作为Android 3.0之后引入的新的对象，ActionBar可以说是一个方便快捷的导航神器。它可以作为活动的标题，突出活动的一些关键操作（如“搜索”、“创建”、“共享”等）、作为菜单的灵活使用，还可以实现类似TabWidget的标签功能以及下拉导航的功能，系统能够很好根据不同的屏幕配置来适应ActionBar的外观，配合起Fragemtn可谓是十分强大。

          那么，对于今天的主角ActionBar怎么去添加？在Android3.0默认主题HloleFraphic（全息）主题中，已经创造了ActionBar，所以只要targetSdkVersion的值不低于11，创建的Activity中默认都会带有ActionBar**。**例如：

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. <manifest ... >
2. <uses-sdk android:minSdkVersion="4"
3. android:targetSdkVersion="11" />
4. ...
5. </manifest>

          当然了，如果你不想为一个特定的Activity设置Action Bar，设置Activity主题为Theme.Holo.NoActionBar。

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. <activity android:theme="@android:style/Theme.Holo.NoActionBar">

          或者在运行时通过调用hide()隐藏Action Bar。自然也有show()。

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. ActionBar actionBar = getActionBar();
2. actionBar.hide();

          下面我们从**下拉导航**、**视窗操作**、**标签导航**三个方面逐一讨论ActionBar

**第一，下拉导航**

                    下拉导航最典型的应用场景就是在Google+中的使用，效果如下图：

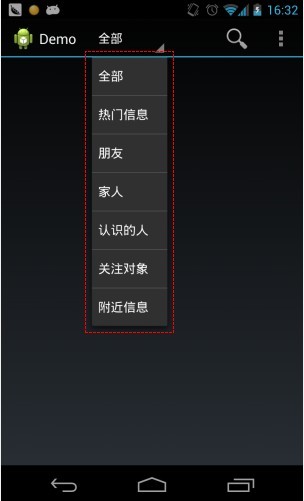
          

图1；Google+                             图2：本文示例

实现此效果分如下几个步骤：

          1.初始化一个SpinnerAdapter

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. SpinnerAdapter mSpinnerAdapter = ArrayAdapter.createFromResource(this,
2. R.array.action\_list,
3. android.R.layout.simple\_spinner\_dropdown\_item);

  2.生成一个OnNavigationListener来响应ActionBar的菜单项点击操作

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. /\*\*
2. \* 在这里配合Fragment，实现不同的页面导航
3. \*/
4. OnNavigationListener mOnNavigationListener = new OnNavigationListener() {
6. @Override
7. public boolean onNavigationItemSelected(int position, long itemId) {
8. Fragment newFragment = null;
9. switch (position) {
10. case 0:
11. newFragment = new Fragment1();
12. break;
13. case 1:
14. newFragment = new Fragment2();
15. break;
16. case 2:
17. newFragment = new Fragment3();
18. break;
19. default:
20. break;
21. }
22. getSupportFragmentManager().beginTransaction()
23. .replace(R.id.container, newFragment, strings[position])
24. .commit();
25. return true;
26. }
27. };

   3，将生成好的适配去和监听器塞给ActionBar

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. actionBar = getActionBar();
2. actionBar.setNavigationMode(ActionBar.NAVIGATION\_MODE\_LIST);//导航模式必须设为NAVIGATION\_MODE\_LIST
3. actionBar.setListNavigationCallbacks(mSpinnerAdapter,
4. mOnNavigationListener);

**第二，操作视窗**

          先上效果图

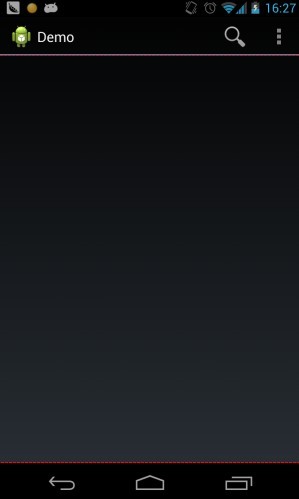
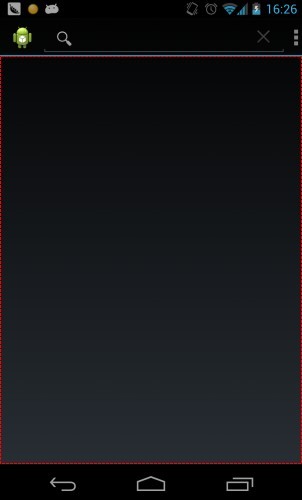
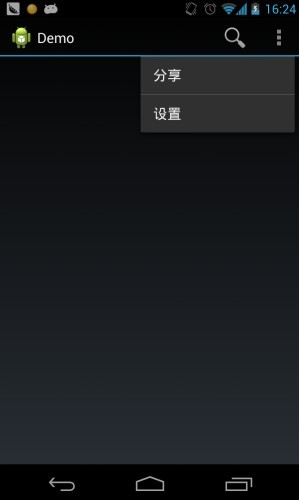
    

