通知

本文内容

- 设计注意事项
- 创建通知
  - 必需的通知内容
  - 可选通知内容和设置
  - 通知操作
  - 通知优先级
  - 创建简单通知
  - 将扩展布局应用于通知
  - 处理兼容性
- 管理通知
  - 更新通知
  - 删除通知
- 启动 Activity 时保留导航
  - 设置常规 Activity PendingIntent
  - 设置特殊 Activity PendingIntent
- 在通知中显示进度
  - 显示持续时间固定的进度指示器
  - 显示持续 Activity 指示器
- 通知元数据
- 浮动通知
- 锁定屏幕通知
- 设置可见性
- 在锁定屏幕上控制媒体播放
- 自定义通知布局

关键类

- `NotificationManager`
- `NotificationCompat`

视频

- 4.1 中的通知

另请参阅

- Android 设计：通知

通知是您可以在应用的常规 UI 外部向用户显示的消息。当您告知系统发出通知时，它将先以图标的形式显示在**通知区域**中。用户可以打开**抽屉式通知栏**查看通知的详细信息。 通知区域和抽屉式通知栏均是由系统控制的区域，用户可以随时查看。

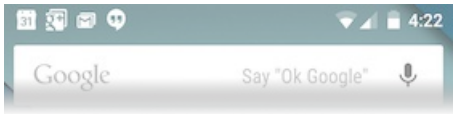

图 1. 通知区域中的通知。

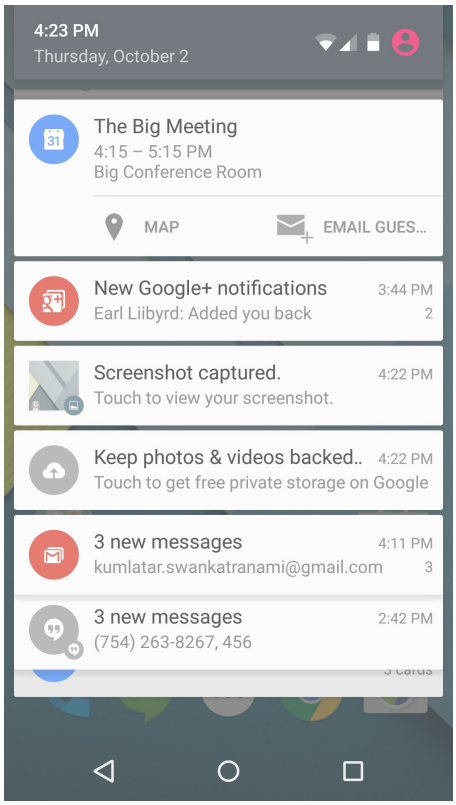

图 2. 抽屉式通知栏中的通知。

> **注：**除非特别注明，否则本指南均引用版本 4 支持库中的 `NotificationCompat.Builder` 类。Android 3.0（API 级别 11）中已添加类 `Notification.Builder`。

设计注意事项

作为 Android 用户界面的一个重要组成部分，通知具有自己的设计指导方针。Android 5.0（API 级别 21）中引入的 Material Design 变更尤为重要，您应查阅Material Design培训资料了解详细信息。 要了解如何设计通知及其交互，请阅读通知设计指南。

创建通知

您可以在 `NotificationCompat.Builder` 对象中为通知指定 UI 信息和操作。要创建通知，请调用 `NotificationCompat.Builder.build()`，它将返回包含您的具体规范的 `Notification` 对象。要发出通知，请通过调用 `NotificationManager.notify()` 将 `Notification` 对象传递给系统。

必需的通知内容

`Notification` 对象必须包含以下内容：

- 小图标，由 `setSmallIcon()` 设置
- 标题，由 `setContentTitle()` 设置
- 详细文本，由 `setContentText()` 设置

可选通知内容和设置

所有其他通知设置和内容都是可选的。如需了解有关它们的更多详情，请参阅 `NotificationCompat.Builder` 参考文档。

通知操作

尽管通知操作都是可选的，但是您至少应向通知添加一个操作。 操作允许用户直接从通知转到应用中的 `Activity`，他们可在其中查看一个或多个事件或执行进一步的操作。

