OCT 2ND, 2015 | COMMENTS

Android开发最佳实践

前段时间，Google公布了Android开发最佳实践的一系列课程，涉及到一些平时开发过程中应该保持的良好习惯以及如何使用最新的Android Design Support Library来快速实现官方推荐的Material Design样式的应用。下面是个人的学习摘要总结，不对的地方请多多交流指点，谢谢！

1）注意对隐式Intent的运行时检查保护

类似打开相机，发送图片等隐式Intent，是并不一定能够在所有的Android设备上都正常运行。例如打开相机的隐式Intent，如果系统相机应用被关闭或者不存在相机应用，又或者是相机应用的某些权限被关闭等等情况都可能导致这个隐式的Intent无法正常工作。一旦发生隐式Intent找不到合适的调用组件的情况，系统就会抛出`ActivityNotFoundException`的异常，如果我们的应用没有对这个异常做任何处理，那应用就会发生Crash。

预防这个问题的最佳解决方案是在发出这个隐式Intent之前调用`resolveActivity`做检查，关于这个API的解释以及用法如下：

public ComponentName **resolveActivity** (PackageManager pm) Added in API level 1

Return the Activity component that should be used to handle this intent. The appropriate component is determined based on the information in the intent, evaluated as follows:

If `getComponent()` returns an explicit class, that is returned without any further consideration.

The activity must handle the `CATEGORY_DEFAULT` Intent category to be considered.

If `getAction()` is non-NULL, the activity must handle this action.

If `resolveType(ContentResolver)` returns non-NULL, the activity must handle this type.

If `addCategory(String)` has added any categories, the activity must handle ALL of the categories specified.

If `getPackage()` is non-NULL, only activity components in that application package will be considered.

If there are no activities that satisfy all of these conditions, a null string is returned.

If multiple activities are found to satisfy the intent, the one with the highest priority will be used. If there are multiple activities with the same priority, the system will either pick the best activity based on user preference, or resolve to a system class that will allow the user to pick an activity and forward from there.

This method is implemented simply by calling `resolveActivity(Intent, int)` with the "defaultOnly" parameter true.

This API is called for you as part of starting an activity from an intent. You do not normally need to call it yourself.

然后这个API的使用范例如下：

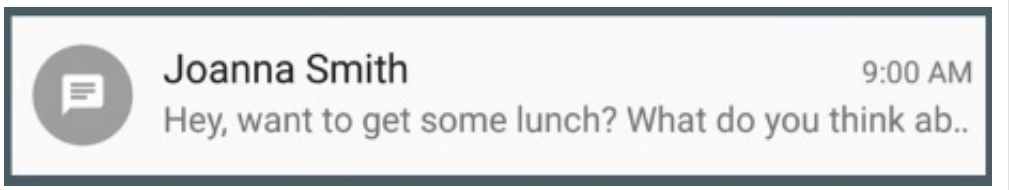
```
1  Intent intent = new Intent(Intent.ACTION_XXX);
2  ComponentName componentName = intent.resolveActivity(getPackageManager());
3  if(componentName != null) {
4      String className = componentName.getClassName();
5  }
```

2）使用NotificationCompat兼容包来处理消息通知

为了解决Android系统版本差异导致的Notification兼容性问题，Android官方提供了`NotificationCompat`兼容类来帮助开发实现体验统一的Notification。通常来说，建立一个Notification至少会有三种元素：图标，标题，文本。我们通常会使用如下的代码来实现一个基础的Notification功能：

```
NotificationCompat.Builder builder =
  new NotificationCompat.Builder(this);
builder
  .setSmallIcon(R.drawable.notification_icon)
  .setContentTitle("Joanna Smith");
String text = "Hey, want to get some lunch? What do " +
  "you think about burritos?";
builder.setContentText(text);



Notification notification = builder.build();
NotificationManagerCompat.from(this).notify(0, notification);
```

上面那段代码，运行时候的效果应该如下所示：



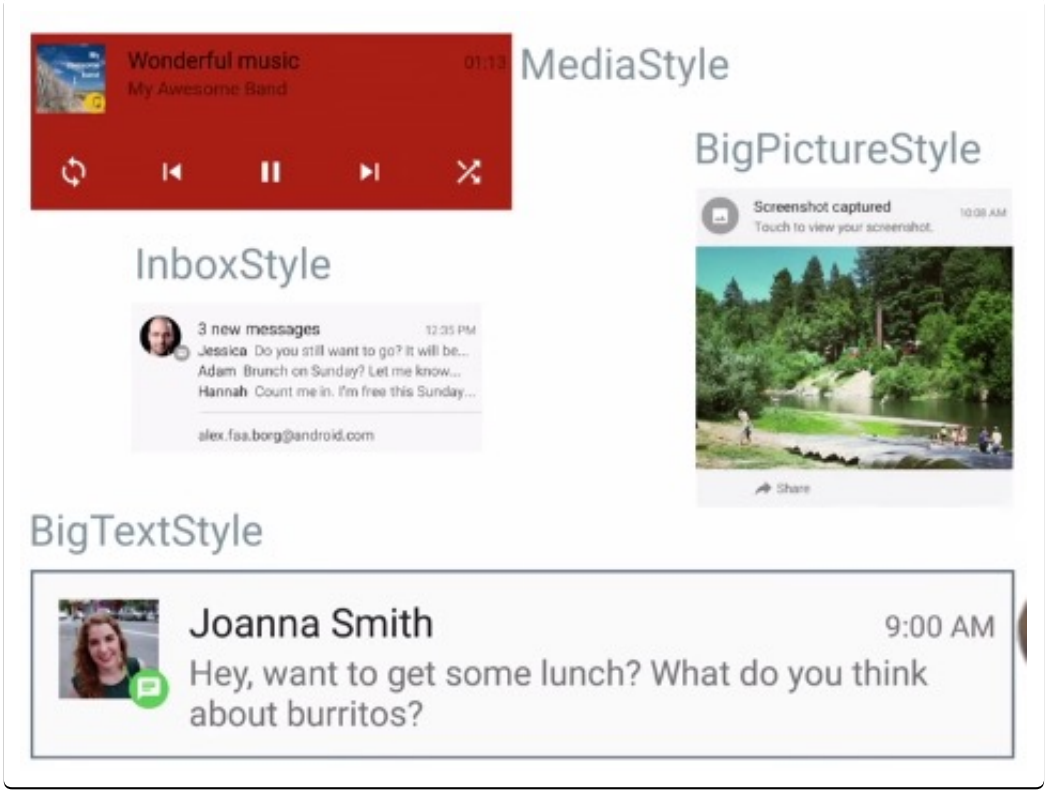
为了给上面的Notification添加点击之后的响应效果，我们还需要构造一个`PendingIntent`作为contentIntent，例如：

```
PendingIntent intent = xxx;
builder.setContentIntent(intent);
```

