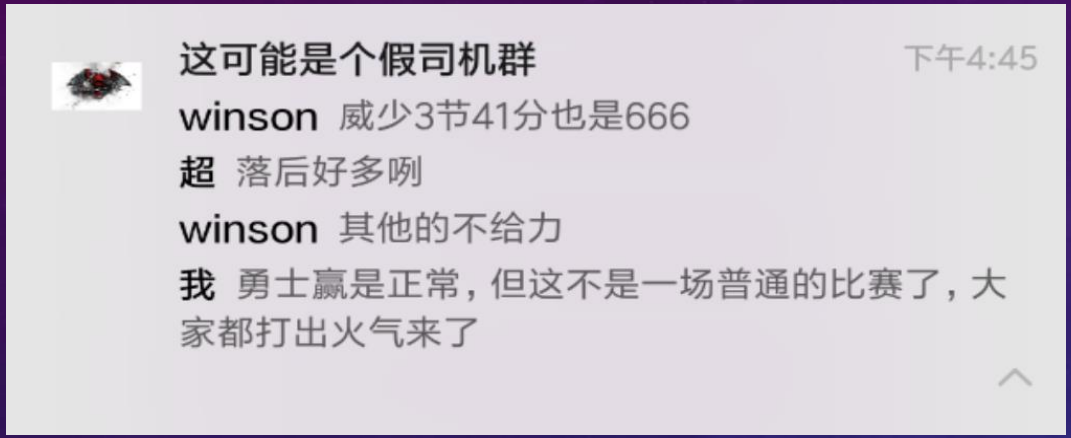


原生

创建方式：

```
Notification.Builder.setProgress(100, 30, false).setSubText("还有20min")
```

MESSAGING STYLE



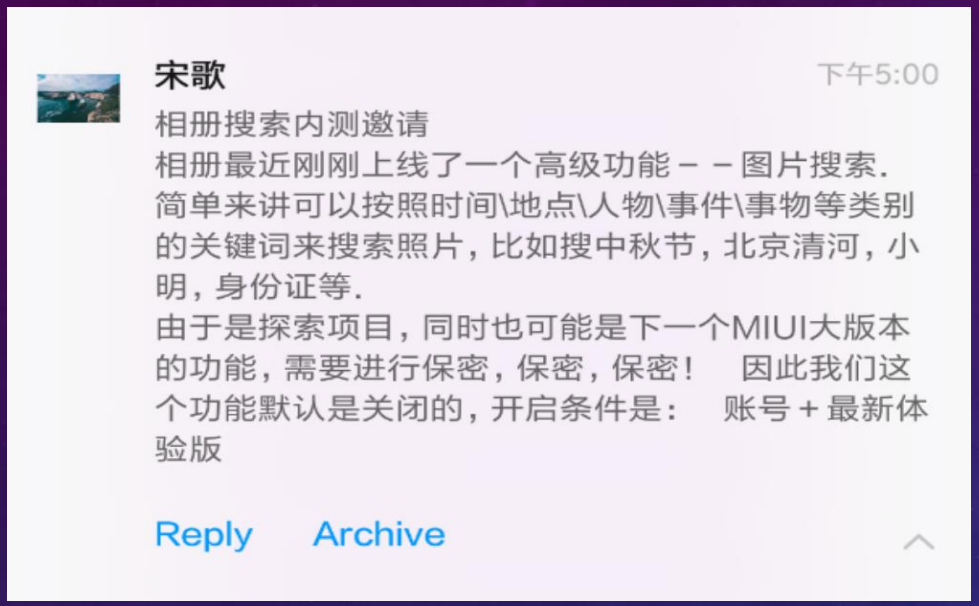
MIUI

原生

创建方式：

```
Notification.Builder.setStyle(new MessagingStyle("我")
    .setConversationTitle("这可能是个假司机群")
    .addMessage("威少3节41分也是666", 1, "winson")
    .addMessage("落后好多咧", 2, "超")
    .addMessage("其他的不给力", 3, "winson")
    .addMessage("勇士赢是正常，但这不是一场普通的比赛了，大家都打出火气来了", 4, "我"))
```

BIG TEXT STYLE



MIUI

原生

创建方式：

```
Notification.Builder.setStyle(new Notification.BigTextStyle()
        .setSummaryText("这是什么")
        .bigText("相册搜索内测邀请 \n" +
        "相册最近刚刚上线了一个高级功能－－图片搜索. \n" +
        "简单来讲可以按照时间\\地点\\人物\\事件\\事物等类别的关键词来搜索照片，比如搜中秋节，北京清河，小明，身份证等. \n" +
        "由于是探索项目，同时也可能是下一个MIUI大版本的功能，需要进行保密，保密，保密！ 因此我们这个功能默认是关闭的，开启条件是： 账号＋最新体验
        "现在邀请大家参加内测，想尝鲜的同学请到 http://wiki.n.miui.com/pages/viewpage.action?pageId=31002591 这个网页下write a comment，格式
```