        图3                                                                             图4                                                                   图5

          在上面的操作视窗里，增加了一个用于搜索的可选菜单项以及分享和设置的两个**自定义ActionProVider**。那么如何在一个活动中，在已有的ActionBar上添加这些操作视窗。同创建可选菜单一样，定义options.xml的menu文件如下：

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. <?xml version="1.0" encoding="utf-8"?>
2. <menu xmlns:android="http://schemas.android.com/apk/res/android" >
4. <item
5. android:id="@+id/menu\_search"
6. android:actionViewClass="android.widget.SearchView"
7. android:icon="@drawable/ic\_menu\_search"
8. android:showAsAction="ifroom"
9. android:title="搜索"/>
10. <item
11. android:id="@+id/menu\_share"
12. android:actionProviderClass="android.widget.ShareActionProvider"
13. android:showAsAction="never"
14. android:title="分享"/>
15. <item
16. android:id="@+id/menu\_setting"
17. android:actionProviderClass="com.example.tabdemo.MyActionProvider"
18. android:showAsAction="never"
19. android:title="设置">
20. <menu>
21. <item
22. android:id="@+id/menu\_theme"
23. android:actionProviderClass="com.example.tabdemo.MyActionProvider"
24. android:showAsAction="always|withText"
25. android:title="更换主题"/>
26. <item
27. android:id="@+id/menu\_system"
28. android:actionProviderClass="com.example.tabdemo.MyActionProvider"
29. android:showAsAction="always|withText"
30. android:title="系统设置"/>
31. </menu>
32. </item>
33. </menu>

          仔细观察可以发现每个Item里都包含如下这两个属性：

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. android:actionProviderClass="com.example.tabdemo...."

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. android:showAsAction=""

          对于actionProviderClass属性用来指定一个构建视窗所使用的布局资源，除了使用actionProviderClass指定外，还可以使用actionLayout或者actionViewClass都可以。SearchView和ShareActionProvider都是系统自带的ActionProvider，MyActionProvider是我们要重写的，后面将会看到如何去自定义一个ActionProvider。

          showAsAction属性共有五个值：ifRoom、never、always、withText、collapseActionView，可以混合使用。

|  |  |
| --- | --- |
| ifRoom | 会显示在Item中，但是如果已经有4个或者4个以上的Item时会隐藏在溢出列表中。当然个 数并不仅仅局限于4个，依据屏幕的宽窄而定 |
| never | 永远不会显示。只会在溢出列表中显示，而且只显示标题，所以在定义item的时候，最好 把标题都带上。 |
| always | 无论是否溢出，总会显示。 |
| withText | withText值示意Action bar要显示文本标题。Action bar会尽可能的显示这个 标题，但是，如果图标有效并且受到Action bar空间的限制，文本标题有可 能显示不全。 |
| collapseActionView | 声明了这个操作视窗应该被折叠到一个按钮中，当用户选择这个按钮时，这个操作视窗展开。否则， 这个操作视窗在默认的情况下是可见的，并且即便在用于不适用的时候，也要占据操作栏的有效空间。 一般要配合ifRoom一起使用才会有效果。 |

**注： 当你的应用程序正在Android4.0（API 级别 14）或以上的版本上运行，那么还有一种叫做“分隔操作栏”的额外模式对action bar有效。当你启用分隔操作栏模式时，在屏幕的底部会显示一个独立的横条，用于显示Activity在窄屏设备（如竖屏手机）上运行时的所有操作项。这里我们不过过多描述，有兴趣自己去研究。**