一个通知可以提供多个操作。您应该始终定义一个当用户点击通知时会触发的操作；通常，此操作会在应用中打开 `Activity`。 您也可以向通

知添加按钮来执行其他操作，例如，暂停闹铃或立即答复短信；此功能自 Android 4.1 起可用。如果使用其他操作按钮，则您还必须使这些按钮的功能在应用的 `Activity` 中可用；请参阅**处理兼容性**部分，以了解更多详情。

在 `Notification` 内部，操作本身由 `PendingIntent` 定义，后者包含在应用中启动 `Activity` 的 `Intent`。要将 `PendingIntent` 与手势相关联，请调用 `NotificationCompat.Builder` 的适当方法。例如，如果您要在用户点击抽屉式通知栏中的通知文本时启动 `Activity`，则可通过调用 `setContentIntent()` 来添加 `PendingIntent`。

在用户点击通知时启动 `Activity` 是最常见的操作场景。此外，您还可以在用户清除通知时启动 `Activity`。在 Android 4.1 及更高版本中，您可以通过操作按钮启动 `Activity`。如需了解更多信息，请阅读参考指南的 `NotificationCompat.Builder` 部分。

通知优先级

您可以根据需要设置通知的优先级。优先级充当一个提示，提醒设备 UI 应该如何显示通知。 要设置通知的优先级，请调用 `NotificationCompat.Builder.setPriority()` 并传入一个 `NotificationCompat` 优先级常量。有五个优先级别，范围从 `PRIORITY_MIN` (-2) 到 `PRIORITY_MAX` (2)；如果未设置，则优先级默认为 `PRIORITY_DEFAULT` (0)。

有关设置适当优先级别的信息，请参阅**通知**设计指南中的“正确设置和管理通知优先级”。

创建简单通知

以下代码段说明了一个指定某项 Activity 在用户点击通知时打开的简单通知。 请注意，该代码将创建 `TaskStackBuilder` 对象并使用它来为操作创建 `PendingIntent`。**启动 Activity 时保留导航**部分对此模式做了更详尽的阐述：

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

就这么简单。您的用户现已收到通知。

将扩展布局应用于通知

要使通知出现在展开视图中，请先创建一个带有所需普通视图选项的 `NotificationCompat.Builder` 对象。接下来，调用以扩展布局对象作为其参数的 `Builder.setStyle()`。

请记住，扩展通知在 Android 4.1 之前的平台上不可用。要了解如何处理针对 Android 4.1 及更早版本平台的通知，请阅读**处理兼容性**部分。

例如，以下代码段演示了如何更改在前面的代码段中创建的通知，以便使用扩展布局：

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Event tracker")
    .setContentText("Events received")
NotificationCompat.InboxStyle inboxStyle =
        new NotificationCompat.InboxStyle();
String[] events = new String[6];
// Sets a title for the Inbox in expanded layout
inboxStyle.setBigContentTitle("Event tracker details:");
...
// Moves events into the expanded layout
for (int i=0; i < events.length; i++) {

    inboxStyle.addLine(events[i]);
}
// Moves the expanded layout object into the notification object.
mBuilder.setStyle(inBoxStyle);
...
// Issue the notification here.
```

处理兼容性

并非所有通知功能都可用于某特定版本，即便用于设置这些功能的方法位于支持库类 `NotificationCompat.Builder` 中也是如此。 例如，依赖于扩展通知的操作按钮仅会显示在 Android 4.1 及更高版本的系统中，这是因为扩展通知本身仅在 Android 4.1 及更高版本的系统中可用。

为了确保最佳兼容性，请使用 `NotificationCompat` 及其子类（特别是 `NotificationCompat.Builder`）创建通知。此外，在实现通知时，请遵循以下流程：

1. 为所有用户提供通知的全部功能，无论他们使用何种版本的 Android 系统。 为此，请验证是否可从应用的 `Activity` 中获得所有功能。要执行此操作，您可能需要添加新的 `Activity`。

   例如，若要使用 `addAction()` 提供停止和启动媒体播放的控件，请先在应用的 `Activity` 中实现此控件。

2. 确保所有用户均可通过点击通知启动 `Activity` 来获得该Activity中的功能。 为此，请为 `Activity` 创建 `PendingIntent`。调用 `setContentIntent()` 以将 `PendingIntent` 添加到通知。

3. 现在，将要使用的扩展通知功能添加到通知。请记住，您添加的任何功能还必须在用户点击通知时启动的 `Activity` 中可用。

管理通知

当您需要为同一类型的事件多次发出同一通知时，应避免创建全新的通知， 而是应考虑通过更改之前通知的某些值和/或为其添加某些值来更新通知。

例如，Gmail 通过增加未读消息计数并将每封电子邮件的摘要添加到通知，通知用户收到了新的电子邮件。 这称为“堆叠”通知；通知设计指南对此进行了更详尽的描述。

> **注：**此 Gmail 功能需要“收件箱”扩展布局，该布局是自 Android 4.1 版本起可用的扩展通知功能的一部分。

下文介绍如何更新和删除通知。

更新通知

要将通知设置为能够更新，请通过调用 `NotificationManager.notify()` 发出带有通知 ID 的通知。 要在发出之后更新此通知，请更新或创建 `NotificationCompat.Builder` 对象，从该对象构建 `Notification` 对象，并发出与之前所用 ID 相同的 `Notification`。如果之前的通知仍然可见，则系统会根据 `Notification` 对象的内容更新该通知。相反，如果之前的通知已被清除，系统则会创建一个新通知。

以下代码段演示了经过更新以反映所发生事件数量的通知。 它将通知堆叠并显示摘要：

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
...
```