BIG PICTURE STYLE

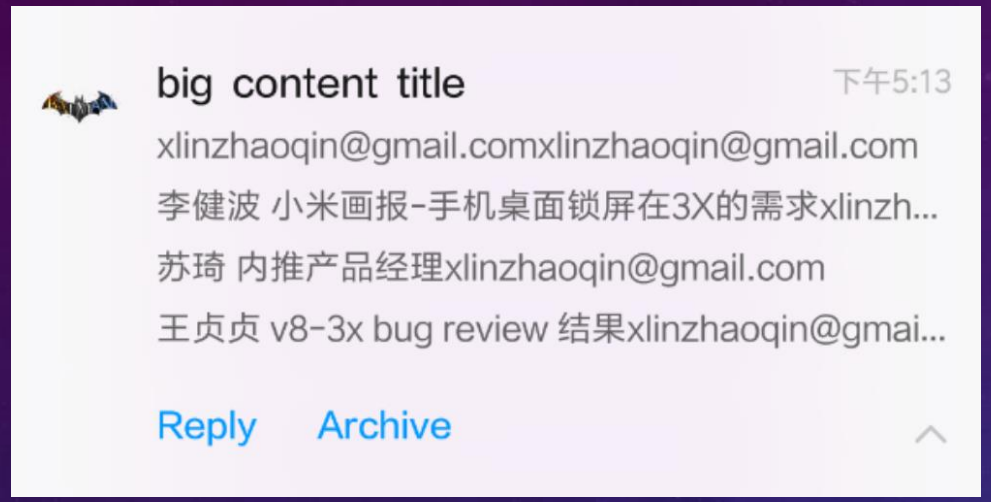
MIUI

原生

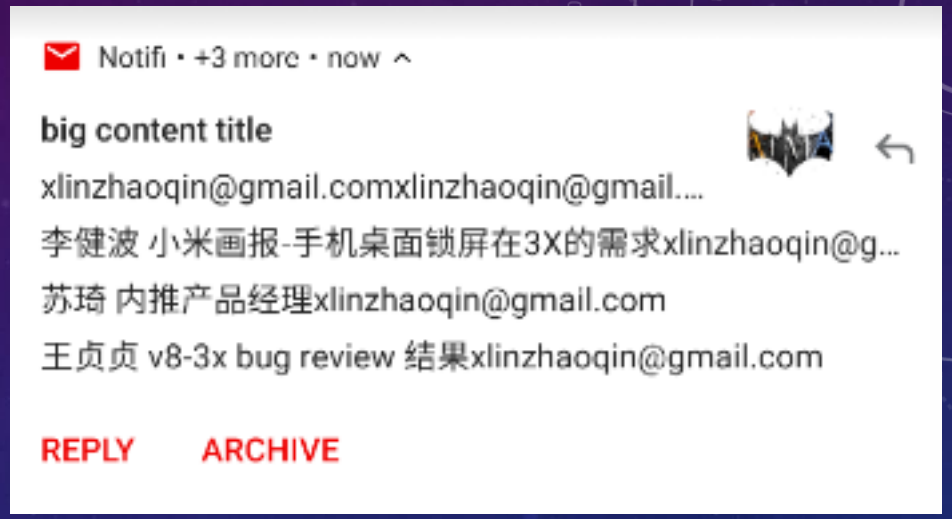
创建方式：

```
Notification.Builder.setStyle(new Notification.BigPictureStyle().bigPicture(bigPicture))
```

INBOX STYLE



MIUI



原生

创建方式:

```
Notification.Builder.setStyle(new Notification.InboxStyle()
        .addLine("xlinzhaoqin@gmail.comxlinzhaoqin@gmail.com")
        .addLine("李健波 小米画报-手机桌面锁屏在3X的需求xlinzhaoqin@gmail.com")
        .addLine("苏琦 内推产品经理xlinzhaoqin@gmail.com")
        .addLine("王贞贞 v8-3x bug review 结果xlinzhaoqin@gmail.com")
        .setSummaryText("+3 more")
        .setBigContentTitle("big content title")
```

MEDIA STYLE

展开（默认）：

MIUI

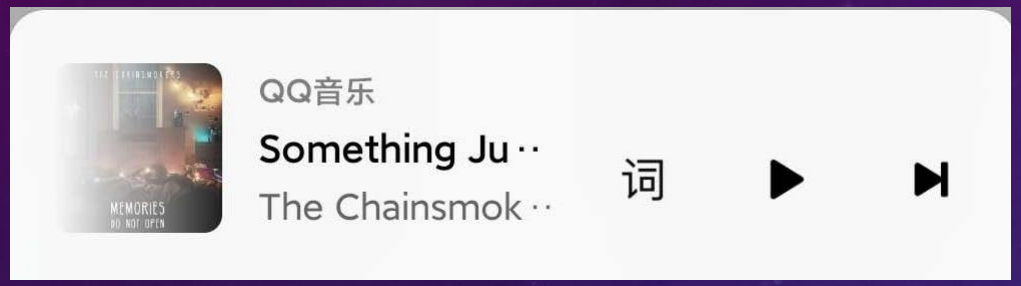
原生

创建方式：

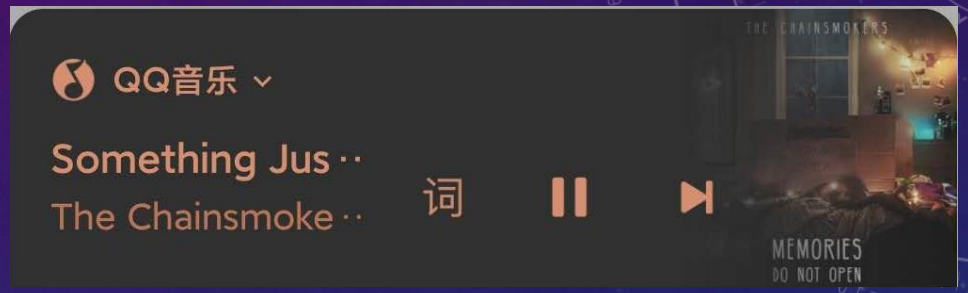
```
Notification.Builder.addAction(previousAction)
.addAction(playAction)
.addAction(nextAction)
.addAction(playlistAction)
.addAction(favoriteAction)
.setStyle(new Notification.MediaStyle().setShowActionsInCompactView(1,2,4))
```

MEDIA STYLE

收起状态：



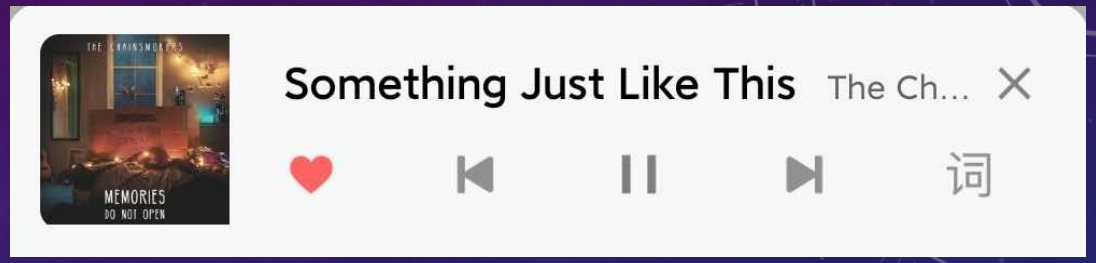
MIUI



原生

CUSTOM STYLE

展开



收起

创建方式：

```
// Get the layouts to use in the custom notification
val notificationLayout = RemoteViews(packageName, R.layout.notification_small)
val notificationLayoutExpanded = RemoteViews(packageName, R.layout.notification_large)

// Apply the layouts to the notification
val customNotification = NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setStyle(NotificationCompat.DecoratedCustomViewStyle())
        .setCustomContentView(notificationLayout)
        .setCustomBigContentView(notificationLayoutExpanded)
        .build()
```

通知创建-范例

```
private void showNotification() {
    Notification.Builder builder = new Notification.Builder(mContext);
    NotificationCompat.setChannelId(builder, NotificationChannels.DND);
    builder.setSmallIcon(com.android.internal.R.drawable.stat_sys_adb)
            .setAutoCancel(false)
            .setGroup(NotificationChannels.DND)
            .setVisibility(Notification.VISIBILITY_PUBLIC)
            .setCustomContentView(new RemoteViews(mContext.getPackageName(), R.layout.dnd_notification))
            .setContentIntent(PendingIntent.getActivity(mContext, 0, generateSilentModeIntent(), 0));
    Notification n = builder.build();
    n.priority = Notification.PRIORITY_MAX;
    MiuiNotificationCompat.setTargetPkg(n, "android");
    MiuiNotificationCompat.setEnableKeyguard(n, true);
    MiuiNotificationCompat.setEnableFloat(n, false);
    MiuiNotificationCompat.setOnlyShowKeyguard(n, true);
    MiuiNotificationCompat.setCustomHeight(n, true);
    MiuiNotificationCompat.setSystemWarnings(n, true);
    mNoMan.notifyAsUser(null, R.string.dnd_notification_warnings_title, n, UserHandle.CURRENT);
}
```