          就像加载menu一样，在activity的onCreateOptionsMenu方法里调用上述的xml文件：

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. getMenuInflater().inflate(R.menu.options, menu);
3. //搜索视窗，因为showAsAction="ifRoom"，所以图三中出现了搜索按钮
4. SearchView searchView = (SearchView) menu.findItem(R.id.menu\_search)
5. .getActionView();
7. //分享视窗，因为showAsAction="never"，所以只能在溢出菜单中才看见到
8. ShareActionProvider mShareActionProvider = (ShareActionProvider) menu
9. .findItem(R.id.menu\_share).getActionProvider();
10. Intent shareIntent = new Intent(Intent.ACTION\_SEND);
11. shareIntent.setType("image/\*");
12. mShareActionProvider.setShareIntent(shareIntent);
14. //设置视窗，MyActionProvider就是我们自定义的ActionProvider
15. MyActionProvider myactionprovider = (MyActionProvider) menu.findItem(
16. R.id.menu\_setting).getActionProvider();
17. return super.onCreateOptionsMenu(menu);

          显然，当成功运行的时候，结果如图三，当点击搜索按钮时，搜索按钮立刻变成了如图四的样子，变成可折叠的操作视窗。

          如何自定义操作视窗，定义一个类MyActionProvider继承自ActionProvider，并实现它的两口回调函数即可。如下：

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. /\*\*
2. \* @ClassName: MyActionProvider
3. \* @Description: 自定义一个视窗操作器，实现构造函数和onCreateActionView即可
4. \* @author yuxianglong
5. \* @date 2013-7-11 下午3:13:44
6. \*
7. \*/
8. public class MyActionProvider extends ActionProvider{
10. private Context context;
11. private LayoutInflater inflater;
12. private View view;
13. private ImageView button;
14. public MyActionProvider(Context context) {
15. super(context);
16. // TODO Auto-generated constructor stub
17. this.context = context;
18. inflater = LayoutInflater.from(context);
19. view = inflater.inflate(R.layout.myactionprovider, null);
20. }

23. @Override
24. public View onCreateActionView() {
25. // TODO Auto-generated method stub
26. button = (ImageView) view.findViewById(R.id.button);
27. button.setOnClickListener(new View.OnClickListener() {
29. @Override
30. public void onClick(View v) {
31. // TODO Auto-generated method stub
32. Toast.makeText(context, "是我，没错", Toast.LENGTH\_SHORT).show();
33. }
34. });
35. return view;
36. }
38. }

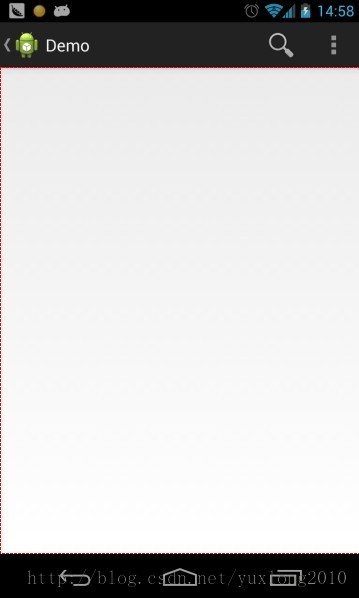
          如此一来，只要在options.xml里直接引用。运行成功效果如图五，点击溢出菜单，设置按钮出来了，如果继续点下去，回调出它的子菜单，因为我们在options.xml里给自定义的ActionProvider分配了子菜单。

                 当然了，最显眼的就是处理Action Bar上的应用程序图标，平时玩手机多的同学应该可以发现，好多应用的图标都是可以点击的，而且大多数都是回到了上一个Activity，或者说是主Activity。那么，如何触发应用程序图标呢，说白了应用程序图标也是一个菜单，并且其id是规定死的，所以只要我们在onOptionsItemSelected方法里去捕捉它的点击事件，做出响应：

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. @Override
2. public boolean onOptionsItemSelected(MenuItem item) {
3. switch (item.getItemId())
4. {
5. case android.R.id.home:
6. Intent intent = new Intent(this, HomeActivity.class);
7. intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);
8. startActivity(intent);
9. return true;
10. default:
11. return super.onOptionsItemSelected(item);
12. }
13. }

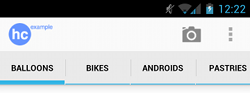
          我们给Intent添加了FLAG\_ACTIVITY\_CLEAR\_TOP标识，该标识的作用是在回到HomeActivity时，把在堆栈中处于HomeActivity上面的活动全部清除。如果这是候运行程序的话，如果系统版本小于4.0的话，是可以正常跑起来的，达到想要的效果，但如果系统大于或者等于4.0的话，那么点击应用图标是无效的。必须加上setHomeButtonEnabled=true，4.0一下 默认为true。如果还想要一个回退箭头的话，再加上一句setDisplayHomeAsUpEnabled(true);效果如下：