删除通知

除非发生以下情况之一，否则通知仍然可见：

- 用户单独或通过使用“全部清除”清除了该通知（如果通知可以清除）。
- 用户点击通知，且您在创建通知时调用了 `setAutoCancel()`。
- 您针对特定的通知 ID 调用了 `cancel()`。此方法还会删除当前通知。
- 您调用了 `cancelAll()` 方法，该方法将删除之前发出的所有通知。

启动 Activity 时保留导航

从通知中启动 `Activity` 时，您必须保留用户的预期导航体验。 点击“返回”应该使用户将应用的正常工作流回退到主屏幕，而点击“最新动态”则应将 `Activity` 显示为单独的任务。 要保留导航体验，您应该在全新任务中启动 `Activity`。如何设置 `PendingIntent` 以获得全新任务取决于正在启动的 `Activity` 的性质。一般有两种情况：

常规 Activity

> 您要启动的 `Activity` 是应用的正常工作流的一部分。在这种情况下，请设置 `PendingIntent` 以启动全新任务并为 `PendingIntent`提供返回栈，这将重现应用的正常“返回”行为。
>
> Gmail 应用中的通知演示了这一点。点击一封电子邮件消息的通知时，您将看到消息具体内容。 触摸**返回**将使您从 Gmail 回退到主屏幕，就好像您是从主屏幕（而不是通知）进入 Gmail 一样。
>
> 无论您触摸通知时处于哪个应用，都会发生这种情况。 例如，如果您在 Gmail 中撰写消息时点击了一封电子邮件的通知，则会立即转到该电子邮件。 触摸“返回”会依次转到收件箱和主屏幕，而不是转到您在撰写的邮件。

特殊 Activity

> 仅当从通知中启动时，用户才会看到此 `Activity`。 从某种意义上说，`Activity` 是通过提供很难显示在通知本身中的信息来扩展通知。对于这种情况，请将 `PendingIntent` 设置为在全新任务中启动。但是，由于启动的 `Activity` 不是应用 Activity 流程的一部分，因此无需创建返回栈。点击“返回”仍会将用户带到主屏幕。

设置常规 Activity PendingIntent

要设置可启动直接进入 `Activity` 的 `PendingIntent`，请执行以下步骤：

1. 在清单文件中定义应用的 `Activity` 层次结构。
    a. 添加对 Android 4.0.3 及更低版本的支持。为此，请通过添加 `<meta-data>` 元素作为 `<activity>`的子项来指定正在启动的 `Activity` 的父项。

对于此元素，请设置 `android:name="android.support.PARENT_ACTIVITY"`。 设置 `android:value="<parent_activity_name>"`，其中，`<parent_activity_name>` 是父 `<activity>` 元素的 `android:name` 值。请参阅下面的 XML 示例。

b. 同样添加对 Android 4.1 及更高版本的支持。为此，请将 `android:parentActivityName` 属性添加到正在启动的 `Activity` 的 `<activity>` 元素中。

