第 28 章 搜　索

接下来的任务是为PhotoGallery应用添加搜索Flickr网站图片的功能。本章将介绍如何以Android特有的方式，整合搜索功能到应用中。

或者说是以多种Android特有的方式实现搜索功能的添加。实际上，搜索功能从一开始就整合进Android系统，但该功能截至目前已发生了巨大的变化。类似菜单功能的演变，新的搜索实现代码已基于当前API构建。因此，即使按照旧模式创建搜索功能，实现的也是全功能的现代Jelly Bean搜索功能。

28.1　搜索 Flickr 网站

要搜索Flickr网站，我们需调用`flickr.photos.search`方法。以下为调用该方法搜索“red”文本的使用样例：

```
http://api.flickr.com/services/rest/?method=flickr.photos.search
&api_key=XXX&extras=url_s&text=red
```

该方法接受一些不同的新参数，可指定具体的搜索项，如一个字符串的查询参数。好消息是解析网站返回XML并获取结果的方式与之前完全相同。

参照代码清单28-1，在`FlickrFetchr`类中，添加新的搜索请求方法。`Search`和`getRecent`以同样的方式解析`GalleryItem`，因此应重构`fetchItems()`方法中的旧代码，形成一个名为`downloadGalleryItems(String)`的新方法。特别要注意的是，`fetchItems()`旧方法里的URL代码已被移入新版本的`fetchItems()`方法，而不是被删除掉了。

代码清单28-1　添加Flickr搜索方法（FlickrFetchr.java）

```
public class FlickrFetchr {
    public static final String TAG = "PhotoFetcher";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "4f721bbafa75bf6d2cb5af54f937bb70";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String METHOD_SEARCH = "flickr.photos.search";
    private static final String PARAM_EXTRAS = "extras";
    private static final String PARAM_TEXT = "text";

    ...
```

```
    public ArrayList<GalleryItem> fetchItems() {
        ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();

        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                    .appendQueryParameter("method", METHOD_GET_RECENT)
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                    .build().toString();
            String xmlString = getUrl(url);
            Log.i(TAG, "Received xml: " + xmlString);
            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
            XmlPullParser parser = factory.newPullParser();
            parser.setInput(new StringReader(xmlString));

            parseItems(items, parser);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (XmlPullParserException xppe) {
            Log.e(TAG, "Failed to parse items", xppe);
        }
        return items;
    }

    public ArrayList<GalleryItem> fetchItems() {
        // Move code here from above
        String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_GET_RECENT)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .build().toString();
        return downloadGalleryItems(url);
    }

    public ArrayList<GalleryItem> search(String query) {
        String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_SEARCH)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .appendQueryParameter(PARAM_TEXT, query)
                .build().toString();
        return downloadGalleryItems(url);
    }
}
```

downloadGalleryItems(String)方法共使用了两次，因为search和getRecent方法使用了相同的下载和URL解析代码。简单来说，搜索就是使用flickr.photos.search新方法，并传入已编码的查询字符串作为text参数。

接下来，在PhotoGalleryFragment.FetchItemsTask类中，利用测试代码调用搜索方法。这里，出于测试目的，我们将使用硬编码搜索字符串，如代码清单28-2所示。

代码清单28-2 硬编码的搜索字符串（PhotoGalleryFragment.java）

```
    private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
        @Override
        protected ArrayList<GalleryItem> doInBackground(Void... params) {
```

```
            String query = "android"; // Just for testing

            if (query != null) {
                return new FlickrFetchr().search(query);
            } else {
                return new FlickrFetchr().fetchItems();
            }
        }

        @Override
        protected void onPostExecute(ArrayList<GalleryItem> items) {
            ...
        }
    }

    ...
}
```