          这里我在扩展一下：使用过Navigation Drawer的同学应该了解，这里点击应用程序图标通常会作为拉出导航抽屉。通常在那种情况下是把活动的onOptionsItemSelected，传送给ActionBarDrawerToggle的onOptionsItemSelected。不多说了，感兴趣的同学自己去研究，后面会把Navigation Bar写出来。

**第三，导航选项标签**

          当你想要在一个Activity中提供导航选择标签时，使用操作栏的选项标签是一个非常好的选择（而不是使用TabWidget类），**因为系统会调整操作栏选项标签来适应不同尺寸的屏幕的需要，在屏幕足够宽的时候，导航选项标签会被放到主操作栏中；当屏幕太窄的时候，选项标签会被放到一个分离的横条中。**如图下：



http://img.blog.csdn.net/20130711191751453?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQveXV4bG9uZzIwMTA=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

          要使用选项标签在Fragmengt之间切换，选择一个选项标签时执行一个Fragment事务，布局里包含一个用于放置跟每个Fragment对象关联的选项标签的ViewGroup对象。该对象有一个资源ID，以便能够在选项标签的切换代码中能够引用它。Activity的布局文件activity\_main.xml定义如下：

**[html]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2. xmlns:tools="http://schemas.android.com/tools"
3. android:id="@+id/container"
4. android:layout\_width="match\_parent"
5. android:layout\_height="match\_parent"
6. tools:context=".MainActivity"
7. tools:ignore="MergeRootFrame" />

          这里的ViewGroup为FragmentLayout。Activity代码如下：

**[java]** [view plain](http://blog.csdn.net/yuxlong2010/article/details/9299507) [copy](http://blog.csdn.net/yuxlong2010/article/details/9299507)

1. /\*
2. \* @ClassName: MainActivity
3. \* @Description: 继承自FragmentActivity,作为Fragment的holder-Activity使用，
4. \*               实现TabListener接口，当切Tab的时候达到切换Fragment的效果
5. \* @author yuxianglong
6. \* @date 2013-7-11 下午7:40:35
7. \*
8. \*/
9. public class MainActivity extends FragmentActivity implements
10. ActionBar.TabListener {
11. private ActionBar actionBar;
13. @Override
14. protected void onCreate(Bundle savedInstanceState) {
15. super.onCreate(savedInstanceState);
16. setContentView(R.layout.activity\_main);
17. actionBar = getActionBar();
18. actionBar.setNavigationMode(ActionBar.NAVIGATION\_MODE\_TABS);// 导航模式必须设为NAVIGATION\_MODE\_Tabs
20. // For each of the sections in the app, add a tab to the action bar.
22. actionBar.addTab(actionBar.newTab().setText(R.string.title\_section1)
23. .setTabListener(this));
24. actionBar.addTab(actionBar.newTab().setText(R.string.title\_section2)
25. .setTabListener(this));
26. actionBar.addTab(actionBar.newTab().setText(R.string.title\_section3)
27. .setTabListener(this));
29. }

32. @Override
33. public void onTabSelected(ActionBar.Tab tab,
34. FragmentTransaction fragmentTransaction) {
35. // When the given tab isselected, show the tabcontents in the
36. // //container view.
37. Fragment fragment3 = null;
38. Fragment fragment1 = null;
39. Fragment fragment2 = null;
40. switch (tab.getPosition()) {
41. case 0:
42. if (fragment1 == null) {
43. fragment1 = new Fragment1();
44. }
45. getSupportFragmentManager().beginTransaction()
46. .replace(R.id.container, fragment1).commit();
47. break;
48. case 1:
49. if (fragment2 == null) {
50. fragment2 = new Fragment2();
51. }
52. getSupportFragmentManager().beginTransaction()
53. .replace(R.id.container, fragment2).commit();
54. break;
55. case 2:
56. if (fragment3 == null) {
57. fragment3 = new Fragment3();
58. }
59. getSupportFragmentManager().beginTransaction()
60. .replace(R.id.container, fragment3).commit();
61. break;
63. default:
64. break;
65. }
67. }
69. @Override
70. public void onTabUnselected(ActionBar.Tab tab,
71. FragmentTransaction fragmentTransaction) {
72. }
74. @Override
75. public void onTabReselected(ActionBar.Tab tab,
76. FragmentTransaction fragmentTransaction) {
77. }
79. }

          最后跑起来的效果如下：



          至此ActionBar的一些常见使用场景，我们就熟悉了，后面继续研究ActionBar的外观样式