通知创建-剖析

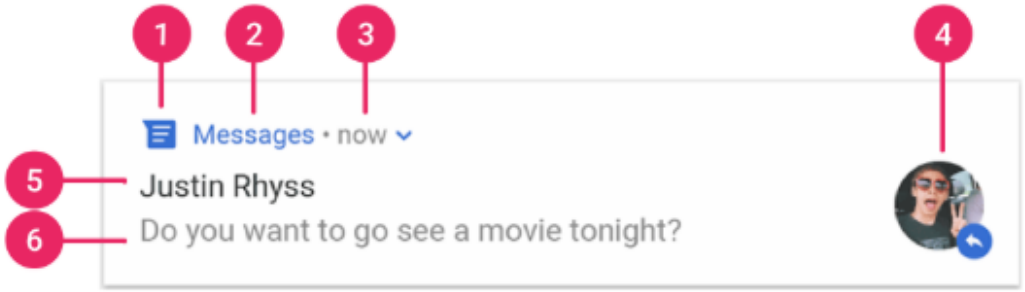


图 7. 包含基本详情的通知

图 7 展示了通知最常见的部分，具体如下所示：

1. 小图标：此为必要图标，通过 `setSmallIcon()` 设置。
2. 应用名称：此由系统提供。
3. 时间戳：此由系统提供，不过您可以通过 `setWhen()` 进行替换，或使用 `setShowWhen(false)` 将其隐藏。
4. 大图标：此为可选图标（通常仅用于联系人照片；请勿将其用于应用图标），通过 `setLargeIcon()` 设置。
5. 标题：此为可选内容，通过 `setContentTitle()` 设置。
6. 文本：此为可选内容，通过 `setContentText()` 设置。

通知创建-FLAGS

Notification.flags

```
public static final int FLAG_ONGOING_EVENT      = 0x00000002;  //常驻通知

public static final int FLAG_INSISTENT          = 0x00000004;  //铃声循环播放

Public static final int FLAG_ONLY_ALERT_ONCE    = 0x00000008;  //只提示一次（振动、铃声）

public static final int FLAG_AUTO_CANCEL        = 0x00000010;  //点击后清除

public static final int FLAG_NO_CLEAR           = 0x00000020;  //清除所有通知时不可清除

public static final int FLAG_FOREGROUND_SERVICE = 0x00000040;  //前台服务通知

public static final int FLAG_GROUP_SUMMARY      = 0x00000200;  //Summary通知

@SystemApi
public static final int FLAG_AUTOGROUP_SUMMARY  = 0x00000400;  //自动分组通知
```

通知创建-MIUI EXTRAS

notification.extras.putXXX():

```
public static final String EXTRA_SHOW_ACTION = “miui.showAction”;    //是否miui action
public static final String EXTRA_ACTION_EXPANDABLE = “miui.actionExpandable”;  //action是否展开
public static final String EXTRA_EXPANDABLE_ON_KEYGUARD = “miui.expandableOnKeyguard”;  //锁屏
可展开
public static final String EXTRA_ENABLE_FLOAT = “miui.enableFloat”;    //是否悬浮
public static final String EXTRA_ENABLE_KEYGUARD = “miui.enableKeyguard”;  //锁屏是否显示
public static final String EXTRA_FLOAT_TIME = “miui.floatTime”;          //悬浮时间
public static final String EXTRA_TARGET_PKG = “miui.targetPkg”; //替换包名，实现模拟应用发通知的消息
效果（如小米推送、SystemUI、）
public static final String EXTRA_SUBSTNAME = “miui.substName”; //快应用App名称
public static final String EXTRA_MESSAGE_COUNT = “miui.messageCount”;  //当前通知代表的信息数，会显
示在桌面icon角标
public static final String EXTRA_MESSAGE_CLASSNAME = “miui.messageClassName”; //对于一个app对应多
个桌面图标的情况，需要指定classname
public static final String EXTRA_ONLY_SHOW_KEYGUARD = “miui.onlyShowKeyguard”; //只显示在锁屏
public static final String EXTRA_KEPT_ON_KEYGUARD = “miui.keptOnKeyguard”;  //锁屏常驻
public static final String EXTRA_CUSTOM_HEIGHT = “miui.customHeight”;    //自定义通知高度
public static final String EXTRA_SYSTEM_WARNINGS = “miui.systemWarnings”; //系统警告通知，排序在最
前
public static final String EXTRA_SHOW_AT_TAIL = “miui.showAtTail”;    //在通知栏末尾显示，排序在最后
public static final String EXTRA_IS_PERSISTENT = “miui.isPersistent”; //该通知禁止滑动操作
public static final String EXTRA_NO_CUSTOM_VIEW_DECORATION = “miui.noCustomViewDecoration”; //
取消自定义通知的圆角
```