FetchItemsTask默认会执行原来的getRecent代码。如果搜索查询字符串的值非空（现在肯定非空），则FetchItemsTask将会获取搜索结果。

运行PhotoGallery并查看返回结果。应该可以看到一两张Android机器人的图片。

28.2　搜索对话框

本节将为PhotoGallery应用创建Android搜索界面。首先我们来创建老式的对话框界面。

28.2.1　创建搜索界面

自Honeycomb开始，Android移除了物理搜索键。不过，即使在这之前，也不能保证各Android机器都配备搜索键。如果面向3.0系统以前的设备做开发，依赖搜索的现代Android应用都必须在应用内提供搜索键。

具体实现并不困难，只需调用Activity.onSearchRequested()方法即可。该方法与点按搜索键执行的操作完全相同。

在res/menu/fragment_photo_gallery.xml中，添加XML搜索菜单定义。PhotoGallery应用也需要清除搜索查询，因此，再添加一个清除按钮定义。如代码清单28-3所示。

代码清单28-3　添加搜索菜单项（res/menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    />
  <item android:id="@+id/menu_item_clear"
    android:title="@string/clear_search"
    android:icon="@android:drawable/ic_menu_close_clear_cancel"
    android:showAsAction="ifRoom"
    />
</menu>
```

XML定义中引用了一些字符串资源，因此在string.xml文件中添加它们，如代码清单28-4所示。（稍后还会需要搜索提示字符串资源，因此在这里提前添加上。）

代码清单28-4 添加搜索字符串资源（res/values/strings.xml）

```
<resources>

    ...

    <string name="title_activity_photo_gallery">PhotoGalleryActivity</string>
    <string name="search_hint">Search Flickr</string>
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>

</resources>
```

接下来，我们来添加选项菜单的回调方法。撤销按钮暂不处理，搜索按钮就调用前述的onSearchRequested()方法，如代码清单28-5所示

代码清单28-5 添加选项菜单回调方法（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    ...
}

...

@Override
public void onDestroyView() {
    ...
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            getActivity().onSearchRequested();
            return true;
        case R.id.menu_item_clear:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行PhotoGallery应用，确认菜单界面能够正确显示，如图28-1所示。

图28-1　搜索界面

不过，当前按下搜索键并无响应。onSearchRequested()方法要能工作，我们必须设置PhotoGalleryActivity为可搜索activity。

28.2.2　可搜索的activity

完成两处文件配置可使activity支持搜索。首先是独立的XML配置文件。该XML配置文件包含一个名为searchable的元素节点，用来描述搜索对话框应该如何显示自身。创建新目录res/xml，并在其中创建一个名为searchable.xml的文件。参照代码清单28-6，添加一个简单的searchable元素节点及其相应内容。

代码清单28-6　搜索配置（res/xml/searchable.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:hint="@string/search_hint"
  />
```

searchable.xml又称为搜索配置文件。Photogallery应用中，搜索配置仅需提供搜索文字提示和应用名称。

对于一些功能复杂的应用，搜索配置文件可能会包含过多的配置，如搜索建议、语音搜索、全局搜索配置、操作键配置、输入类型等等。不过，再简单，类似以上的基本搜索配置还是必需的。

另一处文件配置是应用的AndroidManifest.xml配置文件。首先需更改应用的*启动模式*，其次需为PhotoGalleryActivity声明附加的intent filter以及元数据信息。intent filter表明应用可监听搜索intent，元数据信息则将searchable.xml与目标activity关联起来。

通过以上文件配置工作，我们告诉Android的SearchManager，activity能够处理搜索任务，并同时将搜索配置信息提供给它。SearchManager属于操作系统级别的服务，负责展现搜索对话框并管理搜索相关的交互。

打开AndroidManifest.xml配置文件，添加两个元素节点以及android:launchMode属性定义，如代码清单28-7所示。

代码清单28-7 添加intent filter和元数据信息（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >

  ...