最终的 XML 应如下所示：

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

2. 根据可启动 `Activity` 的 `Intent` 创建返回栈：

a. 创建 `Intent` 以启动 `Activity`。

b. 通过调用 `TaskStackBuilder.create()` 创建堆栈生成器。

c. 通过调用 `addParentStack()` 将返回栈添加到堆栈生成器。 对于在清单文件中所定义层次结构内的每个 `Activity`，返回栈均包含可启动 `Activity` 的 `Intent` 对象。此方法还会添加一些可在全新任务中启动堆栈的标志。

> **注：**尽管 `addParentStack()` 的参数是对已启动 `Activity` 的引用，但是方法调用不会添加可启动 `Activity` 的 `Intent`，而是留待下一步进行处理。

d. 通过调用 `addNextIntent()`，添加可从通知中启动 `Activity` 的 `Intent`。 将在第一步中创建的 `Intent` 作为 `addNextIntent()` 的参数传递。

e. 如需，请通过调用 `TaskStackBuilder.editIntentAt()` 向堆栈中的 `Intent` 对象添加参数。有时，需要确保目标 `Activity` 在用户使用“返回”导航回它时会显示有意义的数据。

f. 通过调用 `getPendingIntent()` 获得此返回栈的 `PendingIntent`。 然后，您可以使用此 `PendingIntent` 作为 `setContentIntent()` 的参数。

以下代码段演示了该流程：

```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

设置特殊 Activity PendingIntent

下文介绍如何设置特殊 Activity `PendingIntent`。

特殊 `Activity` 无需返回栈，因此您不必在清单文件中定义其 `Activity` 层次结构，也不必调用 `addParentStack()` 来构建返回栈。取而代之的是，您可使用清单文件设置 `Activity` 任务选项，并通过调用 `getActivity()` 创建 `PendingIntent`：

1. 在清单文件中，将以下属性添加到 `Activity` 的 `<activity>` 元素

   `android:name="activityclass"`

   Activity 的完全限定类名。

   `android:taskAffinity=""`

   与您在代码中设置的 `FLAG_ACTIVITY_NEW_TASK` 标志相结合，这可确保此 `Activity` 不会进入应用的默认任务。任何具有应用默认关联的现有任务均不受影响。

   `android:excludeFromRecents="true"`

   将新任务从“最新动态”中排除，这样用户就不会在无意中导航回它。

   以下代码段显示了该元素：

   ```
   <activity
       android:name=".ResultActivity"
   ...
       android:launchMode="singleTask"
       android:taskAffinity=""
       android:excludeFromRecents="true">
   </activity>
   ...
   ```

2. 构建并发出通知：

   a. 创建可启动 `Activity`的 `Intent`。

   b. 通过使用 `FLAG_ACTIVITY_NEW_TASK` 和 `FLAG_ACTIVITY_CLEAR_TASK` 标志调用 `setFlags()`，将 `Activity` 设置为在新的空任务中启动。

   c. 为 `Intent` 设置所需的任何其他选项。

   d. 通过调用 `getActivity()` 从 `Intent` 中创建 `PendingIntent`。 然后，您可以使用此 `PendingIntent` 作为 `setContentIntent()` 的参数。

   以下代码段演示了该流程：

```
// Instantiate a Builder object.
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
// Creates an Intent for the Activity
Intent notifyIntent =
        new Intent(this, ResultActivity.class);
// Sets the Activity to start in a new, empty task
notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                        | Intent.FLAG_ACTIVITY_CLEAR_TASK);
// Creates the PendingIntent
PendingIntent notifyPendingIntent =
        PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyPendingIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```

在通知中显示进度

通知可能包括动画形式的进度指示器，向用户显示正在进行的操作状态。 如果您可以估计操作所需的时间以及任意时刻的完成进度，则使用“限定”形式的指示器（进度栏）。 如果无法估计操作的时长，则使用“非限定”形式的指示器（Activity 指示器）。

平台的 `ProgressBar` 类实现中显示有进度指示器。

要在 Android 4.0 及更高版本的平台上使用进度指示器，需调用 `setProgress()`。对于早期版本，您必须创建包括 `ProgressBar` 视图的自定义通知布局。

下文介绍如何使用 `setProgress()` 在通知中显示进度。

显示持续时间固定的进度指示器

要显示限定形式的进度栏，请通过调用 `setProgress(max, progress, false)` 将进度栏添加到通知，然后发出通知。随着操作继续进行，递增 `progress` 并更新通知。操作结束时， `progress` 应该等于 `max`。调用 `setProgress()` 的常见方法是将 `max` 设置为 100，然后将 `progress` 作为操作的“完成百分比”值递增。

