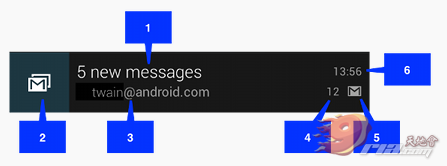
**Android 应用之 Notification**

**前言**

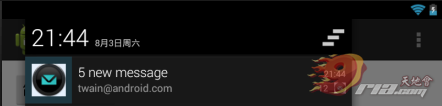
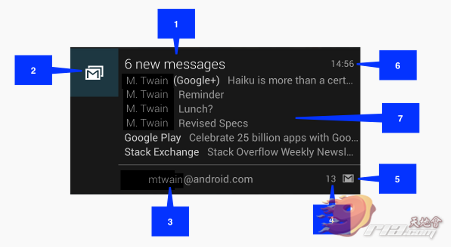
在本文中，将介绍Notification的常规使用，以及自定义方式的使用，对于每种不同的方式，都提供示例展示效果。  
**Notification**  
　　Notification，俗称通知，是一种具有全局效果的通知，它展示在屏幕的顶端，首先会表现为一个图标的形式，当用户向下滑动的时候，展示出通知具体的内容。  
　　注意：因为一些Android版本的兼容性问题，对于Notification而言，Android3.0是一个分水岭，在其之前构建Notification推荐使用Notification.Builder构建，而在Android3.0之后，一般推荐使用NotificationCompat.Builder构建。本文的所有代码环境均在4.3中完成，如果使用4.1一下的设备测试，请注意兼容性问题。  
　　通知一般通过NotificationManager服务来发送一个Notification对象来完成，NotificationManager是一个重要的系统级服务，该对象位于应用程序的框架层中，应用程序可以通过它像系统发送全局的通知。这个时候需要创建一个Notification对象，用于承载通知的内容。但是一般在实际使用过程中，一般不会直接构建Notification对象，而是使用它的一个内部类NotificationCompat.Builder来实例化一个对象（Android3.0之下使用Notification.Builder），并设置通知的各种属性，最后通过NotificationCompat.Builder.build()方法得到一个Notification对象。当获得这个对象之后，可以使用NotificationManager.notify()方法发送通知。  
　　NotificationManager类是一个通知管理器类，这个对象是由系统维护的服务，是以单例模式获得，所以一般并不直接实例化这个对象。在Activity中，可以使用Activity.getSystemService(String)方法获取NotificationManager对象，Activity.getSystemService(String)方法可以通过Android系统级服务的句柄，返回对应的对象。在这里需要返回NotificationManager，所以直接传递Context.NOTIFICATION\_SERVICE即可。  
　　虽然通知中提供了各种属性的设置，但是一个通知对象，有几个属性是必须要设置的，其他的属性均是可选的，必须设置的属性如下：  
•小图标，使用setSamllIcon()方法设置。  
•标题，使用setContentTitle()方法设置。  
•文本内容，使用setContentText()方法设置。   
**更新与移除通知**  
　　在使用NotificationManager.notify()发送通知的时候，需要传递一个标识符，用于唯一标识这个通知。对于有些场景，并不是无限的添加新的通知，有时候需要更新原有通知的信息，这个时候可以重写构建Notification，而使用与之前通知相同标识符来发送通知，这个时候旧的通知就被被新的通知所取代，起到更新通知的效果。  
　　对于一个通知，当展示在状态栏之后，但是使用过后，如何取消呢？Android为我们提供两种方式移除通知，一种是Notification自己维护，使用setAutoCancel()方法设置是否维护，传递一个boolean类型的数据。另外一种方式使用NotificationManager通知管理器对象来维护，它通过notify()发送通知的时候，指定的通知标识Id来操作通知，可以使用cancel(int)来移除一个指定的通知，也可以使用cancelAll()移除所有的通知。