  <application
    ... >
    <activity
      android:name=".PhotoGalleryActivity"
      android:launchMode="singleTop"
      android:label="@string/title_activity_photo_gallery" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable"/>
    </activity>
  </application>

</manifest>
```

首先来介绍刚刚添加的两个元素。第一个附加元素是大家熟悉的intent filter定义。我们可调用startActivity(...)方法，发送包含android.intent.action.SEARCH操作的intent，实现搜索结果的传递。搜索查询将作为extra信息附加在intent上。因此，为表明activity能够处理搜索结果，我们为android.intent.action.SEARCH定义了一个intent filter。

第二个是另一个元数据标签。本书早在第16章就曾介绍过有关元数据的使用，但该元数据标签有点特殊。代替原来的android:value属性，它使用的是android:resource属性。以下示例代码显示了二者间的差异。这里假定在两个不同的元数据标签中引用了同一字符串资源：

```
<meta-data android:name="metadata.value"
  android:value="@string/app_name" />
<meta-data android:name="metadata.resource"
  android:resource="@string/app_name" />
```

倘若从元数据节点取出metadata.value的值，则会发现它包含了@string/app_name中保存的“PhotoGallery”字符串。而metadata:resource的值则会是资源的整数ID值。换句话说，metadata:resource的值实际为R.string.app_name的代码值。

实际使用时，SearchManager需要的是searchable.xml资源的整数ID，而不是XML文件的字符串值。因此，我们使用android:resource属性将文件资源ID提供给SearchManager。

activity标签中定义的android:launchMode属性又有什么作用呢？它是activity的*启动模式*。稍后，在编写代码接收搜索查询时，可了解它的更多信息。

完成配置后，搜索对话框应该可以使用了。运行PhotoGallery应用，然后点击菜单搜索按钮，如图28-2所示。

图28-2　搜索对话框

28.2.3　物理搜索键

在旧设备上点按物理搜索键时，可运行与手动调用onSearchRequested()方法相同的代码。如需进行功能可用性验证，可通过配置模拟器使用物理键盘，实现在任何Android 3.0以前版本的模拟器上使用物理搜索键。具体配置如图28-3所示。

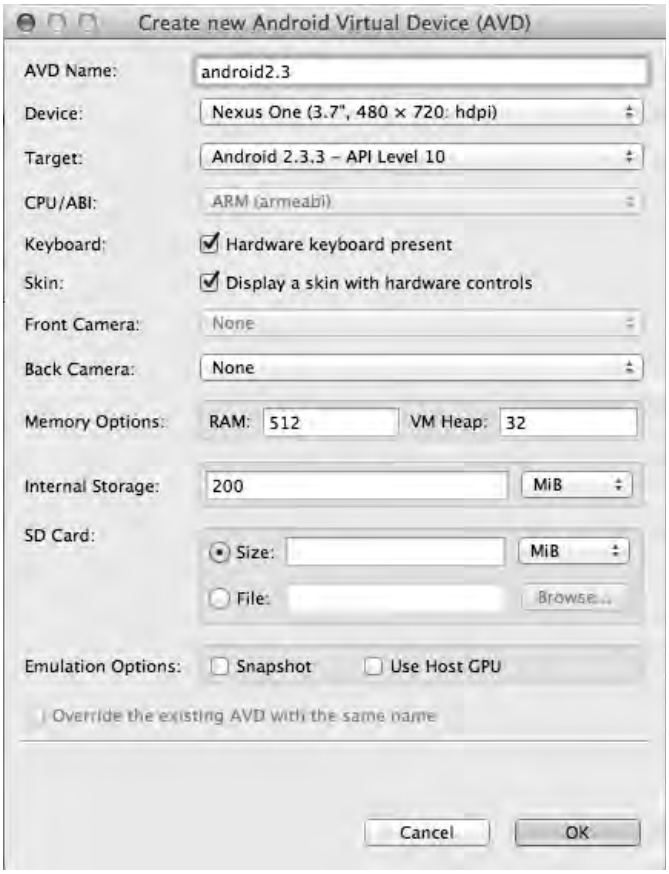

图28-3 添加物理键盘支持

28.2.4 搜索的工作原理

基于前面提到的可搜索activity概念，Android设计实现了搜索功能。可搜索activity由刚才创建的搜索intent filter和搜索配置元数据共同定义。

需要搜索时，我们按下物理搜索键，触发一系列搜索交互，直到搜索intent自身由系统接手处理。`SearchManager`会检查AndroidManifest.xml，以确认当前activity是否支持搜索。如果支持，一个搜索对话框activity就会显示在当前activity上。然后，通过发送新的intent，搜索对话框activity触发搜索，如图28-4所示。