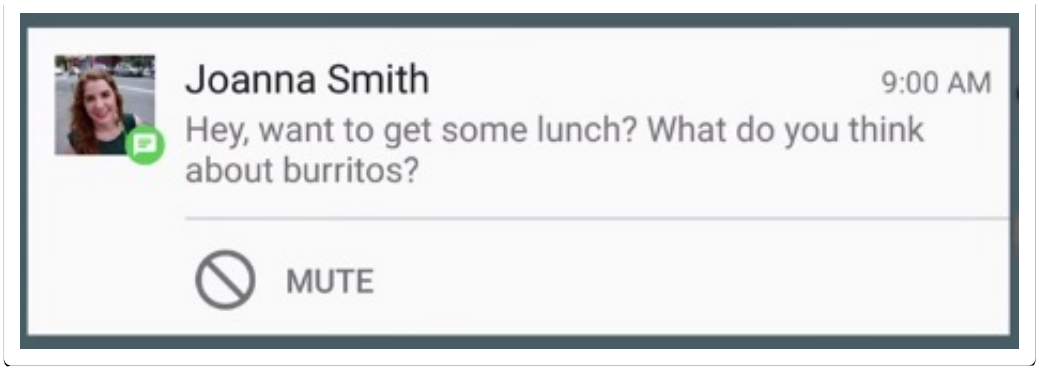
为了使得Notification更加的具有辨识度，我们还有可能做如下的设置：

```
// Add your branding color on 5.0+ devices
int color = getResources().getColor(R.color.branding_ftw);
builder.setColor(color);

// Make things personal with a large icon
Bitmap profilePicture = …
builder.setLargeIcon(profilePicture);
```

从Android 4.1开始，Notification可以支持展开显示的模式，这样一来，Notification就演变出了下面4种不同的风格样式：



Notification还提供了快捷操作的功能，如下图所示：




```
// Set our BigTextStyle
builder.setStyle(new NotificationCompat.BigTextStyle(text));

// Add an action
PendingIntent muteIntent = …;
builder.addAction(new NotificationCompat.Action(
    R.drawable.mute_icon, "Mute", muteIntent));
```

除了显示在手机上的Notification，我们还可以给Notification分别设置在Wearable，Auto上的不同表现行为，例如针对可穿戴设备上显示Notification，我们可以如下的设置：

```
NotificationCompat.WearableExtender wearableExtender
  = new NotificationCompat.WearableExtender();

// Do the magic
RemoteInput remoteInput =
  new RemoteInput.Builder("voice_result_key")
      .setLabel("Reply").build();
wearableExtender.addAction(
  new NotificationCompat.Action.Builder(
    R.drawable.mute_icon, "Reply", replyIntent)
      .addRemoteInput(remoteInput)
      .build());
builder.extend(wearableExtender);
```

关于更多的Wearable上的Notification相关的知识，还可以参考Pages of Content与Stackable Notifications

3）Android 6.0 Marshmallow的运行时权限

Android 6.0开始引入了新的运行时权限检查授权机制，替代了之前安装应用的时候对权限进行授权的方案。为了避免6.0及以上的机器运行发生运行时异常，我们需要做到至少以下5个步骤：

- **检查系统版本号**：针对6.0以下的系统版本，默认权限在安装的时候已经获取到了，对于6.0开始的版本，才需要做运行时的权限检查。
- **检查申请的权限**：在使用某个权限之前，需要检查权限是否已经获取到了。
- **解释申请的权限**：在权限没有获取到的情况下，需要通过`shouldShowRequestPermissionRationable()`的判断来决定如何给用户进行提示。
- **执行申请权限操作**：前面判断没有获取到权限，为了能够让功能顺利执行，我们会需要在代码里面再次执行申请此权限的操作。

