第 26 章

HTTP与后台任务

26

信息时代，互联网应用占用了用户的大量时间。餐桌上无人交谈，每个人都只顾低头摆弄手机。一有时间，人们就上网检查新闻推送、收发短信息，或是玩着联网游戏。

为着手学习Android网络应用的开发，我们将创建一个名为PhotoGallery的应用。PhotoGallery是图片共享网站Flickr的客户端应用。它将获取并展示上传至Flickr网站的最新公共图片。应用的运行效果如图26-1所示。

图26-1　PhotoGallery应用运行效果图

（图26-1中的图片是我们自己准备的图片，而非Flickr上的公共图片。Flickr网站上的图片归上传者私人所有，未经本人许可，任何人不得使用。可访问网址http://pressroom.yahoo.net/pr/ycorp/permissions.aspx，了解更多有关Flickr上第三方内容的使用权限问题。）

接下来的六章我们将学习开发PhotoGallery应用。前两章将介绍有关网络下载、XML文件解析、图像显示等基本知识。随后的几章里，通过各种特色功能的添加，将介绍有关搜索、服务、通知、广播接收器以及网页视图等知识。

本章，我们首先学习Android高级别的HTTP网络编程。当前，几乎所有网络服务的开发都是以HTTP网络协议为基础的。至本章结束时，我们要完成的任务是：获取、解析以及显示Flickr上图片的文字说明。（第27章会介绍图片获取与显示的相关内容。）

图26-2 本章完成的PhotoGallery应用效果图

26.1 创建 PhotoGallery 应用

按照图26-3所示的配置，创建一个全新的Android应用项目。

单击Next按钮，通过应用向导创建一个名为`PhotoGalleryActivity`的空白activity。

PhotoGallery应用将继续沿用前面一直使用的设计架构。PhotoGalleryActivity同样设计为SingleFragmentActivity的子类，其视图为activity_fragment.xml中定义的容器视图。PhotoGalleryActivity负责托管PhotoGalleryFragment实例。稍后我们会创建它。

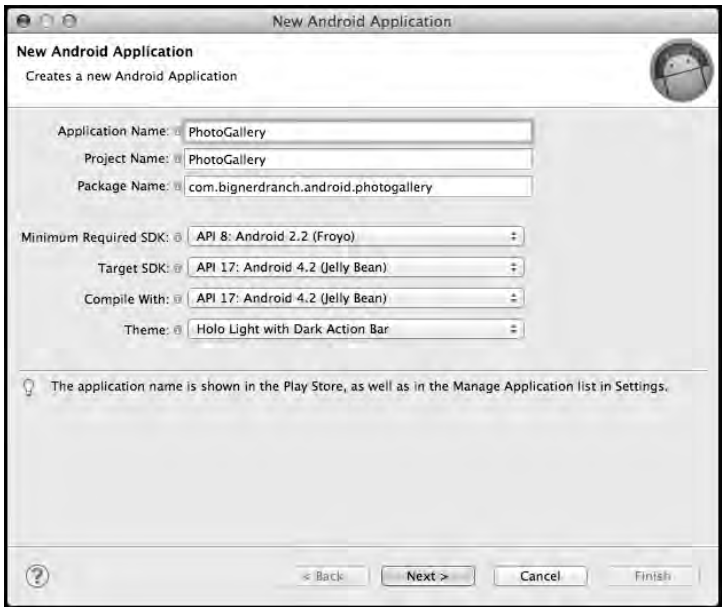

图26-3　创建PhotoGallery应用

将SingleFragmentActivity.java和activity_fragment.xml从以前的项目复制到当前项目中。

在PhotoGalleryActivity.java中，删除自动产生的模板代码。然后设置PhotoGalleryActivity的父类为SingleFragmentActivity并实现它的createFragment()方法。createFragment()方法将返回一个PhotoGalleryFragment类实例。如代码清单26-1所示。（暂时无需理会代码的错误提示，它会在PhotoGalleryFragment类创建完成后自动消失。）

代码清单26-1　activity的调整（PhotoGalleryActivity.java）

```
~~public class PhotoGalleryActivity extends Activity {~~
public class PhotoGalleryActivity extends SingleFragmentActivity {

    ~~/* Auto-generated template code */~~

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }
}
```

PhotoGallery应用将在GridView视图中显示获取的内容。而GridView由PhotoGalleryFragment的视图组成。

按照继承关系，GridView也是一个AdapterView，因此其工作方式与ListView类似。然而，不像ListView，GridView没有内置方便实用的GridFragment。这意味着我们需自己创建布局文件，并在PhotoGalleryFragment类中进行实例化。在本章的后面，我们将在PhotoGalleryFragment类中使用adapter提供图片说明文字给GridView视图显示。