通知创建-CHANNEL

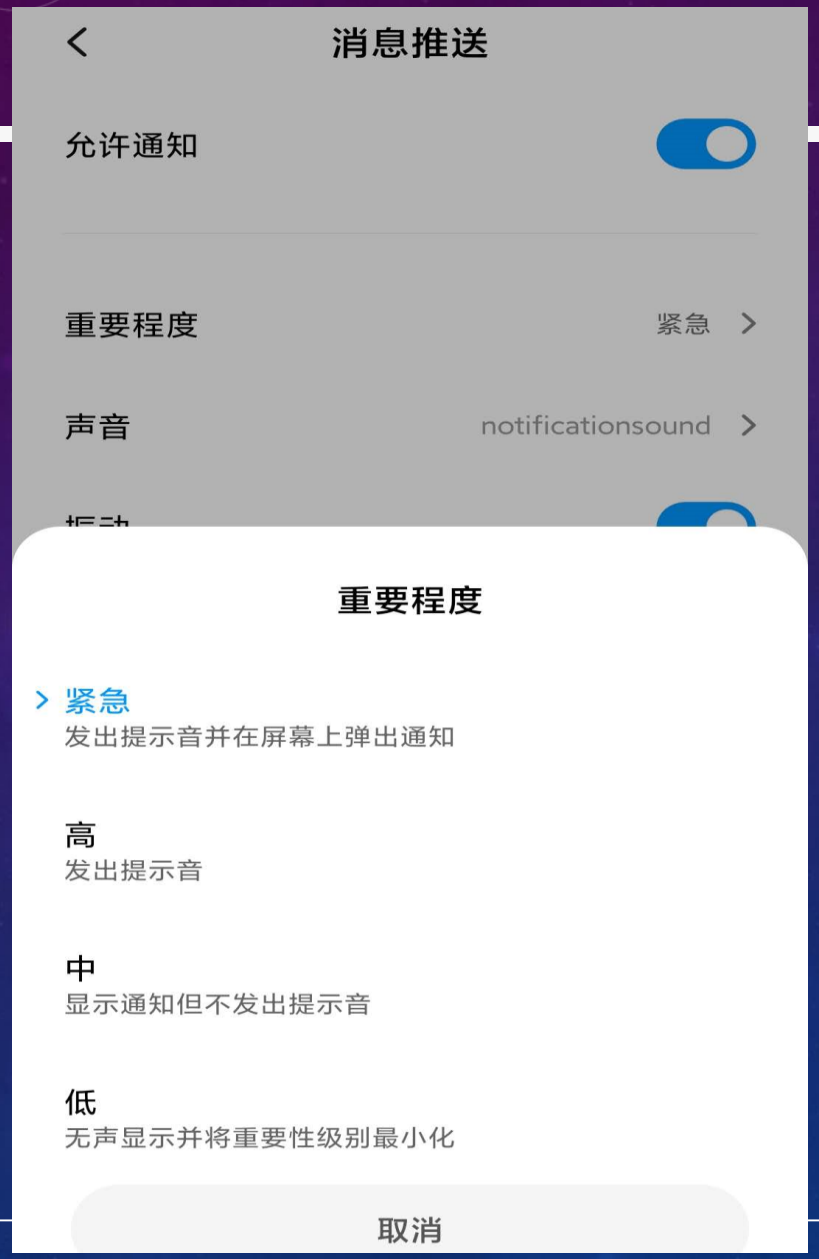
通知Channel：目的是通过对通知进行分类，达到对通知的精细化控制，避免通知滥用导致泛滥

创建Channel：

```
@TargetApi(Build.VERSION_CODES.O)
private void createChannel(Context context, String channelId, String channelName, Uri soundUri, boolean shouldVibration) {
    String description = context.getString(R.string.notification_channel_description);
    int importance = NotificationManager.IMPORTANCE_HIGH;
    NotificationChannel channel = new NotificationChannel(channelId, channelName, importance);
    channel.setDescription(description);
    if (soundUri == null) {
        channel.setSound(null, null);
    } else {
        channel.setSound(soundUri, Notification.AUDIO_ATTRIBUTES_DEFAULT);
    }
    channel.enableLights(true);
    channel.enableVibration(shouldVibration);
    // 创建channel
    mNotificationManager.createNotificationChannel(channel);
}
```

在targetSdk<26时，NMS会自动给通知加上id叫” miscellaneous”的channel

通知重要性


消息推送
允许通知
重要程度
紧急
声音
notificationsound
重要程度
紧急
发出提示音并在屏幕上弹出通知
高
发出提示音
中
显示通知但不发出提示音
低
无声显示并将重要性级别最小化
取消

通知重要性

< Android O:
通知重要性是通过通知优先级体现，通过notificaiton.priority来设置
PRIORITY_DEFAULT=0；
PRIORITY_LOW=-2；
PRIORITY_MIN=-1；
PRIORITY_HIGH=1；
PRIORITY_MAX=2；

>= Android O:
通知重要性是由channel的importance来体现，通过channel.setImportance()来设置

```
public static final int IMPORTANCE_UNSPECIFIED = -1000; //未知，不会用于最终的通知重要性
public static final int IMPORTANCE_NONE = 0;      // 无重要性，不会显示在通知栏
public static final int IMPORTANCE_MIN = 1;       // 不重要，会被折叠
public static final int IMPORTANCE_LOW = 2;       // 低，不提示
public static final int IMPORTANCE_DEFAULT = 3;   //提示，但不悬浮
public static final int IMPORTANCE_HIGH = 4;      //提示且悬浮
public static final int IMPORTANCE_MAX = 5;       //未用
```

通知重要性计算-1

```
// Migrate notification priority flag to a priority value.
if (0 != (n.flags & Notification.FLAG_HIGH_PRIORITY)) {
    n.priority = Notification.PRIORITY_MAX;
}

// Convert priority value to an importance value, used only for pre-channels notifications.
int requestedImportance = IMPORTANCE_DEFAULT;
n.priority = NotificationManagerService.clamp(n.priority, Notification.PRIORITY_MIN,
        Notification.PRIORITY_MAX);
switch (n.priority) {
    case Notification.PRIORITY_MIN:
        requestedImportance = IMPORTANCE_MIN;
        break;
    case Notification.PRIORITY_LOW:
        requestedImportance = IMPORTANCE_LOW;
        break;
    case Notification.PRIORITY_DEFAULT:
        requestedImportance = IMPORTANCE_DEFAULT;
        break;
    case Notification.PRIORITY_HIGH:
    case Notification.PRIORITY_MAX:
        requestedImportance = IMPORTANCE_HIGH;
        break;
}
```

通知重要性计算-2

```
stats.isNoisy = mSound != null || mVibration != null;

// For pre-channels notifications, apply system overrides and then use requestedImportance
// as importance.
if (mPreChannelsNotification
        && (importance == IMPORTANCE_UNSPECIFIED
        || (!getChannel().hasUserSetImportance()))) {
    if (!stats.isNoisy && requestedImportance > IMPORTANCE_LOW) {
        requestedImportance = IMPORTANCE_LOW;
    }

    if (stats.isNoisy) {
        if (requestedImportance < IMPORTANCE_DEFAULT) {
            requestedImportance = IMPORTANCE_DEFAULT;
        }
    }

    if (n.fullScreenIntent != null) {
        requestedImportance = IMPORTANCE_HIGH;
    }
    importance = requestedImportance;
    mInitialImportanceExplanationCode =
            MetricsEvent.IMPORTANCE_EXPLANATION_APP_PRE_CHANNELS;
}
```

通知振动

< Android O: n.vibrate(new long[]{200, 300, 200})设置
>= Android O: channel.enableVibrate()，channel.setVibrationPattern()来设置，默认开启