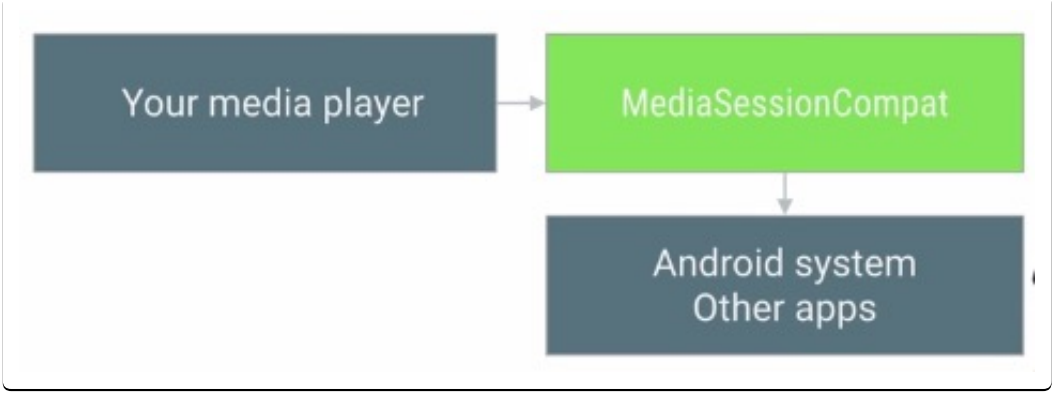
```
public void showCamera(View view) {
    // Check if the Camera permission is already available.
    if (checkSelfPermission(Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {
        // Camera permissions is already available, show the camera preview.

        showCameraPreview();
    } else {
        // Camera permission has not been granted.

        // Provide an additional rationale to the user if the permission was not granted
        // and the user would benefit from additional context for the use of the permission.
        if (shouldShowRequestPermissionRationale(Manifest.permission.CAMERA)) {
            Toast.makeText(this, "Camera permission is needed to show the camera preview.",
                    Toast.LENGTH_SHORT).show();
        }

        // Request Camera permission
        requestPermissions(new String[]{Manifest.permission.CAMERA}, REQUEST_CAMERA);
    }
}
```

- **处理权限申请的结果**：申请权限之后，我们需要处理申请的响应结果，分别处理权限申请成功与失败的情况

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {

    if (requestCode == REQUEST_CAMERA) {
        // Received permission result for camera permission.

        // Check if the only required permission has been granted
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Camera permission has been granted, preview can be displayed
            showCameraPreview();
        } else {
            // Camera permission was denied, so we cannot use this feature.
            Toast.makeText(this, "Permission was not granted", Toast.LENGTH_SHORT).show();

        }
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

4）使用MediaSessionCompat操作音乐的播放

MediaSessionCompat来自Android官方的兼容包，通过它可以告诉Android系统与其他的应用，自己正在播放的内容是什么以及自己支持哪些类型的播放控制：



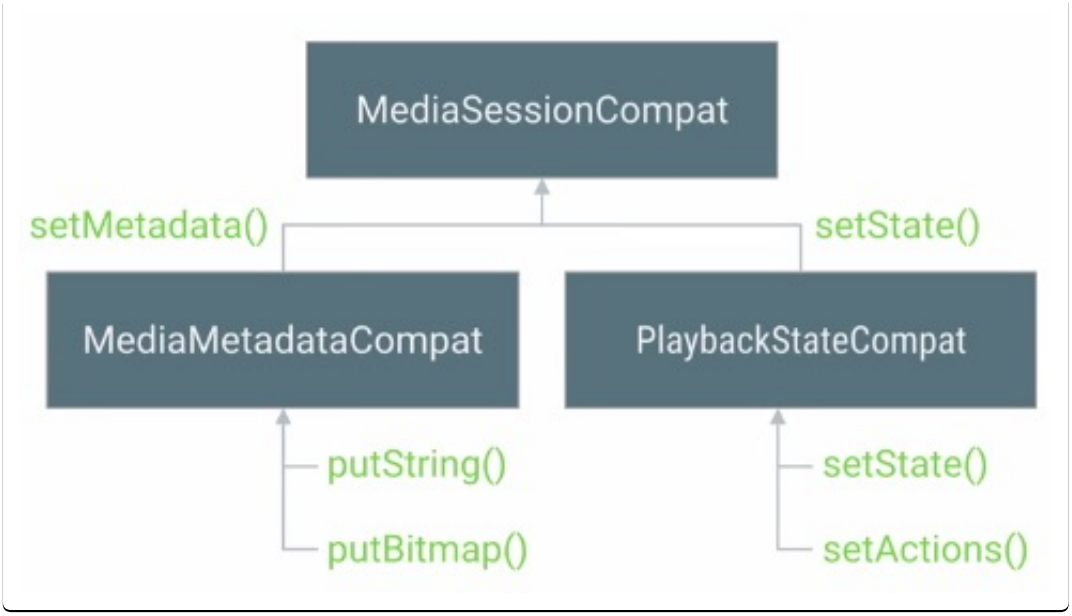
在Android的官方培训课程中有介绍过关于Media Button Receiver的概念，Android系统会把来自蓝牙控制器或者是耳机等其他设备的操作事件转换成Media Button事件传递出来，如果我们的应用程序需要监听这些事件并做出相应的响应，就需要注册MEDIA_BUTTON的action，接收到这些事件之后，再传递给音乐播放模块进行控制处理。

```
<receiver android:name=".RemoteControlReceiver">
    <intent-filter>
        <action android:name="android.intent.action.MEDIA_BUTTON" />
    </intent-filter>
</receiver>
```