这意味着点按搜索按钮通常会启动一个新的activity。但在PhotoGallery应用中，系统并没有启动新的activity。为什么？这是因为添加了`android:launchMode="singleTop"`后（代码清单28-7），我们改变了启动模式。

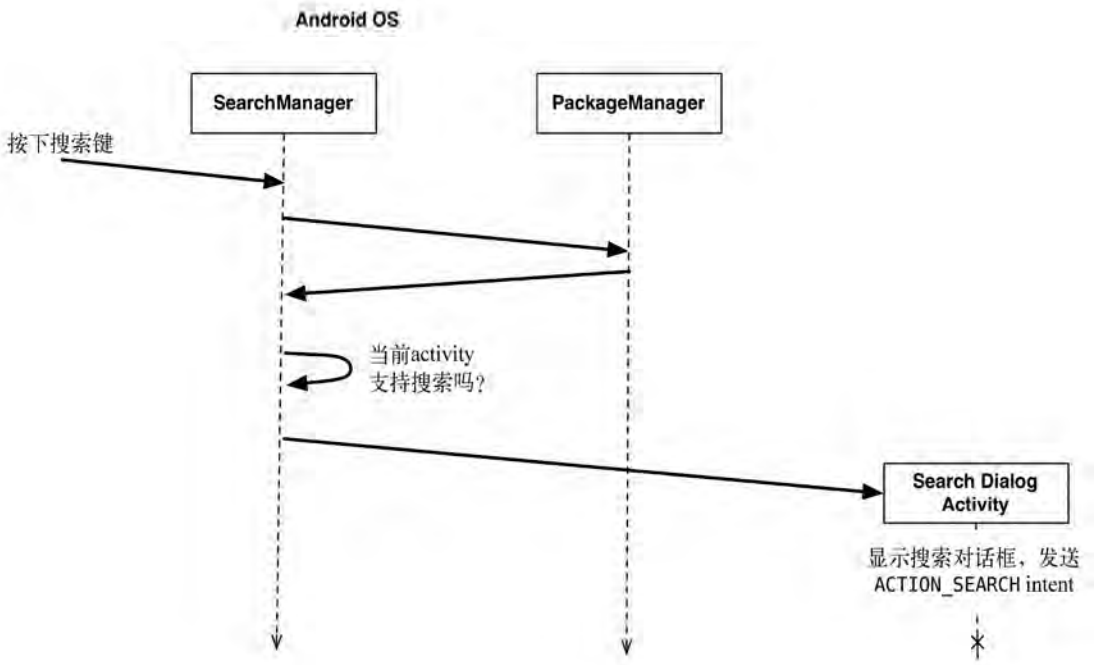

图28-4　系统搜索

28.2.5　启动模式与新的intent

什么是启动模式？启动模式决定了activity在收到新intent时的启动方式。有时，在activity发送intent启动另一个activity时，启动模式也决定了activity的行为方式。启动模式分为许多种，如表28-1所示。

表28-1　各种不同的启动模式

启动模式	行　为
`standard`	默认行为模式。针对每一个收到的新intent，都启动新的activity
`singleTop`	如果activity实例已经处在回退栈的顶端，则不创建新的activity，而直接路由新intent给现有activity
`singleTask`	在自身task中启动activity。如果task中activity已存在，则清除回退栈中该activity上的任何activity，然后路由新intent给现有activity
`singleInstance`	在自身task中启动activity。该activity是task中唯一的activity。如果任何其他activity从该task中启动，它们都各自启动到自己的task中。如果task中activity已存在，则直接路由新intent给现有activity

目前为止，我们开发的全部activity都使用了standard启动模式。该行为模式我们早已熟悉，即当intent以standard启动模式启动一个activity时，都会创建activity新实例并添加到回退栈中。

以上activity的行为模式并非永远适用于`PhotoGalleryActivity`。（稍后介绍`SearchView`以及Honeycomb搜索时，我们会解释具体原因。）这一次我们指定了singleTop启动模式。这意味

着接收到的搜索intent将发送给回退栈顶部处于运行状态的PhotoGalleryActivity，而不是启动一个新的activity。

通过覆盖Activity中的onNewIntent(Intent)方法，可接收到新的intent。无论该intent是何时接收到的，都需在PhotoGalleryFragment中刷新显示图片项。

重构PhotoGalleryFragment，添加一个updateItems()方法，用来运行FetchItemsTask以刷新当前图片项，如代码清单28-8所示。

代码清单28-8 添加更新方法（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenul(true);

    ~~new FetchItemsTask().execute();~~
    updateItems();

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    mThumbnailThread.setListener(new ThumbnailDownloader.Listener<ImageView>() {
        ...
    });
    mThumbnailThread.start();
    mThumbnailThread.getLooper();
}

public void updateItems() {
    new FetchItemsTask().execute();
}
```