**PendingIntent**  
　　对于一个通知而言，它显示的消息是有限的，一般仅用于提示一些概要信息。但是一般简短的消息，并不能表达需要告诉用户的全部内容，所以需要绑定一个意图，当用户点击通知的时候，调用一个意图展示出一个Activity用来显示详细的内容。而Notification中，并不使用常规的Intent去传递一个意图，而是使用PendingIntent。  
　　先来聊聊Intent和PendingIntent的区别，PendingIntent可以看做是对Intent的包装，通过名称可以看出PendingIntent用于处理即将发生的意图，而Intent用来处理马上发生的意图。而对于通知来说，它是一个系统级的全局的通知，并不确定这个意图被执行的时间。当在应用外部执行PendingIntent时，因为它保存了触发App的Context，使得外部App可以如果当前App一样执行PendingIntent里的Intent，就算执行时触发通知的App已经不存在了，也能通过存在PendingIntent里的Context照常执行Intent，并且还可以处理Intent所带来的额外的信息。  
　　PendingIntent提供了多个静态的getXxx()方法，用于获得适用于不同场景的PendingIntent对象。一般需要传递的几个参数都很常规，只介绍一个flag参数，用于标识PendingIntent的构造选择：  
•FLAG\_CANCEL\_CURRENT：如果构建的PendingIntent已经存在，则取消前一个，重新构建一个。  
•FLAG\_NO\_CREATE：如果前一个PendingIntent已经不存在了，将不再构建它。  
•FLAG\_ONE\_SHOT：表明这里构建的PendingIntent只能使用一次。  
•FLAG\_UPDATE\_CURRENT：如果构建的PendingIntent已经存在，则替换它，常用。  
**Notification视觉风格**  
　　Notification有两种视觉风格，一种是标准视图(Normal view)、一种是大视图（Big view）。标准视图在Android中各版本是通用的，但是对于大视图而言，仅支持Android4.1+的版本。  
　　从官方文档了解到，一个标准视图显示的大小要保持在64dp高，宽度为屏幕标准。标准视图的通知主体内容有一下几个：  


1. 通知标题。
2. 大图标。
3. 通知内容。
4. 通知消息。
5. 小图标。
6. 通知的时间，一般为系统时间，也可以使用setWhen()设置。

　　下面通过一个示例，模仿上面效果的通知。

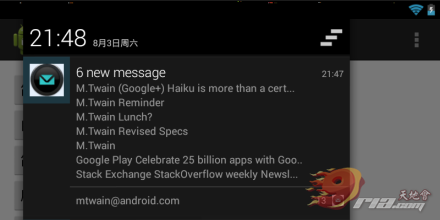
1. btnNotification.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View v) {
4. Bitmap btm = BitmapFactory.decodeResource(getResources(),
5. R.drawable.msg);
6. NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(
7. MainActivity.this).setSmallIcon(R.drawable.msg)
8. .setContentTitle("5 new message")
9. .setContentText("twain@android.com");
10. mBuilder.setTicker("New message");//第一次提示消息的时候显示在通知栏上
11. mBuilder.setNumber(12);
12. mBuilder.setLargeIcon(btm);
13. mBuilder.setAutoCancel(true);//自己维护通知的消失
14. //构建一个Intent
15. Intent resultIntent = new Intent(MainActivity.this,
16. ResultActivity.class);
17. //封装一个Intent
18. PendingIntent resultPendingIntent = PendingIntent.getActivity(
19. MainActivity.this, 0, resultIntent,
20. PendingIntent.FLAG\_UPDATE\_CURRENT);
21. // 设置通知主题的意图
22. mBuilder.setContentIntent(resultPendingIntent);
23. //获取通知管理器对象
24. NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);
25. mNotificationManager.notify(0, mBuilder.build());
26. }
27. });

　　显示效果：  
   
　　而对于大视图（Big View）而言，它的细节区域只能显示256dp高度的内容，并且只对Android4.1+之后的设备才支持，它比标准视图不一样的地方，均需要使用setStyle()方法设定，它大致的效果如下：  
   
　　setStyle()传递一个NotificationCompat.Style对象，它是一个抽象类，Android为我们提供了三个实现类，用于显示不同的场景。分别是：

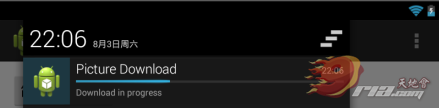
* NotificationCompat.BigPictureStyle, 在细节部分显示一个256dp高度的位图。
* NotificationCompat.BigTextStyle，在细节部分显示一个大的文本块。
* NotificationCompat.InboxStyle，在细节部分显示一段行文本。

　　如果仅仅显示一个图片，使用BigPictureStyle是最方便的；如果需要显示一个富文本信息，则可以使用BigTextStyle；如果仅仅用于显示一个文本的信息，那么使用InboxStyle即可。后面会以一个示例来展示InboxStyle的使用，模仿上面图片的显示。  
　　实现代码：