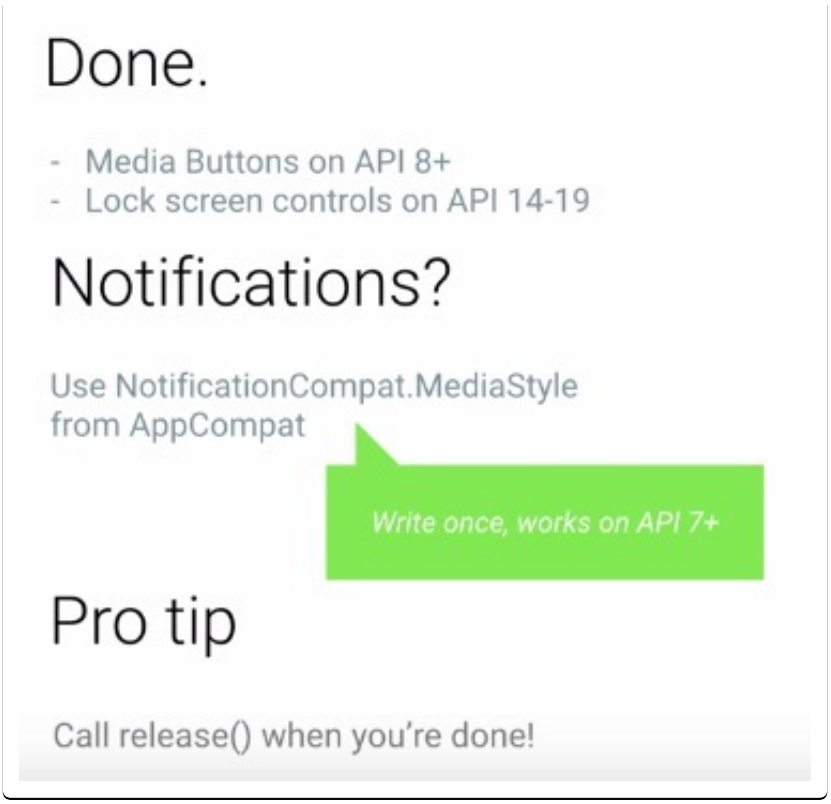
基于上面的认知，我们现在演示如何使用MediaSessionCompat，下面演示了如何构造一个MediaSessionCompat以及构造完之后通常需要做的三件事情：设置合理的flag，设置回调（在5.0开始会响应onPlay，onPause等等回调），设置激活。

```
ComponentName mediaButtonReceiver =
  new ComponentName(context, RemoteControlReceiver.class);
MediaSessionCompat mediaSession =
  new MediaSessionCompat(context,
    tag, // Debugging tag, any string
    mediaButtonReceiver,
    null);

mediaSession.setFlags(
  MediaSessionCompat.FLAG_HANDLES_MEDIA_BUTTONS |
  MediaSessionCompat.FLAG_HANDLES_TRANSPORT_CONTROLS);
mediaSession.setCallback(this); // a MediaSessionCompat.Callba

mediaSession.setActive(true);
```

搭建好了MediaSessionCompat之后，还需要通过`MediaMetadataCompat`来传递播放的资料信息，通过`PlaybackStateCompat`来传递播放的状态信息。



做了上面那些操作之后，MediaSessionCompat的任务就算是完成了。



5）使用Toolbar替代ActionBar

自从MaterialDesign开始，Android官方就开始使用Toolbar替代了原来的ActionBar，现在Toolbar已经加入Support兼容包。Toolbar是一个相比起ActionBar更加丰富，更加灵活的组件，另外它的布局本身还是View Hierarchy的一部分，这就意味着可以对Toolbar执行动画操作，增加点击滑动事件等等，甚至我们还可以在一个页面里面加入两个Toolbar。

为了启用Toolbar，首先要做的事情就是关闭当前Activity的ActionBar。我们可以通过使得

Activity的主题继承`Theme.AppCompat.NoActionBar`，然后在对应的XML布局文件中，添加类似下面的toolbar布局信息：

```
<!--Toolbar-->
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:minHeight="?attr/actionBarSize">
</android.support.v7.widget.Toolbar>
```