为创建fragment布局，重命名layout/activity_photo_gallery.xml为layout/fragment_photo_gallery.xml。然后以图26-4所示的GridView替换原有布局内容。

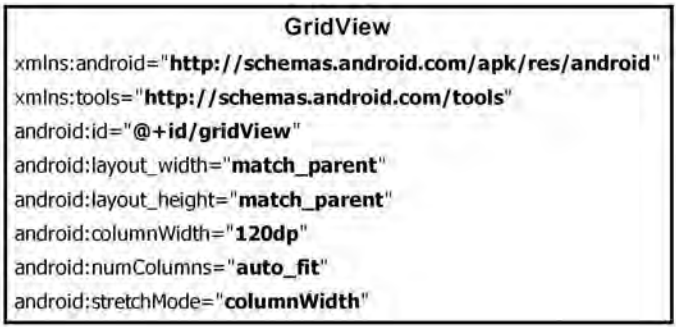

图26-4 GridView视图（layout/fragment_photo_gallery.xml）

这里，我们设置列的宽度为120dp，并使用numColumns属性指示GridView创建尽可能多的列，以铺满整个屏幕。如果在列的空间分配上出现少于120dp的剩余空间，则stretchMode属性会要求GridView在全部列间均分这部分剩余空间。

最后，创建PhotoGalleryFragment类，设置其为保留fragment，实例化生成新建布局并引用GridView视图，如代码清单26-2所示。

代码清单26-2 一些代码片断（PhotoGalleryFragment.java）

```
package com.bignerdranch.android.photogallery;

...

public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

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

        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        return v;
    }
}
```

在继续学习之前，尝试运行PhotoGallery应用。如果一切正常，我们会看到一个空白视图。

26.2　网络连接基本

PhotoGallery应用中，我们需要一个处理网络连接的专用类。新建一个Java类。应用要访问连接的是Flickr网站，因此命名新建类为FlickrFetchr。

FlickrFetchr类一开始只有getUrlBytes(String)和getUrl(String)两个方法。getUrlBytes(String)方法从指定URL获取原始数据并返回一个字节流数组。getUrl(String)方法则将getUrlBytes(String)方法返回的结果转换为String。

在FlickrFetchr.java中，为getUrlBytes(String)和getUrl(String)方法添加实现代码，如代码清单26-3所示。

代码清单26-3　基本网络连接代码（FlickrFetchr.java）

```
package com.bignerdranch.android.photogallery;

...

public class FlickrFetchr {
    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return null;
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }

    public String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

在getUrlBytes(String)方法中，根据传入的字符串参数，如http://www.google.com，首先创建一个URL对象。然后调用openConnection()方法创建一个指向要访问URL的连接对象。URL.openConnection()方法默认返回的是URLConnection对象，但我们要连接的是http URL，因此

需将其强制类型转换为HttpURLConnection对象。随后，我们得以调用它的getInputStream()、getResponseCode()等方法。

HttpURLConnection对象虽然提供了一个连接，但只有在调用getInputStream()方法时（如果是POST请求，则调用getOutputStream()方法），它才会真正连接到指定的URL地址。在此之前我们无法获得有效的返回代码。

一旦创建了URL并打开了网络连接，我们便可循环调用read()方法读取网络连接到的数据，直到取完为止。只要还有数据存在，InputStream类便可不断输出字节流数据。数据全部输出后，关闭网络连接，并将读取的数据写入ByteArrayOutputStream字节数组中。

虽然getUrlBytes(String)方法完成了最重要的数据获取任务，但getUrl(String)才是本章真正需要的方法。它负责将getUrlBytes(String)方法获取的字节数据转换为String。至此，是不是想问，难道不可以在一个方法中完成全部任务？当然可以，但是处理下一章的图像数据下载时，我们需要使用两个独立的方法。

获取网络使用权限

要连接使用网络，还需完成一件事：取得使用网络的权限。正如用户不愿被偷拍照片一样，他们也不想有人偷偷下载他们的图片。

要取得网络使用权限，参照代码清单26-4，添加以下代码到AndroidManifest.xml文件中。

代码清单26-4 在配置文件中添加网络使用权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET" />

  ...

</manifest>
```

26.3 使用 AsyncTask 在后台线程上运行代码

接下来是调用并测试新添加的网络连接代码。注意，不能直接在PhotoGalleryFragment类中调用FlickrFetchr.getURL(String)方法，我们应创建一个后台线程，然后在该线程中运行代码。

使用后台线程最简便的方式是使用AsyncTask工具类。AsyncTask创建后台线程后，我们便可在该线程上调用doInBackground(...)方法运行代码。

在PhotoGalleryFragment.java中，添加一个名为FetchItemsTask的内部类。覆盖AsyncTask.

doInBackground(...)方法，从目标网站获取数据并记录下日志，如代码清单26-5所示。

代码清单26-5　实现AsyncTask工具类方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;

    ...