1. btnBigViewNotification.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View v) {
4. Bitmap btm = BitmapFactory.decodeResource(getResources(),
5. R.drawable.msg);
6. Intent intent = new Intent(MainActivity.this,
7. ResultActivity.class);
8. PendingIntent pendingIntent = PendingIntent.getActivity(
9. MainActivity.this, 0, intent,
10. PendingIntent.FLAG\_CANCEL\_CURRENT);
11. Notification noti = new NotificationCompat.Builder(
12. MainActivity.this)
13. .setSmallIcon(R.drawable.msg)
14. .setLargeIcon(btm)
15. .setNumber(13)
16. .setContentIntent(pendingIntent)
17. .setStyle(
18. new NotificationCompat.InboxStyle()
19. .addLine(
20. "M.Twain (Google+) Haiku is more than a cert...")
21. .addLine("M.Twain Reminder")
22. .addLine("M.Twain Lunch?")
23. .addLine("M.Twain Revised Specs")
24. .addLine("M.Twain ")
25. .addLine(
26. "Google Play Celebrate 25 billion apps with Goo..")
27. .addLine(
28. "Stack Exchange StackOverflow weekly Newsl...")
29. .setBigContentTitle("6 new message")
30. .setSummaryText("mtwain@android.com"))
31. .build();
32. NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);
33. mNotificationManager.notify(0, noti);
34. }
35. });

　　展示效果：  
   
**进度条样式的通知**  
　　对于一个标准通知，有时候显示的消息并不一定是静态的，还可以设定一个进度条用于显示事务完成的进度。  
　　Notification.Builder类中提供一个setProgress(int max,int progress,boolean indeterminate)方法用于设置进度条，max用于设定进度的最大数，progress用于设定当前的进度，indeterminate用于是否是一个确定进度只的进度条。通过indeterminate的设置，可以实现两种不同样式的进度条，一种是有进度的（false）,一种是循环流动的（false）。下面分别用两个示例演示：  
　　有进度的进度条，实现代码：

1. btnProgreNotification.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View v) {
4. manager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);
5. builder = new NotificationCompat.Builder(MainActivity.this)
6. .setSmallIcon(R.drawable.ic\_launcher)
7. .setContentTitle("Picture Download")
8. .setContentText("Download in progress");
9. builder.setAutoCancel(true);
10. //通过一个子线程，动态增加进度条刻度
11. new Thread(new Runnable() {
12. @Override
13. public void run() {
14. int incr;
15. for (incr = 0; incr <= 100; incr += 5) {
16. builder.setProgress(100, incr, false);
17. manager.notify(0, builder.build());
18. try {
19. Thread.sleep(300);
20. } catch (InterruptedException e) {
21. Log.i(TAG, "sleep failure");
22. }
23. }
24. builder.setContentText("Download complete")
25. .setProgress(0, 0, false);
26. manager.notify(0, builder.build());
27. }
28. }).start();
29. }
30. });

　　显示效果：  
   
　　对于循环流动的进度条，下面是实现代码：

1. btnProNotification.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View v) {
4. manager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);
5. builder = new NotificationCompat.Builder(MainActivity.this)
6. .setSmallIcon(R.drawable.ic\_launcher)
7. .setContentTitle("Picture Download")
8. .setContentText("Download in progress");
9. builder.setProgress(0, 0, true);//设置为true，表示流动
10. manager.notify(0, builder.build());
11. //5秒之后还停止流动
12. new Thread(new Runnable() {
13. @Override
14. public void run() {
15. try {
16. Thread.sleep(5000);
17. } catch (InterruptedException e) {
18. e.printStackTrace();
19. }
20. builder.setProgress(100, 100, false);//设置为true，表示刻度
21. manager.notify(0, builder.build());
22. }
23. }).start();
24. }
25. });

　　效果展示：  
   
**自定义通知**  
　　和Toast一样，通知也可以使用自定义的XML来自定义样式，但是对于通知而言，因为它的全局性，并不能简单的通过inflate膨胀出一个View，因为可能触发通知的时候，响应的App已经关闭，无法获取当指定的XML布局文件。所以需要使用单独的一个RemoteViews类来操作。  
　　RemoteViews，描述了一个视图层次的结构，可以显示在另一个进程。层次结构也是从布局文件中“膨胀”出一个视图，这个类，提供了一些基本的操作求改其膨胀的内容。  
　　RemoteViews提供了多个构造函数，一般使用RemoteViews(String packageName,int layoutId)。第一个参数为包的名称，第二个为layout资源的Id。当获取到RemoteViews对象之后，可以使用它的一系列setXxx()方法通过控件的Id设置控件的属性。最后使用NotificationCompat.Builder.setContent(RemoteViews)方法设置它到一个Notification中。  
　　下面通过一个示例展示它：  
　　自定义的布局XML代码：