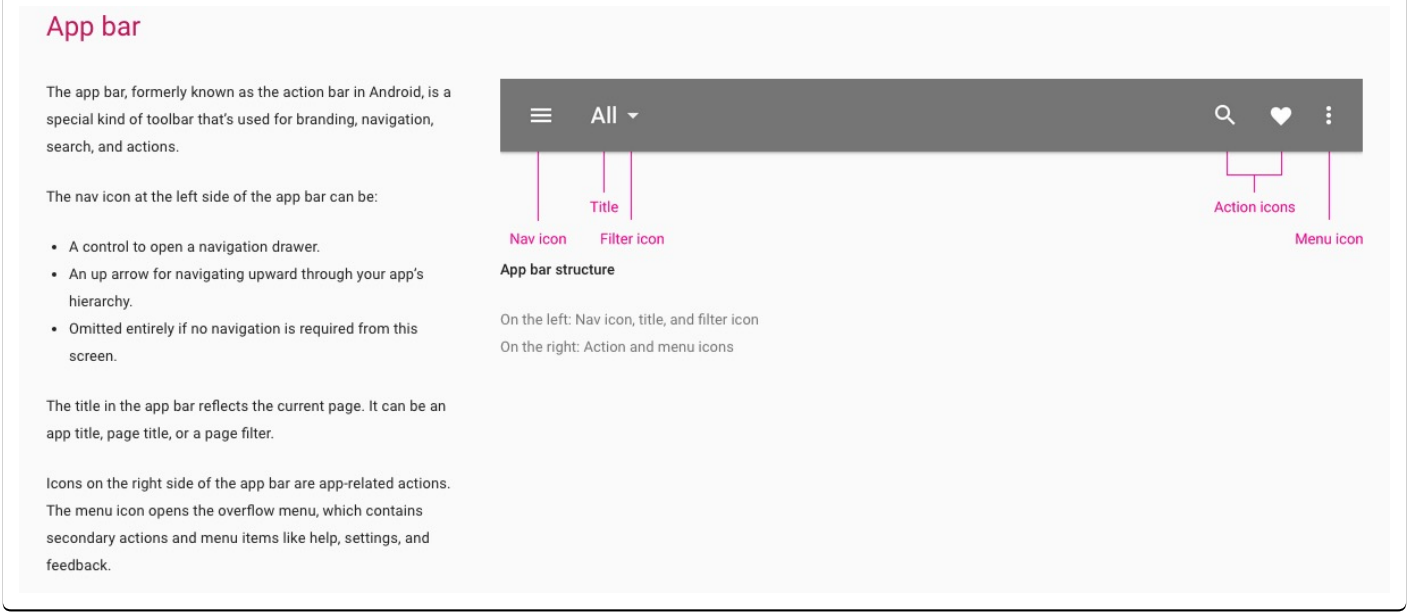
在布局文件中添加Toolbar的信息之后，需要启动Toolbar替代ActionBar，需要像下面一样做设置：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar_layout);
    setSupportActionBar(toolbar);
}
```

关于更多Toolbar的细节，请参考官方文档https://developer.android.com/reference/android/support/v7/widget/Toolbar.html

6）使用AppBarLayout并处理滑动手势

AppBarLayout是一个在android.support.design兼容包（这里有关于该兼容包的官方博客介绍http://android-developers.blogspot.com/2015/05/android-design-support-library.html）里面的新推出的组件，它是一个垂直方向的LinearLayout，包装了很多Material Design的设计元素，例如滑动手势的处理。我们可以使用`app:layout_scrollFlags`这样的标签来设置滑动的行为表现。关于App Bar，官方还有这样一段描述：



使用AppBarLayout需要注意下面几个要点：

- 首先，AppBarLayout必须作为`CoordinatorLayout`的直接子View
- 其次，在AppBarLayout里面必须包含一个ToolBar
- 最后，在CoordinatorLayout里面可以添加那些可以滑动的组件，例如RecyclerView

一个标准的布局文件应该是类似下面结构的：

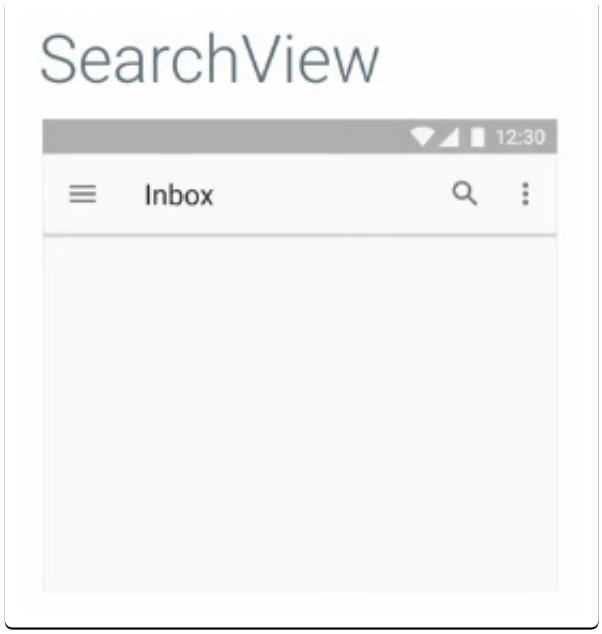
```
<android.support.design.widget.CoordinatorLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:app="http://schemas.android.com/apk/res-auto"
       android:layout_width="match_parent"
       android:layout_height="match_parent">

    <! -- Your Scrollable View -->
   <android.support.v7.widget.RecyclerView
           android:layout_width="match_parent"
           android:layout_height="match_parent"
           app:layout_behavior="@string/appbar_scrolling_view_behavior" />

   <android.support.design.widget.AppBarLayout
           android:layout_width="match_parent"
           android:layout_height="wrap_content">
  <android.support.v7.widget.Toolbar
                  ...
                  app:layout_scrollFlags="scroll|enterAlways">

       <android.support.design.widget.TabLayout
                  ...
                  app:layout_scrollFlags="scroll|enterAlways">
    </android.support.design.widget.AppBarLayout>
</android.support.design.widget.CoordinatorLayout>
```

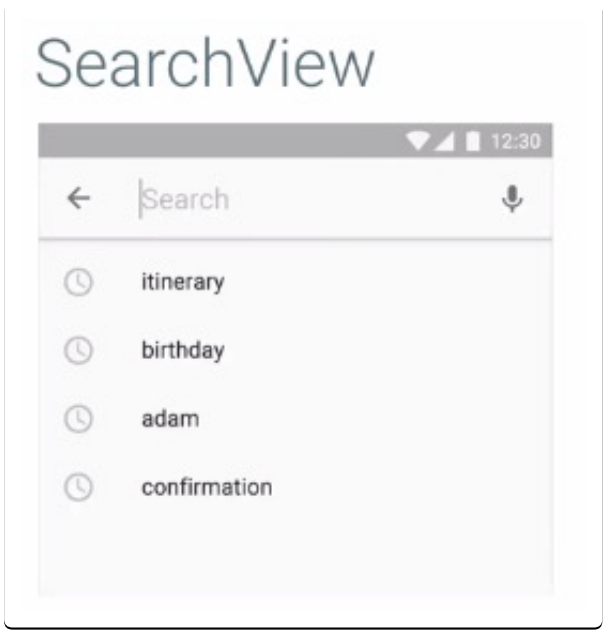
我们需要注意，在Toolbar里面设置的layout_scrollFlags会影响到滑动之后的显示效果，请看

下面的具体解释：

7）使用SearchView来实现搜索功能

关于在使用搜索的时候及时显示给用户的候选词，会需要根据数据的类型以及具体的情况做不同的处理，这里先暂时不讨论那一块的内容。





上面的图示已经清楚的演示了使用SearchView处理搜索的通常情况，关于如何实现这个功能，需要做到以下几个步骤：

- 在Menu的XML文件中，声明使用SearchView