    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr().getUrl("http://www.google.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

然后，在PhotoGalleryFragment.onCreate(...)方法中，调用FetchItemsTask新实例的execute()方法，如代码清单26-6所示。

代码清单26-6　实现AsyncTask工具类方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

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

        setRetainInstance(true);
        new FetchItemsTask().execute();
    }

    ...

}
```

调用execute()方法将启动AsyncTask，继而触发后台线程并调用doInBackground(...)方法。运行PhotoGallery应用，查看LogCat窗口，可看到混合了大量Javascript的Google HTML主页源代码，如图26-5所示。

既然已创建了后台线程，并成功完成了网络连接代码的测试，接下来，我们来深入学习Android线程的知识。

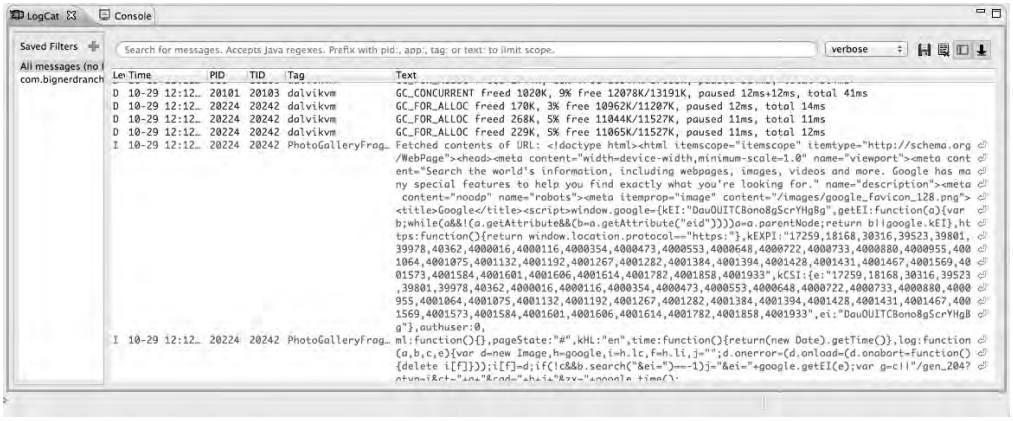

图26-5 LogCat中的Google HTML代码

26.4 线程与主线程

网络通常无法立即联通。网络服务器可能需要1~2秒的时间来响应访问请求，文件下载则耗时更久。鉴于网络连接的耗时，自Honeycomb系统版本开始，Android禁止在主线程中发生任何网络连接行为。即使强行为之，Android也会抛出`NetworkOnMainThreadException`的异常。这是为什么呢？要知道其中原因，首先需了解什么是线程，什么是主线程以及主线程的用途是什么。

线程是一个单一的执行序列。单个线程中的代码可得到逐步执行。每个Android应用的运行都是从主线程开始的。然而，主线程并非如线程般的预定执行序列，如图26-6所示。相反，它处于一个无限循环的运行状态，等待着用户或系统触发事件的发生。事件触发后，主线程便负责执行代码，以响应这些事件。

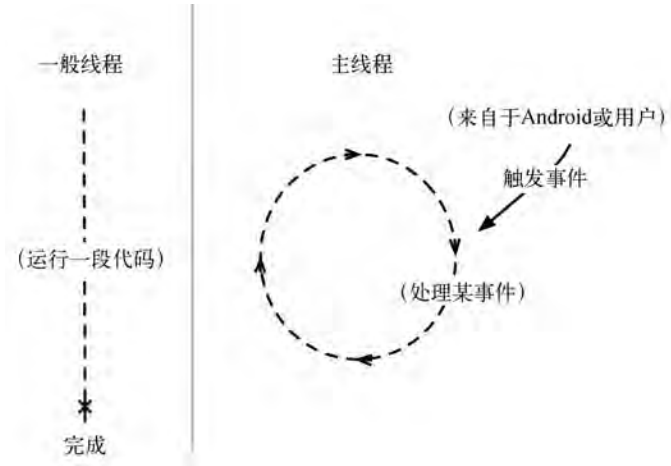

图26-6 一般线程与主线程

想象一下，应用就是一家大型鞋店，而闪电侠是唯一的员工。（是不是人人梦寐以求的场景？）要让客户满意，鞋店里有大量工作需要他去处理，如摆放布置商品、为顾客取鞋、用量脚器为顾客量尺寸等等。作为售货员，即使所有工作都由闪电侠一人完成，他也能够快速响应，兼顾每一位顾客的需求。

为保证完成任务，闪电侠不能在单一事件上耗时过久。要是一批货丢了怎么办？这时，必须得有人打电话四处联络并设法找到丢失的货物才行。假设让闪电侠处理这项耗时的任务，他在忙于联络货物时，店里的顾客可能会等得有些不耐烦。

闪电侠就如同应用里的主线程。它运行着所有用于更新UI的代码，其中包括响应activity的启动、按钮的点击等不同UI相关事件的代码。（由于响应的事件基本都与用户界面相关，主线程有时也叫做UI线程。）

事件处理的循环使得UI相关代码得以顺序执行。这足以保证任何事件处理都不会发生冲突，同时代码也能够快速响应执行。目前为止，我们编写的所有代码（刚刚使用`AsyncTask`工具类完成的代码除外）都是在主线程中执行的。