1. <RELATIVELAYOUT
2. android:layout\_width="match\_parent"
3. android:layout\_height="match\_parent"
4. android:padding="10dp" >
5. <IMAGEVIEW
6. android:id="@+id/imageNo"
7. android:layout\_width="wrap\_content"
8. android:layout\_height="match\_parent"
9. android:layout\_alignParentLeft="true"
10. android:layout\_marginRight="10dp" />
11. <TEXTVIEW
12. android:id="@+id/titleNo"
13. android:layout\_width="wrap\_content"
14. android:layout\_height="wrap\_content"
15. android:layout\_toRightOf="@id/imageNo" />
16. <TEXTVIEW
17. android:id="@+id/textNo"
18. android:layout\_width="wrap\_content"
19. android:layout\_height="wrap\_content"
20. android:layout\_below="@id/titleNo"
21. android:layout\_toRightOf="@id/imageNo" />

实现代码：

1. btnCustomNotification.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View v) {
4. RemoteViews contentViews = new RemoteViews(getPackageName(),
5. R.layout.custom\_notification);
6. //通过控件的Id设置属性
7. contentViews
8. .setImageViewResource(R.id.imageNo, R.drawable.btm1);
9. contentViews.setTextViewText(R.id.titleNo, "自定义通知标题");
10. contentViews.setTextViewText(R.id.textNo, "自定义通知内容");
11. Intent intent = new Intent(MainActivity.this,
12. ResultActivity.class);
13. PendingIntent pendingIntent = PendingIntent.getActivity(
14. MainActivity.this, 0, intent,
15. PendingIntent.FLAG\_CANCEL\_CURRENT);
16. NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(
17. MainActivity.this).setSmallIcon(R.drawable.ic\_launcher)
18. .setContentTitle("My notification")
19. .setTicker("new message");
20. mBuilder.setAutoCancel(true);
21. mBuilder.setContentIntent(pendingIntent);
22. mBuilder.setContent(contentViews);
23. mBuilder.setAutoCancel(true);
24. NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);
25. mNotificationManager.notify(10, mBuilder.build());
26. }
27. });

　　效果展示：  
   
**设定提示响应**  
　　对于有些通知，需要调用一些设备的资源，使用户能更快的发现有新通知，一般可设定的响应有：铃声、闪光灯、震动。对于这三个属性，NotificationCompat.Builder提供了三个方法设定：

* setSound(Uri sound)：设定一个铃声，用于在通知的时候响应。传递一个Uri的参数，格式为“file:///mnt/sdcard/Xxx.mp3”。
* setLights(

int argb, int onMs, int offMs

)：设定前置LED灯的闪烁速率，持续毫秒数，停顿毫秒数。

* setVibrate(long[] pattern)：设定震动的模式，以一个long数组保存毫秒级间隔的震动。

　　大多数时候，我们并不需要设定一个特定的响应效果，只需要遵照用户设备上系统通知的效果即可，那么可以使用setDefaults(int)方法设定默认响应参数，在Notification中，对它的参数使用常量定义了，我们只需使用即可：

* DEFAULT\_ALL：铃声、闪光、震动均系统默认。
* DEFAULT\_SOUND：系统默认铃声。
* DEFAULT\_VIBRATE：系统默认震动。
* DEFAULT\_LIGHTS：系统默认闪光。

　　而在Android中，如果需要访问硬件设备的话，是需要对其进行授权的，所以需要在清单文件AndroidManifest.xml中增加两个授权，分别授予访问振动器与闪光灯的权限：

1. <!-- 闪光灯权限 -->
2. <uses-permission android:name="android.permission.FLASHLIGHT"/>
3. <!-- 振动器权限 -->
4. <uses-permission android:name="android.permission.VIBRATE"/>

　　因为只是一个属性的设定，并且大部分时候，使用系统设定即可，这里就不提供代码示例了。   
**总结**  
　　通知算是Android中比较常用的一个功能，可以保持自己App的长存，在用户没有进入App的时候，也提供了与用户交互的可能。  
原文链接：<http://www.cnblogs.com/plokmju/p/android_Notification.html>