```
<item android:id="@+id/action_search"
      android:title="@string/action_search"
      android:icon="@drawable/ic_action_search"
      app:showAsAction="ifRoom|collapseActionView"
      app:actionViewClass="android.support.v7.widget.SearchView" />
```

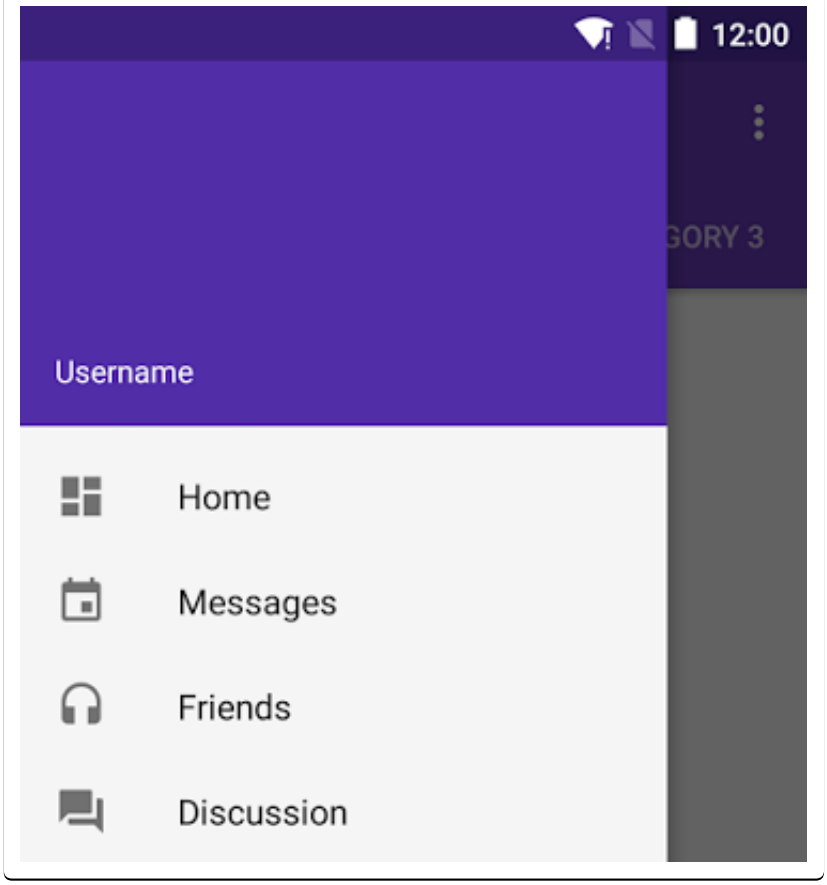
- 在onCreateOptionsMenu的回调函数里面获取到SearchView，并设置监听（请注意使用MenuItemCompat的那行代码，否则会出现很多兼容性问题，获取不到这个View等等奇怪的BUG），在监听回调里面处理业务逻辑

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.main_activity_actions, menu);
  MenuItem searchItem = menu.findItem(R.id.action_search);
  SearchView searchView = (SearchView)
      MenuItemCompat.getActionView(searchItem);

  searchView.setOnQueryTextListener(
      new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextChange (String newText) {
          // Text has changed, apply filtering?
          return false;
        }
        @Override
        public onQueryTextSubmit(String query) {
          // Perform the final search
        }
  });
}
```

至此，其实就已经实现了一个基础的搜索功能。但是，如果为了能够让自己的应用的某些功能被Android系统的Search功能检索到，我们就需要做更进一步的操作，例如定义Searchable，实现一个SearchableActivity，响应系统的Search行为等等。国内的应用很少会去关注这个功能，这里就不展开了，感兴趣点击下面的链接进一步学习：https://developer.android.com/guide/topics/search/index.html

8）Navigation Drawer, DrawerLayout, NavigationView

Navigation Drawer是Material Design当中很重要的一种设计元素，为了能够快速的实现这种设计，Android在新的design support包里面为我们提供了DrawerLayout与NavigationView。



实现这样的一个功能，我们需要在XML中写下类似下面的布局文件

```
<android.support.v4.widget.DrawerLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">

    <!-- your content layout -->

    <android.support.design.widget.NavigationView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            app:headerLayout="@layout/drawer_header"
            app:menu="@menu/drawer"/>
</android.support.v4.widget.DrawerLayout>
```

在NavigationView中有两个重要的标签，app:headerLayout与app:menu，分别代表了拉出菜单的顶部布局与下面的菜单项列表。创建菜单项列表，可以使用类似下面的MenuItem文件：

```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/navigation_item_1"
        android:checked="true"
        android:icon="@drawable/ic_android"
        android:title="@string/navigation_item_1"/>
    <item
        android:id="@+id/navigation_item_2"
        android:icon="@drawable/ic_android"
        android:title="@string/navigation_item_2"/>
</group>
```

`android:checked`表示当前选中的Item，会被系统Highlight出来。除了上面简单的平铺的菜单，还可以使用菜单嵌套的方式实现多级的菜单。关于点击具体某个菜单的时候的监听与响应，需要做如下的设置：

```
private void setupDrawerContent(NavigationView navigationView) {
    navigationView.setNavigationItemSelectedListener(
            new NavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(MenuItem menuItem) {
            menuItem.setChecked(true);
            mDrawerLayout.closeDrawers();
            return true;
        }
    });
```