通过调用VibratorService来实现振动效果，如wave、oneshot等

```
private long[] calculateVibration() {
    long[] vibration;
    final long[] defaultVibration =  NotificationManagerService.getLongArray(
            mContext.getResources(),
            com.android.internal.R.array.config_defaultNotificationVibePattern,
            NotificationManagerService.VIBRATE_PATTERN_MAXLEN,
            NotificationManagerService.DEFAULT_VIBRATE_PATTERN);
    if (getChannel().shouldVibrate()) {
        vibration = getChannel().getVibrationPattern() == null
                ? defaultVibration : getChannel().getVibrationPattern();
    } else {
        vibration = null;
    }
    if (mPreChannelsNotification
            && (getChannel().getUserLockedFields()
            & NotificationChannel.USER_LOCKED_VIBRATION) == 0) {
        final Notification notification = sbn.getNotification();
        final boolean useDefaultVibrate =
                (notification.defaults & Notification.DEFAULT_VIBRATE) != 0;
        if (useDefaultVibrate) {
            vibration = defaultVibration;
        } else {
            vibration = notification.vibrate;
        }
    }
    return vibration;
}
```

通知铃声

< Android O:通过n.sound来设置声音Uri

>= Android O:通过channel.enableSound()和channel.setSound()来设置，默认开关关闭

通过调用Systemui的NotificationPlayer播放铃声

```
private Uri calculateSound() {
    final Notification n = sbn.getNotification();

    // No notification sounds on tv
    if (mContext.getPackageManager().hasSystemFeature(PackageManager.FEATURE_LEANBACK)) {
        return null;
    }

    Uri sound = mChannel.getSound();
    if (mPreChannelsNotification && (getChannel().getUserLockedFields()
            & NotificationChannel.USER_LOCKED_SOUND) == 0) {

        final boolean useDefaultSound = (n.defaults & Notification.DEFAULT_SOUND) != 0;
        if (useDefaultSound) {
            sound = Settings.System.DEFAULT_NOTIFICATION_URI;
        } else {
            sound = n.sound;
        }
    }
    return sound;
}
```

通知呼吸灯闪烁

< Android O:通过n.ledARGB来设置声音Uri

>= Android O:通过channel.enableLights()和channel.setLightColor()来设置，默认开关关闭

通过调用LightService来让呼吸灯闪烁

```
private Light calculateLights() {
    int defaultLightColor = mContext.getResources().getColor(
            com.android.internal.R.color.config_defaultNotificationColor);
    int defaultLightOn = mContext.getResources().getInteger(
            com.android.internal.R.integer.config_defaultNotificationLedOn);
    int defaultLightOff = mContext.getResources().getInteger(
            com.android.internal.R.integer.config_defaultNotificationLedOff);

    int channelLightColor = getChannel().getLightColor() != 0 ? getChannel().getLightColor()
            : defaultLightColor;
    Light light = getChannel().shouldShowLights() ? new Light(channelLightColor,
            defaultLightOn, defaultLightOff) : null;
    if (mPreChannelsNotification
            && (getChannel().getUserLockedFields()
            & NotificationChannel.USER_LOCKED_LIGHTS) == 0) {
        final Notification notification = sbn.getNotification();
        if ((notification.flags & Notification.FLAG_SHOW_LIGHTS) != 0) {
            light = new Light(notification.ledARGB, notification.ledOnMS,
                    notification.ledOffMS);
            if ((notification.defaults & Notification.DEFAULT_LIGHTS) != 0) {
                light = new Light(defaultLightColor, defaultLightOn,
                        defaultLightOff);
            }
        } else {
            light = null;
        }
    }
    return light;
}
```

通知折叠

MIUI9引入的，将重要性较低的通知折叠起来，突出重要性较高的通知

1、通知过滤开关关闭不折叠
2、不可清除通知不折叠
3、过滤规则改为默认以下折叠
4、summary通知不折叠
5、重要性等于MIN折叠
6、打分结果决定是否折叠

```
private boolean calculateFold() {
    if (!NotificationUtil.isUserFold()) {
        return false;
    }
    // never fold a non-clearable notification cause we suppose it was important.
    if (!isClearable()) {
        return false;
    }
    // 通知过滤规则，即手动设置为"重要"和"不重要"
    if (mFoldImportance != FOLD_IMPORTANCE_DEFAULT) {
        return mFoldImportance < FOLD_IMPORTANCE_DEFAULT;
    }
    // never fold a summary notification
    if (getNotification().isGroupSummary()) {
        return false;
    }
    // O以上Channel通知重要程度
    if (mImportance == IMPORTANCE_MIN) {
        return true;
    } else if (mImportance >= IMPORTANCE_HIGH) {
        return false;
    }
    if (getNotification().priority == Notification.PRIORITY_MAX
            || mShowSum <= RankUtil.UNFLOD_LIMIT
            || NotificationUtil.isSystemNotification( sbn: this)) {
        return false;
    }
    return mBelowThreshold;
}
```