然后在PhotoGalleryActivity中覆盖onNewIntent(Intent)方法，以接收新的intent并刷新PhotoGalleryFragment展示的图片项，如代码清单28-9所示。

代码清单28-9 覆盖onNewIntent(...)方法（PhotoGalleryActivity.java）

```
public class PhotoGalleryActivity extends SingleFragmentActivity {
    private static final String TAG = "PhotoGalleryActivity";

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }
    @Override
    public void onNewIntent(Intent intent) {
        PhotoGalleryFragment fragment = (PhotoGalleryFragment)
            getSupportFragmentManager().findFragmentById(R.id.fragmentContainer);

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUERY);
            Log.i(TAG, "Received a new search query: " + query);
        }

        fragment.updateItems();
    }
}
```

运行搜索时，在LogCat日志窗口可看到PhotoGalleryActivity接收到了新的intent，同时PhotoGallery应用首先重新显示了Brian的大头照，随后刷新显示了查询结果。

关于onNewIntent(Intent)方法的使用应重点注意：如果需要新intent的值，切记将其储存起来。从getIntent()方法获取的是旧intent的值。这是因为getIntent()方法只返回**启动**了当前activity的intent，而非接收到的最新intent。

接下来的任务是将搜索查询整合到应用中，并控制在任何时候一次只触发一个搜索查询。另外，如果查询信息能够持久化保存那就更完美了。

28.2.6　使用shared preferences实现轻量级数据存储

我们可采用序列化对象并保存至外部存储设备的方式（参见第17章），实现持久化保留搜索查询信息。然而，对于轻量级数据的持久化，使用shared preferences会更加简单高效。

shared preferences本质上就是文件系统中的文件，可使用SharedPreferences类对其进行读写。SharedPreferences实例使用起来更像一个键值对仓库（类似于Bundle），但它可以通过持久化存储保存数据。键值对中的键为字符串，而值是原子数据类型。进一步查看shared preferences文件可知，它们实际上是一种简单的XML文件，但SharedPreferences类已屏蔽了读写文件的实现细节。

为在shared preferences中保存并使用简单的字符串值，只需添加一个常量作为键值对中的键来对应要存储的首选项值即可。在FlickrFetchr类中，添加所需常量，如代码清单28-10所示。

代码清单28-10　shared preferences常量（FlickrFetchr.java）

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    public static final String PREF_SEARCH_QUERY = "searchQuery";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    ...
```

要获得特定的SharedPreferences实例，可使用Context.getSharedPreferences(String,int)方法。然而，在实际开发中，由于shared preferences共享于整个应用，我们通常并不太关心获取的特定实例是什么。这种情况下，我们最好使用PreferenceManager.getDefaultSharedPreferences(Context)方法，该方法会返回具有私有权限与默认名称的实例。

在PhotoGalleryActivity类中，取得默认的SharedPreferences实例，将查询字符串信息保存到shared preferences中，如代码清单28-11所示。

代码清单28-11　存储查询信息（PhotoGalleryActivity.java）