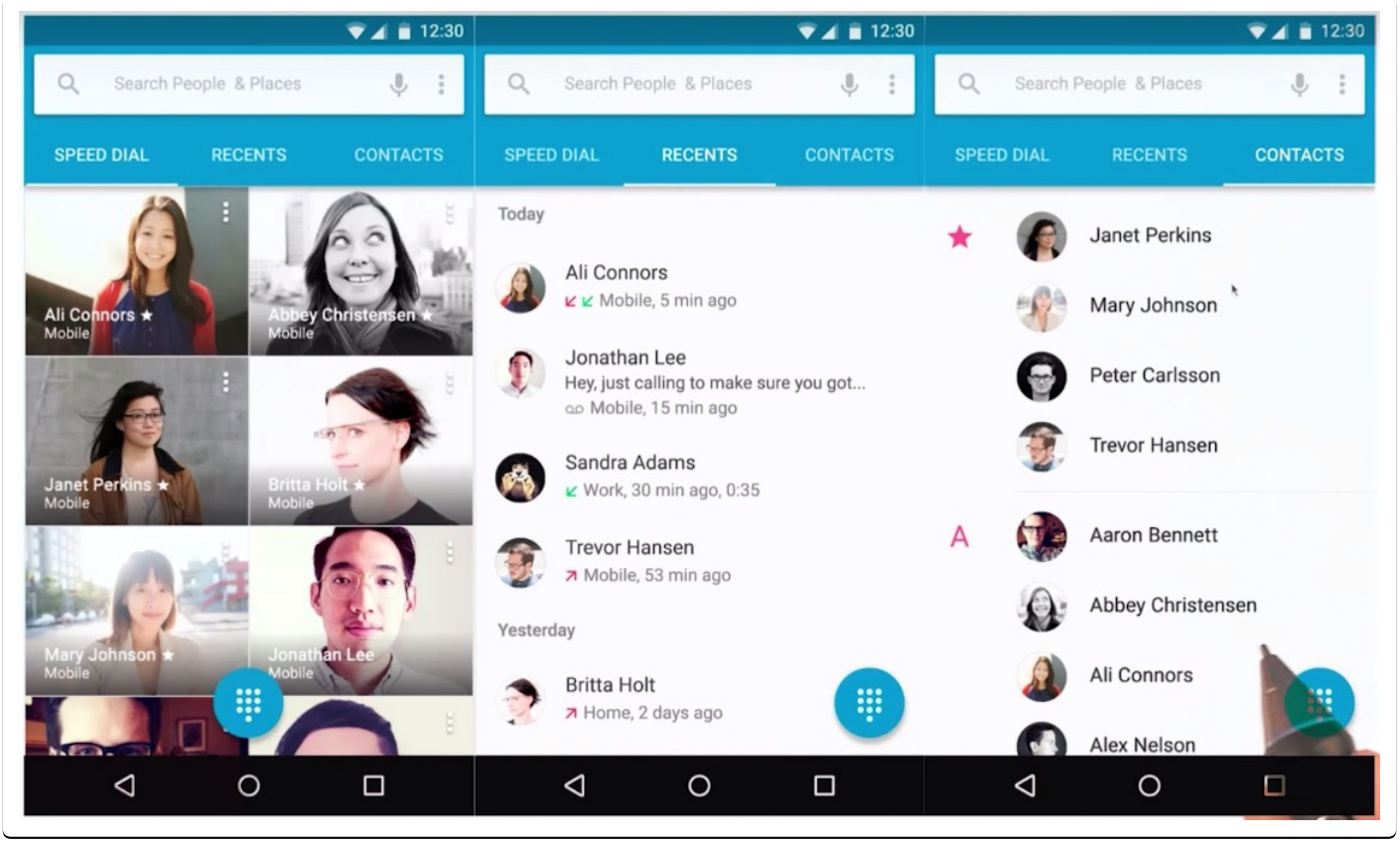
除了点击菜单事件，我们还需要处理整个侧滑菜单的打开与关闭事件，当我们给DrawerLayout设置了setDrawerListener之后，可以得到下面两个回调：

```
/** Called when a drawer has settled in a completely closed state. */
public void onDrawerClosed(View view) {
    super.onDrawerClosed(view);
    getActionBar().setTitle(mTitle);
    invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
}

/** Called when a drawer has settled in a completely open state. */
public void onDrawerOpened(View drawerView) {
    super.onDrawerOpened(drawerView);
    getActionBar().setTitle(mDrawerTitle);
    invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
}
```

大多数时候，侧滑菜单都是从左到右滑出的，但是我们也可以做到从右往左滑出，只需要在DrawerLayout的菜单布局LinearLayout里面修改一下margin的相关属性即可：

```
<LinearLayout
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginStart="16dp"
    android:layout_width="wrap_content"
    android:orientation="vertical">
```

9）Tabs and ViewPager



ViewPager是Android上面实现横向滑动的基础组件，能够帮助大家迅速搭建类似上面图示一样的左右滑动交互设计。ViewPager需要使用PagerAdapter来提供内容，除了PagerAdpater，Android还提供了FragmentPagerAdpater与FragmentStatePagerAdapter，前者会把所有的fragment都保存在内存中，以便提高切换速度，后者仅仅保留了fragment状态信息，fragment还是会进行正常的重建与销毁。一个典型的使用demo代码如下：

```
class FixedTabsPagerAdapter extends FragmentPagerAdapter {

    public FixedTabsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public int getCount() {
        return 3;
    }

    @Override
    public Fragment getItem(int position) {
        switch(position) {
            case 0:
                return new SpeedDialFragment();
            case 1:
                return new RecentsFragment();
            case 2:
                return new ContactsFragment();
            default:
                return null;
        }
    }
```