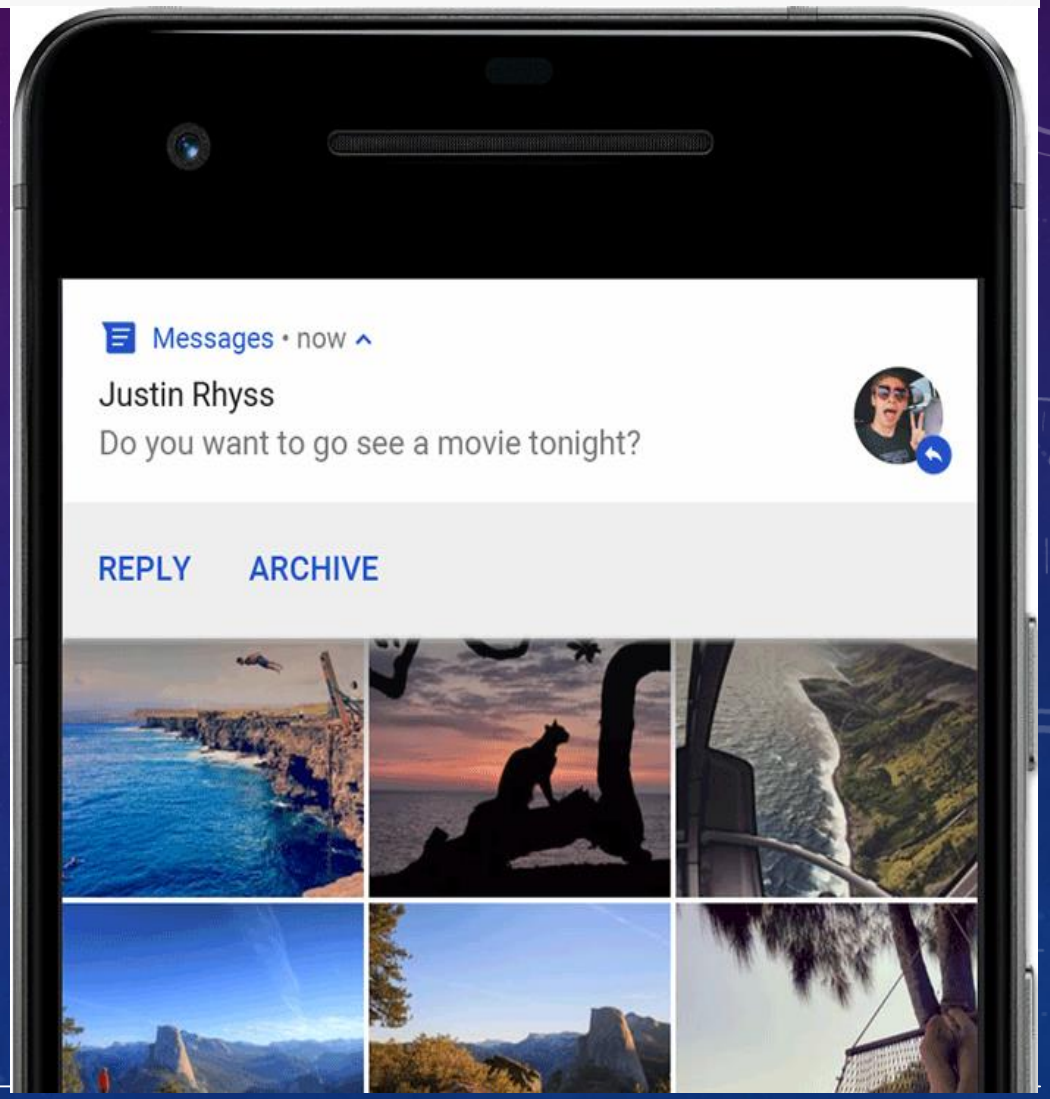
通知显示-悬浮通知

App可通过notification.extraNotification.setEnableFloat设置悬浮
通过notification.extraNotificaiton.setFloatTime()来设置悬浮时间

条件(太多，几十个):