您可以在操作完成后仍保留显示进度栏，也可以将其删除。无论哪种情况，都请记住更新通知文本以显示操作已完成。 要删除进度栏，请调用 `setProgress(0, 0, false)`。例如：

```
...
mNotifyManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                    // Sets the progress indicator to a max value, the
                    // current completion percentage, and "determinate"
                    // state
                    mBuilder.setProgress(100, incr, false);
                    // Displays the progress bar for the first time.
                    mNotifyManager.notify(0, mBuilder.build());
                        // Sleeps the thread, simulating an operation
                        // that takes time
                        try {
                            // Sleep for 5 seconds
                            Thread.sleep(5*1000);
                        } catch (InterruptedException e) {
                            Log.d(TAG, "sleep failure");
                        }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
                    .setProgress(0,0,false);
            mNotifyManager.notify(ID, mBuilder.build());
        }
    }
// Starts the thread by calling the run() method in its Runnable
).start();
```

显示持续 Activity 指示器

要显示非限定形式的 Activity 指示器，请使用 `setProgress(0, 0, true)` 将其添加到通知（忽略前两个参数），然后发出通知。这样一来，指示器的样式就与进度栏相同，只是其动画还在继续。

在操作开始之际发出通知。除非您修改通知，否则动画将一直运行。 操作完成后，调用 `setProgress(0, 0, false)`，然后更新通知以删除 Activity 指示器。 请务必这样做；否则，即使操作完成，动画仍将运行。同时，请记得更改通知文本，以表明操作已完成。

要了解 Activity 指示器的工作方式，请参阅上述代码段。找到以下几行：

```
// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```

将找到的这几行替换为以下几行：

```
 // Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```

通知元数据

通知可根据您使用以下 `NotificationCompat.Builder` 方法分配的元数据进行排序：

- 当设备处于“优先”模式时，`setCategory()` 会告知系统如何处理应用通知（例如，通知代表传入呼叫、即时消息还是闹铃）。
- 如果优先级字段设置为 `PRIORITY_MAX` 或 `PRIORITY_HIGH` 的通知还有声音或振动，则 `setPriority()` 会将其显示在小型浮动窗口中。
- `addPerson()` 允许您向通知添加人员名单。您的应用可以使用此名单指示系统将指定人员发出的通知归成一组，或者将这些人员发出的通知视为更重要的通知。

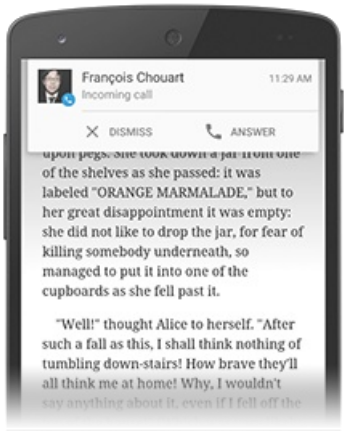

图 3. 显示浮动通知的全屏 Activity

浮动通知

对于 Android 5.0（API 级别 21），当设备处于活动状态时（即，设备未锁定且其屏幕已打开），通知可以显示在小型浮动窗口中（也称为“浮动通知”）。 这些通知看上去类似于精简版的通知，只是浮动通知还显示操作按钮。 用户可以在不离开当前应用的情况下处理或清除浮动通知。

可能触发浮动通知的条件示例包括：

- 用户的 Activity 处于全屏模式中（应用使用 `fullScreenIntent`），或者
- 通知具有较高的优先级并使用铃声或振动

锁定屏幕通知

随着 Android 5.0（API 级别 21）的发布，通知现在还可显示在锁定屏幕上。您的应用可以使用此功能提供媒体播放控件以及其他常用操作。用户可以通过“设置”选择是否将通知显示在锁定屏幕上，并且您可以指定您应用中的通知在锁定屏幕上是否可见。

设置可见性

您的应用可以控制在安全锁定屏幕上显示的通知中可见的详细级别。 调用 `setVisibility()` 并指定以下值之一：

- `VISIBILITY_PUBLIC` 显示通知的完整内容。
- `VISIBILITY_SECRET` 不会在锁定屏幕上显示此通知的任何部分。
- `VISIBILITY_PRIVATE` 显示通知图标和内容标题等基本信息，但是隐藏通知的完整内容。