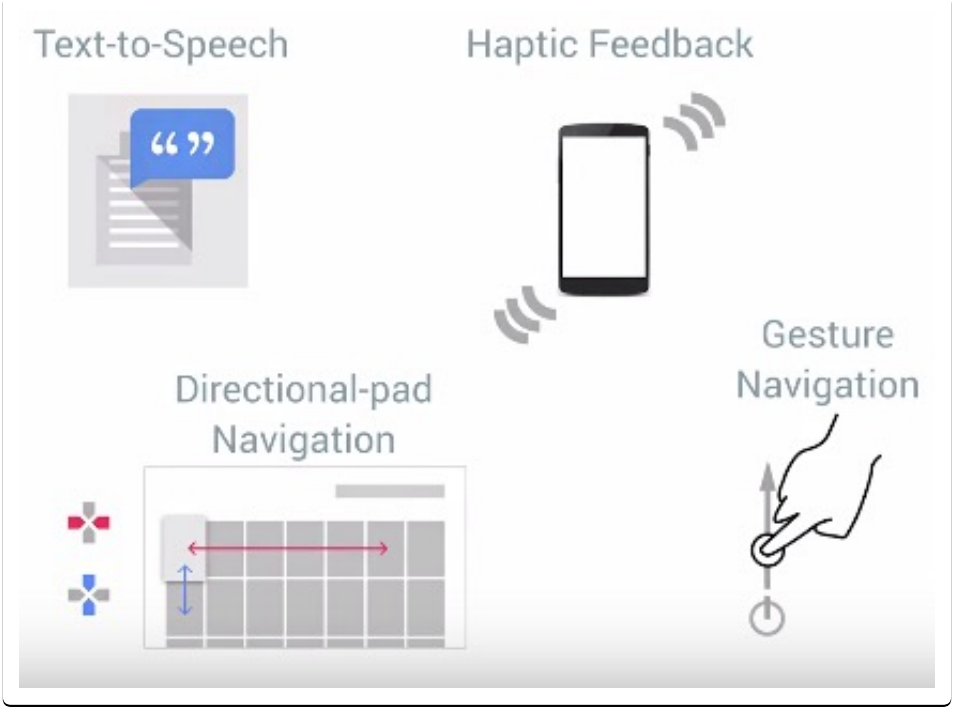
为了实现前面图示的Tab与ViewPager的绑定，我们可以使用Android Design Support Library提供的`TabLayout`，仅仅需要按照下面的代码示例一样把TabLayout与ViewPager做一个绑定，就能够实现左右滑动与点击Tab快速切换的功能：

```
ViewPager viewPager = (ViewPager) findViewById(R.id.view_pager);
PagerAdapter pagerAdapter =
        new FixedTabsPagerAdapter(getSupportFragmentManager());
viewPager.setAdapter(pagerAdapter);

TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
tabLayout.setupWithViewPager(viewPager);
```

关于Material Design里面的Tabs设计，请再参考http://www.google.com/design/spec/components/tabs.html以及官方Training课程里面的http://developer.android.com/training/implementing-navigation/lateral.html

10）Making Apps Accessible

为了照顾部分视力或者听觉不好的用户，我们需要做一定的处理使得自己的应用能够被每一个可用。Android系统为了帮助应用实现辅助功能，提供了诸如文本朗读，触感反馈，指向炳导航，手势导航等等功能来更好的帮助用户使用这些应用。

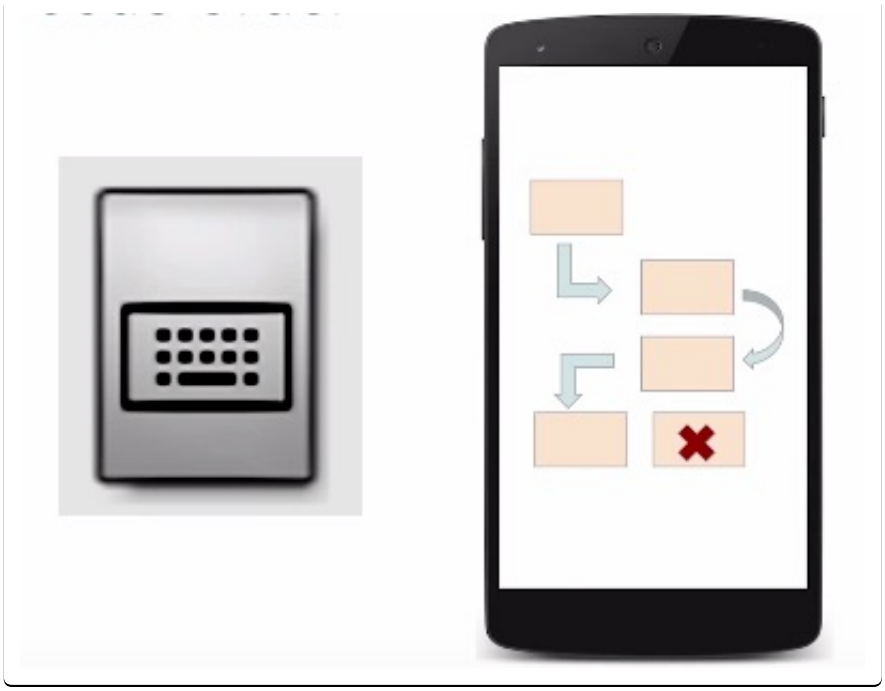


为了确保你的应用能够被Android系统提供的辅助功能正常使用，需要做以下三个步骤的检查：

- **Content Description**：确保类似ImageView，ImageButton，CheckBox等组件都包含了content descrption。

```
<ImageButton
    android:id="@+id/add_note_button"
    android:src="@drawable/add_note"
    android:contentDescription="@string/add_note"/>
```

- **Focus Order**：确保给布局里面的关键元素增加了Focus的指示顺序，只有这样，辅助功能才能够在指向导航的时候帮助用户按照指定的顺序来聚焦界面元素。



```
<LinearLayout android:orientation="horizontal"
        ... >
    <EditText android:id="@+id/edit"
        android:nextFocusDown="@+id/text"
        ... />
    <TextView android:id="@+id/text"
        android:focusable="true"
        android:text="Hello, I am a focusable TextView"
        android:nextFocusUp="@id/edit"
        ... />
</LinearLayout>
```

- **Feedback Mechanisms**：确保部分关键的操作有多个反馈，例如当短信来的时候，既有声音也有震动，这样才能够确保听力不好的用户可以通过震动的反馈来感知到响应。

更多关于辅助功能的知识，请参考http://developer.android.com/guide/topics/ui/accessibility/checklist.html