- 是一条组通知且app指定了不允许组通知悬浮时
- 来电通知正在悬浮时
- 通话页面在前台且不是一条视频来电通知时
- 不重要通知

白名单控制：config_allowFloatPackages

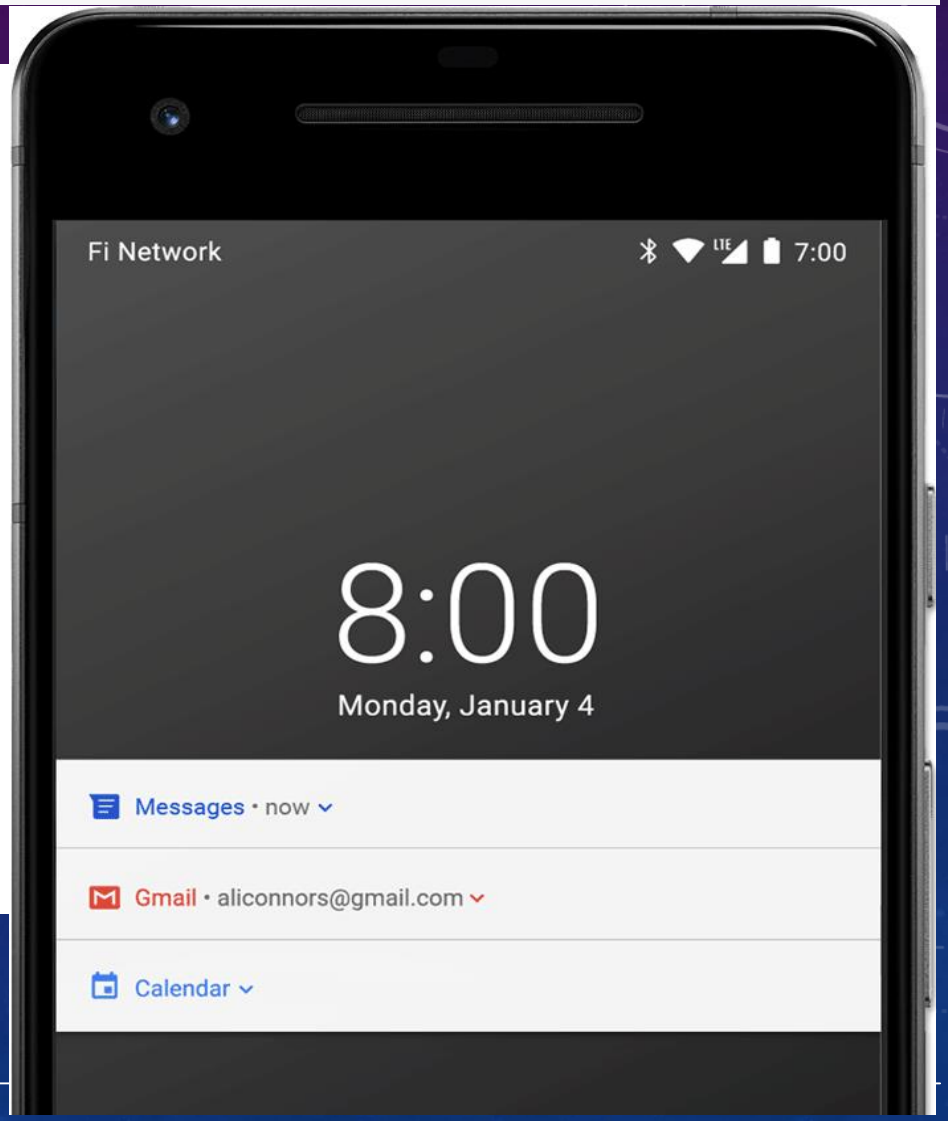


通知类型-锁屏通知

通知锁屏显示策略

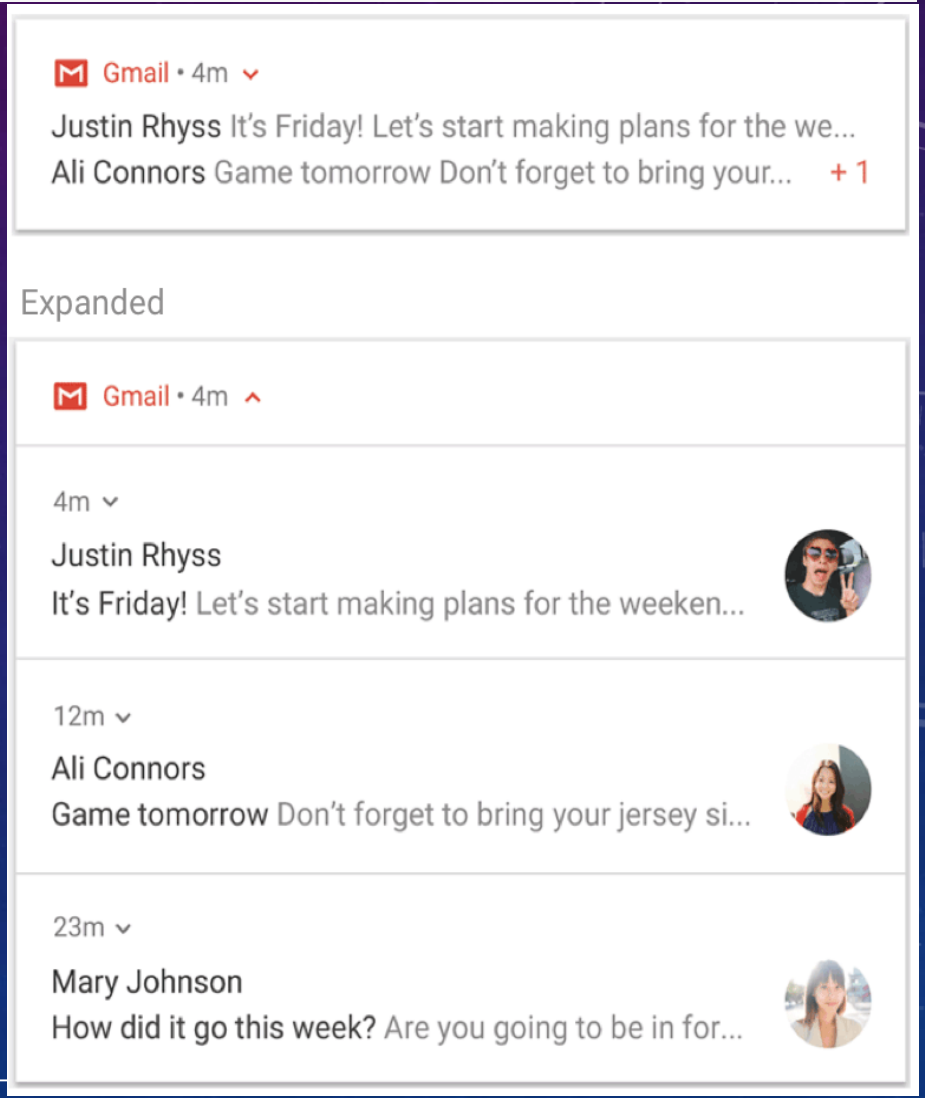
同时满足以下多个条件时，以靠前的判断逻辑为准：

- 设置-锁屏、密码和指纹-锁屏高级设置中，"屏幕锁定时"选择"完全不显示通知"时，锁屏不显示通知
- 迷你通知不允许锁屏显示
- 不重要通知不允许锁屏显示
- 媒体通知允许锁屏显示
- cts test apk发出的通知允许锁屏显示
- 解锁后在通知栏出现过的通知，锁屏后不在锁屏上显示
- ongoing通知不允许锁屏显示
- 单独的一条auto group summary不在锁屏上显示
- app通知设置页关闭掉"锁屏通知"开关时不在锁屏显示
- MiuiNotification.setEnableKeyguard(false)时不允许锁屏显示



通知显示-组通知

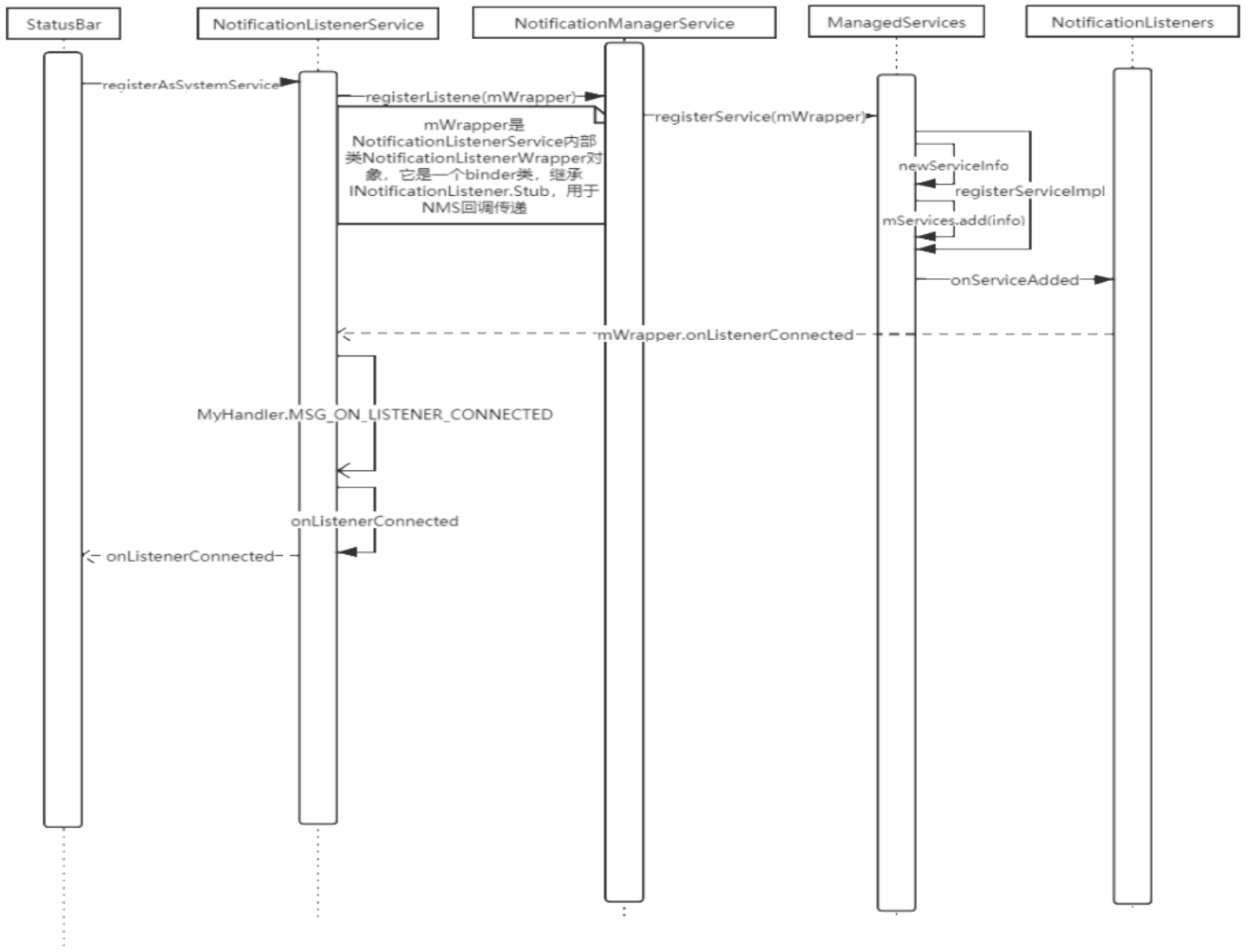
- app发通知时可以指定group summary和group key，同一个group key的通知将被分到一组
- group summary也是一条通知，有child的group summary的内容将被隐藏，单纯作为组通知的容器
- 点击组通知会使其展开，但如果app给group summary设置了click intent，点击组通知时会send click intent而不会展开
- 如果app没有指定group summary，通知不会被分组，指定了group key的通知也不会被系统自动分组
- 当通知栏中某个app的通知大于等于4条时，系统将给此app fake一个group summary(id="ranker_group")，强制让其分组