设置 `VISIBILITY_PRIVATE` 后，您还可以提供其中隐藏了某些详细信息的替换版本通知内容。例如，短信 应用可能会显示一条通知，指出“您有 3 条新短信”，但是隐藏了短信内容和发件人。要提供此替换版本的通知，请先使用 `NotificationCompat.Builder` 创建替换通知。创建专用通知对象时，请通过 `setPublicVersion()` 方法为其附加替换通知。

在锁定屏幕上控制媒体播放

在 Android 5.0（API 级别 21）中，锁定屏幕不再基于 `RemoteControlClient`（现已弃用）显示媒体控件。取而代之的是，将 `Notification.MediaStyle` 模板与 `addAction()` 方法结合使用，后者可将操作转换为可点击的图标。

> **注：**该模板和 `addAction()` 方法未包含在支持库中，因此这些功能只能在 Android 5.0 及更高版本的系统上运行。

要在 Android 5.0 系统的锁定屏幕上显示媒体播放控件，请将可见性设置为 `VISIBILITY_PUBLIC`，如上文所述。然后，添加操作并设置 `Notification.MediaStyle` 模板，如以下示例代码中所述：

```
Notification notification = new Notification.Builder(context)
    // Show controls on lock screen even when user hides sensitive content.
    .setVisibility(Notification.VISIBILITY_PUBLIC)
    .setSmallIcon(R.drawable.ic_stat_player)
    // Add media control buttons that invoke intents in your media service
    .addAction(R.drawable.ic_prev, "Previous", prevPendingIntent) // #0
    .addAction(R.drawable.ic_pause, "Pause", pausePendingIntent)  // #1
    .addAction(R.drawable.ic_next, "Next", nextPendingIntent)     // #2
    // Apply the media style template
    .setStyle(new Notification.MediaStyle()
    .setShowActionsInCompactView(1 /* #1: pause button */)
    .setMediaSession(mMediaSession.getSessionToken())
    .setContentTitle("Wonderful music")
    .setContentText("My Awesome Band")
    .setLargeIcon(albumArtBitmap)
    .build();
```

> **注：**弃用 `RemoteControlClient` 会对控制媒体产生进一步的影响。如需了解有关用于管理媒体会话和控制播放的新 API 的详细信息，请参阅媒体播放控件。

自定义通知布局

您可以利用通知框架定义自定义通知布局，由该布局定义通知在 `RemoteViews` 对象中的外观。 自定义布局通知类似于常规通知，但是它们是基于 XML 布局文件中所定义的 `RemoteViews`。

自定义通知布局的可用高度取决于通知视图。普通视图布局限制为 64 dp，扩展视图布局限制为 256 dp。

要定义自定义通知布局，请首先实例化 `RemoteViews` 对象来扩充 XML 布局文件。然后，调用 `setContent()`，而不是调用 `setContentTitle()` 等方法。要在自定义通知中设置内容详细信息，请使用 `RemoteViews` 中的方法设置视图子项的值：

1. 在单独的文件中为通知创建 XML 布局。您可以根据需要使用任何文件名，但必须使用扩展名 `.xml`。
2. 在您的应用中，使用 `RemoteViews` 方法定义通知的图标和文本。通过调用 `setContent()` 将此 `RemoteViews` 对象放入 `NotificationCompat.Builder` 中。避免在 `RemoteViews` 对象上设置背景 `Drawable`，因为文本颜色可能使文本变得难以阅读。

此外，`RemoteViews` 类中还有一些方法可供您轻松将 `Chronometer` 或 `ProgressBar` 添加到通知布局。如需了解有关为通知创建自定义布局的详细信息，请参阅 `RemoteViews` 参考文档。

> **注意：**使用自定义通知布局时，要特别注意确保自定义布局适用于不同的设备方向和分辨率。 尽管这条建议适用于所有“视图”布局，但对通知尤为重要，因为抽屉式通知栏中的空间非常有限。 不要让自定义布局过于复杂，同时确保在各种配置中对其进行测试。

对自定义通知文本使用样式资源

始终对自定义通知的文本使用样式资源。通知的背景颜色可能因设备和系统版本的不同而异，使用样式资源有助于您充分考虑到这一点。 从 Android 2.3 开始，系统定义了标准通知布局文本的样式。若要在面向 Android 2.3 或更高版本系统的多个应用中使用相同样式，则应确保文本在显示背景上可见。