主线程之外

网络连接如同致电经营鞋类业务的分销商：相比其他任务，网络连接比较耗时。等待响应的时候，用户界面将会毫无反应，这可能会导致*应用无响应*（ANR：Application Not Responding）。

如果Android系统监控服务确认主线程无法响应重要事件，如按下后退键等，则应用无响应会发生。用户会看到如图26-7所示的画面。

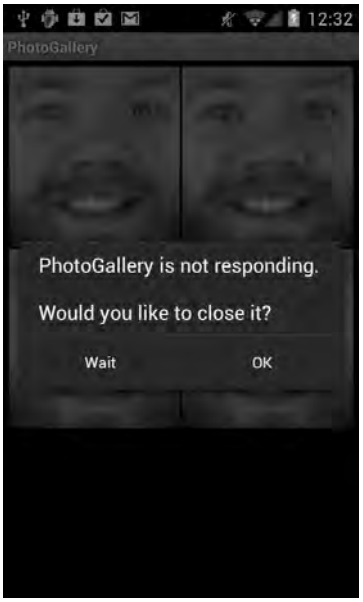

图26-7　应用无响应

这也就是为什么自Honeycomb系统开始，Android禁止在主线程中发生任何网络连接行为的原因所在。

回到假想的鞋店中，要想解决问题，（自然）需要再雇佣一名闪电侠专门负责与供销商之间的联络工作。Android系统中所做的操作与之差不多，即创建一个后台线程，然后通过该线程访问网络。

怎样利用后台线程才最容易呢？答案是：非AsyncTask工具类莫属。

本章的后面，我们会使用AsyncTask类处理一些其他任务。现在，我们还是先使用网络访问代码做点实际工作。

26.5 获取 Flickr XML 数据

Flickr提供了方便而强大的XML API。可从http://www.flickr.com/services/api/文档页查看使用细节。在常用浏览器中打开API文档网页，找到Request Formats列表。我们准备使用最简单的REST响应格式。查看文档得知，它的API端点（endpoint）是http://api.flickr.com/services/rest/。我们可在此端点上调用Flickr提供的方法。

回到API文档主页，找到API Methods列表。向下滚动到photos区域并定位flickr.photos.getRecent方法。点击查看该方法，可以看到，文档对该方法的描述为："返回最近上传到flickr的公共图片列表清单。"这恰好就是PhotoGallery应用所需的方法。

getRecent方法唯一需要的参数是一个API key。为获得该参数，返回http://www.flickr.com/services/api/文档主页，找到并点击API keys链接进行申请。申请需使用Yahoo ID进行登录。登录成功后，申请一个全新的非商业用途APIkey。成功提交申请后，可获得类似4f721bgafa75bf6d-2cb9af54f937bb70的API key。

获取API key后，可直接向Flickr网络服务发起一个GET请求，即http://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=xxx。

接下来开始编码工作。首先，在FlickrFetchr类中添加一些常量，如代码清单26-7所示。

代码清单26-7 添加一些常量（FlickrFetchr.java）

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

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "yourApiKeyHere";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";

    private static final String EXTRA_SMALL_URL = "url_s";
```