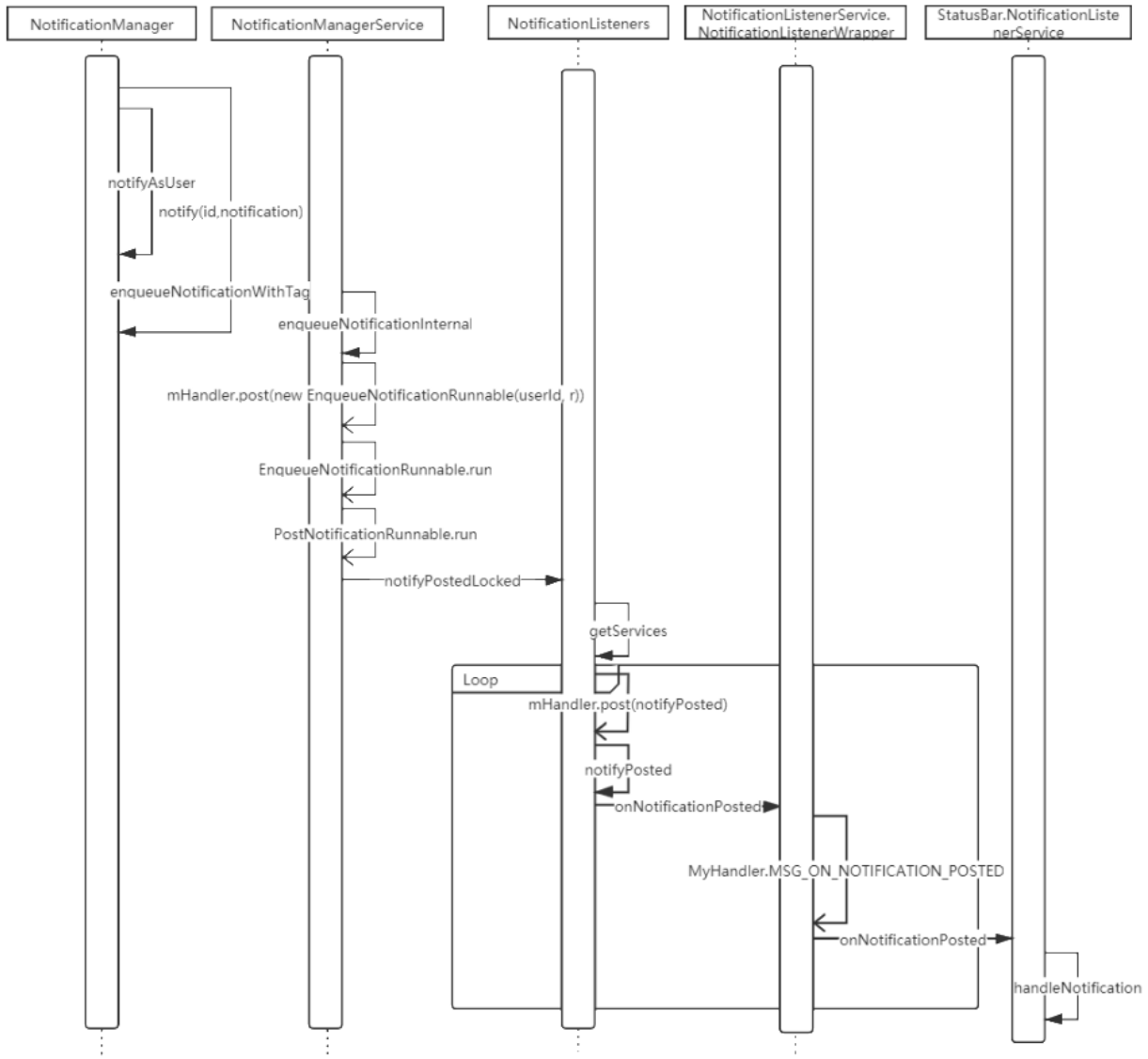
通知显示-带有FULLSCREENINTENT的通知

- Notification.setFullscreenIntent()设置
- 自动提升重要性至紧急（IMPORTANCE_HIGH）

通知发送流程-1


StatusBar
NotificationListenerService
NotificationManagerService
ManagedServices
NotificationListeners
registerAsSystemService
registerListene(mWrapper)
mWrapper是NotificationListenerService内部类NotificationListenerWrapper对象，它是一个binder类，继承INotificationListener.Stub，用于NMS回调传递
registerService(mWrapper)
newServiceInfo
registerServiceImpl
mServices.add(info)
onServiceAdded
mWrapper.onListenerConnected
MyHandler.MSG_ON_LISTENER_CONNECTED
onListenerConnected
onListenerConnected

通知发送流程-2


NotificationManager
NotificationManagerService
NotificationListeners
NotificationListenerService.
NotificationListenerWrapper
StatusBar.NotificationListe
nerService
notifyAsUser
notify(id,notification)
enqueueNotificationWithTag
enqueueNotificationInternal
mHandler.post(new EnqueueNotificationRunnable(userId, r))
EnqueueNotificationRunnable.run
PostNotificationRunnable.run
notifyPostedLocked
getServices
Loop
mHandler.post(notifyPosted)
notifyPosted
onNotificationPosted
MyHandler.MSG_ON_NOTIFICATION_POSTED
onNotificationPosted
handleNotification

通知日志分析-1

Dump SystemUI： adb shell dumpsys activity service SystemUIService
Dump NMS： adb shell dumpsys notification

查找App通知设置: 关键字”AppNotificationSettings”

悬浮通知白名单: 关键字”float_whitelist”

锁屏白名单: 关键字”keyguard_whitelist”

过滤通知发送： adb logcat -b all | grep -aE ‘notification_enqueue’
过滤通知提醒： adb logcat -b all | grep -aE ‘notification_alert’
过滤通知曝光： adb logcat -b all | grep -aE ‘notification_visibility’
过滤通知点击： adb logcat -b all | grep -aE ‘notification_clicked’
过滤通知清除： adb logcat -b all | grep -aE ‘notification_canceled’
过滤全屏通知发送： adb logcat -b all | grep -aE ‘sysui_fullscreen_notification’

更多： https://wiki.n.miui.com/pages/viewpage.action?pageId=129268023

引用

https://wiki.n.miui.com/pages/viewpage.action?pageId=129743781

https://developer.android.com/guide/topics/ui/notifiers/notifications?hl=zh-cn

感谢老板们