```
@Override
public void onNewIntent(Intent intent) {
    PhotoGalleryFragment fragment = (PhotoGalleryFragment)getSupportFragmentManager()
            .findFragmentById(R.id.fragmentContainer);

    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
```

```
            String query = intent.getStringExtra(SearchManager.QUERY);
            Log.i(TAG, "Received a new search query: " + query);

            PreferenceManager.getDefaultSharedPreferences(this)
                .edit()
                .putString(FlickrFetchr.PREF_SEARCH_QUERY, query)
                .commit();
        }
        fragment.updateItems();
    }
```

以上代码中，通过调用SharedPreferences.edit()方法，可获取一个SharedPreferences.Editor实例。它就是我们在SharedPreferences中保存查询信息要使用的类。与FragmentTransaction的使用类似，利用SharedPreferences.Editor，我们可将一组数据操作放入一个事务中。如有一批数据更新操作，我们可在一个事务中批量进行数据存储写入操作。

完成所有数据的变更准备后，调用SharedPreferences.Editor的commit()方法最终写入数据。这样，该SharedPreferences文件的其他用户即可看到写入的数据。

相比数据的写入，取回存储的数据要简单的多，我们只需调用对应写入数据类型的方法即可，如SharedPreferences.getString(...)方法或者SharedPreferences.getInt(...)方法等。在PhotoGalleryFragment类中，添加代码清单28-12所示代码，从默认的SharedPreferences中取回搜索查询信息。

代码清单28-12 取回搜索查询信息（PhotoGalleryFragment.java）

```
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
        String query = "android"; // just for testing
        Activity activity = getActivity();
        if (activity == null)
            return new ArrayList<GalleryItem>();

        String query = PreferenceManager.getDefaultSharedPreferences(activity)
            .getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
        if (query != null) {
            return new FlickrFetchr().search(query);
        } else {
            return new FlickrFetchr().fetchItems();
        }
    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> items) {
        ...
    }
}
```

Preferences是PhotoGallery应用的数据存储引擎。（相比JSON的序列化，显然这种数据持久化方式要容易的多。）

搜索功能现在应该可以正常工作了。运行PhotoGallery应用，尝试一些搜索并查看返回结果。

最后，为实现搜索的撤销，添加代码清单28-13所示代码，从shared preferences中清除搜索信息

并再次调用updateItems()方法：

代码清单28-13　搜索的撤销（PhotoGalleryFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_clear:
            PreferenceManager.getDefaultSharedPreferences(getActivity())
                .edit()
                .putString(FlickrFetchr.PREF_SEARCH_QUERY, null)
                .commit();
            updateItems();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

28.3　在 Android 3.0 以后版本的设备上使用 SearchView

当前搜索界面在各场景下都运作良好。然而，它并不适用于Honeycomb及以后的系统。

Honeycomb新增加了一个SearchView类。SearchView属于操作视图（action view），可内置在操作栏里。因此，不像对话框那样叠置在activity上，SearchView支持将搜索界面内嵌在activity的操作栏里。这意味着SearchView的搜索界面使用了与应用完全一致的样式与主题，这显然再好不过了。

要使用操作视图，只需在菜单项标签中添加一个android:actionViewClass属性即可，如代码清单28-14所示。

代码清单28-14　在菜单中添加操作视图（res/menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    android:actionViewClass="android.widget.SearchView"
    />
  <item android:id="@+id/menu_item_clear"
    ...
    />
</menu>
```

在菜单XML中指定操作视图，相当于告诉Android“不要在操作栏对此菜单项使用常规视图部件，请使用指定的视图类。”通常，这也意味着连带获得了其他不同的行为。一个典型的例子是：SearchView不会产生任何onOptionsItemSelected(...)回调方法。这反而带来了方便，因为这意味着可将这些回调方法预留给那些不支持操作视图的老设备使用。