这些常量定义了访问端点、方法名、API key以及一个值为url_s的extra参数。指定url_s值的extra参数，实际是告诉Flickr：如有小尺寸图片，也请一并将其URL包括在内并返回。

使用刚才定义的常量，编写一个方法，构建适当的请求URL并获取所需内容，如代码清单26-8所示。

代码清单26-8　添加fetchItems()方法（FlickrFetchr.java）

```
public class FlickrFetchr {

    ...

    String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        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);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

这里我们使用Uri.Builder构建完整的Flickr API请求URL。便利类Uri.Builder可创建正确转义的参数化URL。Uri.Builder.appendQueryParameter(String,String)可自动转义查询字符串。

最后，修改PhotoGalleryFragment类中的AsyncTask内部类，调用新的fetchItems()方法，如代码清单26-9所示。

代码清单26-9　调用fetchItems()方法（PhotoGalleryFragment.java）

```
private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            String result = new FlickrFetchr().getUrl("http://www.google.com");
            Log.i(TAG, "Fetched contents of URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch URL: ", ioe);
        }
        new FlickrFetchr().fetchItems();
        return null;
    }
}
```

运行PhotoGallery应用。可看到LogCat窗口中出现的大量的Flickr XML，如图26-8所示。

成功获取Flickr XML返回结果后，该如何使用它呢？我们可按需处理这些数据，也即将其存入一个或多个模型对象中。我们待会要为PhotoGallery应用创建的模型类名为GalleryItem。图26-9为PhotoGallery应用的对象图解。

注意，为重点显示fragment和网络连接代码，图26-9并没有显示托管activity。

创建GalleryItem类并添加代码清单26-10所示代码。

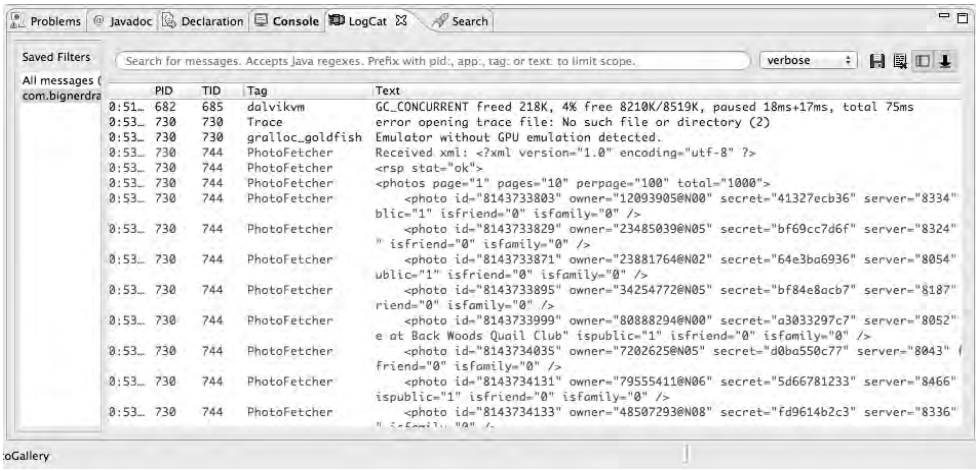

图26-8 Flickr XML

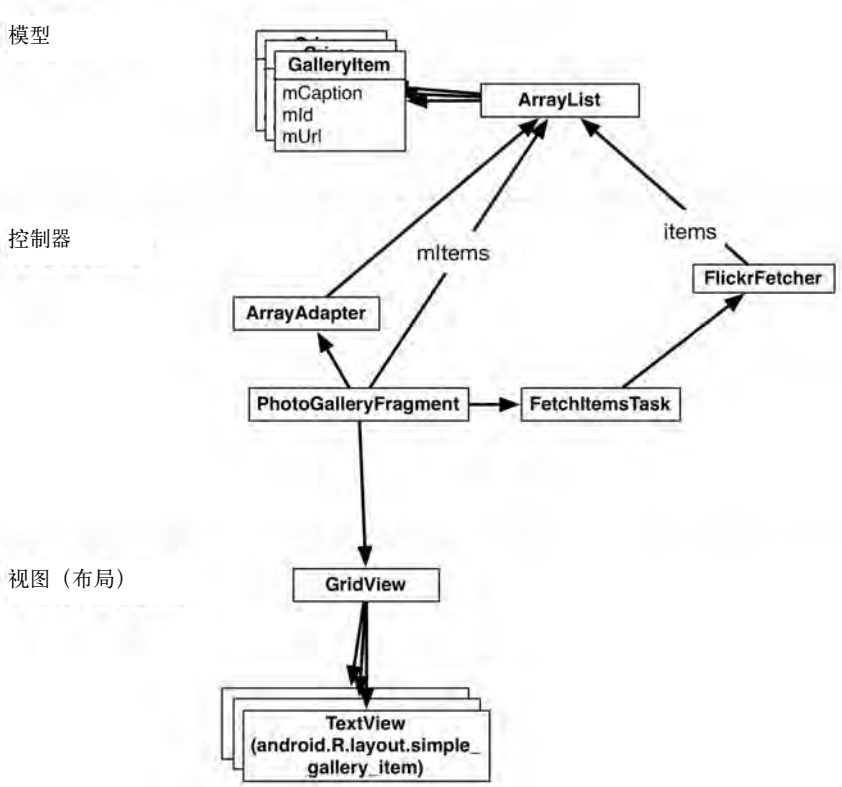

图26-9 PhotoGallery应用的对象图解

代码清单26-10　创建模型对象类（GalleryItem.java）

```
package com.bignerdranch.android.photogallery;

public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    public String toString() {
        return mCaption;
    }
}
```

利用Eclipse自动为mId，mCaption以及mUrl变量生成getter与setter方法。

完成模型层对象的创建后，接下来的任务就是使用从Flickr XML中解析的数据对其进行填充。要从XML中获取数据，需使用XmlPullParser接口。

使用XmlPullParser

XmlPullParser接口采用拉的方式从XML数据流中获取解析事件。Android内部也使用XmlPullParser接口来实例化布局文件。

在FlickrFetchr类中，添加一个指定图片XML元素名称的常量。然后再添加一个parseItems(...)方法，使用XmlPullParser解析XML中的全部图片。每张图片会产生一个GalleryItem对象，并被添加到ArrayList中。如代码清单26-11所示。

代码清单26-11　解析Flickr图片（FlickrFetchr.java）

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

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "your API key";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";

    private static final String XML_PHOTO = "photo";

    ...

    public void fetchItems() {
        ...
    }

    void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
            throws XmlPullParserException, IOException {
        int eventType = parser.next();

        while (eventType != XmlPullParser.END_DOCUMENT) {
            if (eventType == XmlPullParser.START_TAG &&
                XML_PHOTO.equals(parser.getName())) {
                String id = parser.getAttributeValue(null, "id");
                String caption = parser.getAttributeValue(null, "title");
                String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);
```

```
                GalleryItem item = new GalleryItem();
                item.setId(id);
                item.setCaption(caption);
                item.setUrl(smallUrl);
                items.add(item);
            }

            eventType = parser.next();
        }
    }
}
```

想象XmlPullParser有根手指放在XML文档上，它会逐步处理START_TAG，END_TAG和END_DOCUMENT等不同的XML节点事件。每一步，在XmlPullParser当前指向的事件上，都可调用getText()、getName()以及getAttributeValue(...)等方法，来获取我们需要的当前节点事件的任何信息。要继续指向下一个XML节点事件，我们可调用XmlPullParser的next()方法。方便的是，该方法还会返回当前指向的事件类型。XmlPullParser的工作原理如图26-10所示。