（说到老设备，我们可能会想到支持库中的SearchViewCompat类。不过仅从类名猜功能已经不

准了，SearchViewCompat类并不是用于老设备的SearchView类的兼容类。相反，它包含了一些静态方法，用于方便地在需要的地方有选择地插入SearchView。因此，它解决不了这里的问题。）

如果需要，可运行PhotoGallery应用，看看SearchView的显示效果。不过，当前它还无法正常工作。在发送搜索intent之前，SearchView还需知道当前的搜索配置信息。在onCreateOptionsMenu(...)方法中添加相应代码，取得搜索配置信息并发送给SearchView。

在系统服务SearchManager的帮助下，实际的代码实现并没有想象中的困难。SearchManager的getSearchableInfo(ComponentName)方法可查找manifest配置，打包相关信息并返回SearchableInfo对象。然后，只需将SearchableInfo对象转发给SearchView实例即可。参照代码清单28-15，添加具体实现代码。

代码清单28-15 配置SearchView（PhotoGalleryFragment.java）

```
@Override
@TargetApi(11)
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // Pull out the SearchView
        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        SearchView searchView = (SearchView)searchItem.getActionView();

        // Get the data from our searchable.xml as a SearchableInfo
        SearchManager searchManager = (SearchManager)getActivity()
            .getSystemService(Context.SEARCH_SERVICE);
        ComponentName name = getActivity().getComponentName();
        SearchableInfo searchInfo = searchManager.getSearchableInfo(name);

        searchView.setSearchableInfo(searchInfo);
    }
}
```

首先的任务是找到SearchView。可通过搜索MenuItem的资源ID找到它，然后调用getActionView()方法获得它的操作视图。

接下来，通过SearchManager取得搜索配置信息。SearchManager是负责处理所有搜索相关事宜的系统服务。前面实现搜索对话框时，正是SearchManager在幕后承担了获取搜索配置信息并显示搜索界面的工作。有关搜索的全部信息，包括应该接收intent的activity名称以及所有searchable.xml中的信息，都存储在了SearchableInfo对象中。调用getSearchableInfo(ComponentName)方法可获得该对象。

取得SearchableInfo对象后，调用setSearchableInfo(SearchableInfo)方法将相关信息通知给SearchView。现在，SearchView已实现完成。运行应用，在Android 3.0以后版本的设备上使用搜索功能，确认SearchView能够正常工作，如图28-5所示。

SearchView配置正确后，使用起来基本与前面的搜索对话框并无二致。

然而，它还存在一个小问题：如果在模拟器上尝试使用物理键盘，可看到搜索连续执行了两次。

图28-5　activity中的搜索界面

这是SearchView的一个bug。如点击物理键盘上的回车键，会触发搜索intent两次。在默认启动模式的activity中，这将启动两个同样的activity去处理同一搜索请求。

前面我们设置的singleTop启动模式已规避了该问题。因而，我们可确保intent首先发送给当前存在的activity，这样即使发送了重复的搜索intent，也不会有新的activity启动。虽然，重复的intent依然会导致搜索连续运行两次，但这远比启动两个同样的activity来处理同一搜索请求要好得多。

28.4　挑战练习

本章的挑战练习难度不大。第一个练习与Activity.startSearch(...)方法的使用有关。

后台实现时，我们是通过onSearchRequested()方法间接调用Activity.startSearch(...)方法启动搜索对话框的。后者有更多的方式和选项配置启动搜索对话框。例如，可在用户输入搜索的EditText中指定初始查询字符串，也可在intent extra中添加额外的Bundle数据信息并发送给可搜索activity，或请求一个全局网络搜索对话框（类似在主屏点按搜索按钮弹出的对话框）。

作为第一个练习，使用Activity.startSearch(...)方法，在搜索对话框中，默认加亮显示当前搜索查询字符串。

作为第二个练习，在提交新的搜索查询后，使用Toast消息提示返回的查询结果总数。小提示：可查看Flickr返回XML文件的一级元素属性，获知返回的搜索结果数目。