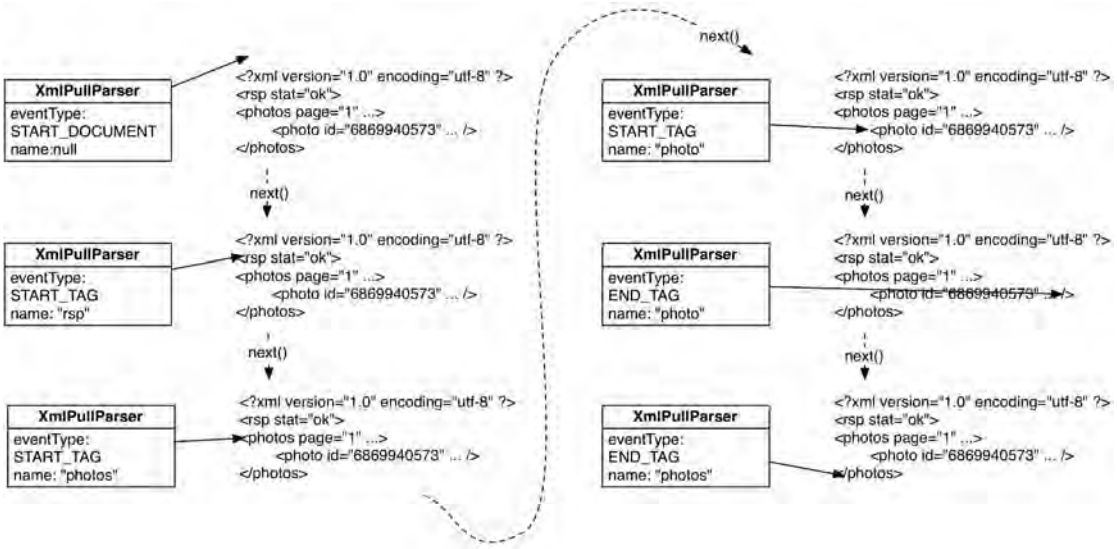

图26-10 XmlPullParser的工作原理

parseItems(...)方法需要一个XmlPullParser参数和一个ArrayList参数。因此我们创建一个parser实例，并输入Flickr返回的xmlString。然后，调用parseItems(...)方法并传入准备就绪的parser以及空数组列表参数，如代码清单26-12所示。

代码清单26-12 调用parseItems(...)方法（FlickrFetchr.java）

```
~~public void fetchItems() {~~
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;
    }
```

运行PhotoGallery应用，测试XML解析代码。现在，PhotoGallery应用还无法展示ArrayList中的内容。因此，要确认代码是否工作正常，需设置合适的断点，并使用调试器来检查代码逻辑。

26.6　从 AsyncTask 回到主线程

为完成本章的既定目标，我们回到视图层部分，实现在PhotoGalleryFragment类的GridView中显示图片的文字说明。

同ListView一样，GridView也是一个AdapterView，因此它需要adapter协助配置视图中要显示的内容。

在PhotoGalleryFragment.java中，添加一个GalleryItem类型的ArrayList变量，并使用Android提供的简单布局为GridView视图设置一个ArrayAdapter，如代码清单26-13所示。

代码清单26-13　实现setupAdapter()方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList<GalleryItem> mItems;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        setupAdapter();

        return v;
    }
```

```
void setupAdapter() {
    if (getActivity() == null || mGridView == null) return;

    if (mItems != null) {
        mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
                android.R.layout.simple_gallery_item, mItems));
    } else {
        mGridView.setAdapter(null);
    }
}
```

GridView无方便的GridFragment类可用，所以我们必须自己编码实现adapter的管理。这里，我们的实现方式是使用setupAdapter()方法。该方法根据当前模型数据的不同状态，可对应设置GridView的adapter。我们应在onCreateView(...)方法中调用该方法，这样每次因设备旋转重新生成GridView视图时，可重新为其配置对应的adapter。另外，每次模型层对象发生改变时，也应保证该方法的及时调用。

布局android.R.layout.simple_gallery_item由TextView组成。前面在创建GalleryItem类时，我们已覆盖toString()方法返回对象的mCaption。因此，要在GridView显示图片的文字说明，只需传入GalleryItem数组列表以及android.R.layout.simple_gall-ery_item布局给adapter即可。

注意，设置adapter前，应检查getActivity()的返回结果是否为空。这是因为fragment可脱离activity而存在。这之前没出现这种情况是因为所有的方法调用都是由系统框架的回调方法驱动的。如果fragment收到回调指令，则它必然关联着某个activity；如它脱离activity而存在，则会收不到回调指令。

既然正在使用AsyncTask，我们必须自己负责触发相应的事件，而且也不能确定fragment是否与activity相关联。因此需检查确认fragment是否仍与activity相关联。如果fragment脱离了activity，则依赖于activity的操作（如创建ArrayAdapter）就会失败。

从Flickr成功获取数据后，需在合适的地方调用setupAdapter()方法。我们的第一反应可能是在FetchItemsTask的doInBackground(...)方法尾部调用setupAdapter()方法。这不是个好主意。再次回到闪电侠与鞋店的假想场景。现在，我们有两个闪电侠在店里忙碌，一个忙于应付大量顾客，一个忙于与Flickr电话沟通。如果第二个闪电侠结束通话后，过来帮忙招呼店里的顾客，会发生什么情况呢？结局很可能是两位闪电侠无法协调一致，产生冲突，结果是帮了倒忙。

而在计算机里，这种内存对象间步调不一致的冲突会导致应用的崩溃。因此，为避免安全隐患，不推荐也不允许在后台线程中更新UI。

那么应该怎么做呢？不用担心，AsyncTask提供有另一个可覆盖的onPostExecute(...)方法。onPostExecute(...)方法在doInBackground(...)方法执行完毕后才会运行，而且它是在主线程而非后台线程上运行的。因此，在该方法中更新UI比较安全。

修改FetchItemsTask类以新的方式更新mItems，并在成功获取图片后调用setupAdapter()方法，如代码清单26-14所示。

代码清单26-14 添加adapter更新代码（PhotoGalleryFragment.java）

```
~~private class FetchItemsTask extends AsyncTask<Void,Void,Void> {~~
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
```

```
        protected Void doInBackground(Void... params) {
        protected ArrayList<GalleryItem> doInBackground(Void... params) {
            new FlickrFetchr().fetchItems();
            return new FlickrFetchr().fetchItems();
            return null;
        }

        @Override
        protected void onPostExecute(ArrayList<GalleryItem> items) {
            mItems = items;
            setupAdapter();
        }
    }
```

这里总共做了三处调整。首先，我们改变了FetchItemsTask类第三个泛型参数的类型。该参数是AsyncTask返回结果的数据类型。它设置了doInBackground(...)方法返回结果的类型，以及onPostExecute(...)方法输入参数的数据类型。

其次，我们让doInBackground(...)方法返回了GalleryItem类型的数据列表。这样，既修正了编译代码的错误。同时，还将数组列表数据传递给onPostExecute(...)方法。

最后，我们添加了onPostExecute(...)方法的实现代码。该方法接收从doInBackground(...)方法获取的列表数据，并将返回数据放入mItems变量，然后调用setupAdapter()方法更新GridView视图的adapter。

至此，本章的任务就完成了。运行PhotoGallery应用，可看到屏幕上显示了下载的全部GalleryItem的文字说明，如图26-11所示。

图26-11　来自Flickr的图片文字说明

26.7 深入学习：再探 AsyncTask

本章我们已知道如何使用AsyncTask的第三个类型参数。那另外两个类型参数又该如何使用呢?

第一个类型参数可指定输入参数的类型。可参考以下示例使用该参数：

```
AsyncTask<String,Void,Void> task = new AsyncTask<String,Void,Void>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }

        return null;
    }
};

task.execute("First parameter", "Second parameter", "Etc.");
```

输入参数传入execute(...)方法（可接受一个或多个参数）。然后，这些变量参数再传递给doInBackground(...)方法。

第二个类型参数可指定发送进度更新需要的类型。以下为示例代码：

```
final ProgressBar progressBar = /* A determinate progress bar */;
progressBar.setMax(100);

AsyncTask<Integer,Integer,Void> task = new AsyncTask<Integer,Integer,Void>() {
    public Void doInBackground(Integer... params) {
        for (Integer progress : params) {
            publishProgress(progress);
            Thread.sleep(1000);
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        progressBar.setProgress(progress);
    }
};

task.execute(25, 50, 75, 100);
```

进度更新通常发生在执行的后台进程中。在后台进程中，我们无法完成必要的UI更新。因此AsyncTask提供了publishProgress(...)和onProgressUpdate(...)两个方法。

其工作方式如下：在后台线程中，我们从doInBackground(...)方法中调用publishProgress(...)方法。这样onProgressUpdate(...)方法便能够在UI线程上被调用。因此我们可在onProgressUpdate(...)方法中执行UI更新，但我们必须在doInBackground(...)方法中使用publishProgress(...)方法对它们进行管理控制。

清理AsyncTask

本章，AsyncTask的设计使用合理，因此我们无需对其进行跟踪管理。然而有些情况下，我

们必须对其进行掌控，甚至在需要的时候，要能够撤销或重新运行AsyncTask。

在一些复杂的使用场景下，我们需将AsyncTask赋值给实例变量。一旦能够掌控它，我们就可以随时调用AsyncTask.cancel(boolean)方法，撤销运行中的AsyncTask。

AsyncTask.cancel(boolean)方法有两种工作模式：粗暴的和温和的。如调用温和的cancel(false)方法，该方法会设置isCancelled()的状态为true。随后，AsyncTask会检查doInBackground(...)方法中的isCancelled()状态，然后选择提前结束运行。

然而，如调用粗暴的cancel(true)方法，它会直接终止doInBackground(...)方法当前所在的线程。AsyncTask.cancel(true)方法停止AsyncTask的方式简单粗暴，如果可能，应尽量避免使用此种方式。

26.8　挑战练习：分页

默认情况下，flickr.photos.getRecent方法返回每页 100个结果的一页数据。不过，该方法还有个叫做page的附加参数，我们可以使用它返回第二页、第三页等更多页数据。

本章的挑战任务是：添加代码到adapter，实现对数组列表数据的结束判断，然后使用下一页返回结果替换当前页。