第 31 章

网页浏览 31

从Flickr下载的图片都有对应关联的网页。本章将实现点击PhotoGallery应用中的图片时，能自动跳转到图片所在网页。我们将学习在应用中整合网页内容的两种方法：使用浏览器应用和使用WebView类。

31.1 最后一段 Flickr 数据

无论哪种方式，都需要取得图片所在Flickr页的URL。如果查看下载图片的XML文件，可看到图片的网页地址并不包含在内。

```
<photo id="8232706407" owner="70490293@N03" secret="9662732625"
    server="8343" farm="9" title="111_8Q1B2033" ispublic="1"
    isfriend="0" isfamily="0"
    url_s="http://farm9.staticflickr.com/8343/8232706407_9662732625_m.jpg"
    height_s="240" width_s="163" />
```

因此，我们想当然地认为需要编码获取更多XML内容才行。访问http://www.flickr.com/services/api/misc.urls.html查看Flickr官方文档的Web Page URL部分，我们知道可按以下格式创建单个图片的URL：

```
http://www.flickr.com/photos/user-id/photo-id
```

这里的photo-id即XML文件的id属性值。该值已保存在GalleryItem类的mId属性中。那么剩下的user-id呢？继续查阅Flickr文档可知，XML文件的owner属性值就是用户ID。因此，只需从XML文件解析出owner属性值，即可创建图片的完整URL：

```
http://www.flickr.com/photos/owner/id
```

在GalleryItem中添加代码清单31-1所示代码，创建图片URL。

代码清单31-1 添加创建图片URL的代码（GalleryItem.java）

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

    ...
```

```
    public void setUrl(String url) {
        mUrl = url;
    }

    public String getOwner() {
        return mOwner;
    }

    public void setOwner(String owner) {
        mOwner = owner;
    }

    public String getPhotoPageUrl() {
        return "http://www.flickr.com/photos/" + mOwner + "/" + mId;
    }

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

以上代码新建了一个mOwner属性，以及一个生成图片URL的getPhotoPageUrl()方法。

现在，修改parseItems(...)方法，从XML文件中获取owner属性，如代码清单31-2所示。

代码清单31-2　从XML中获取owner属性（FlickrFetchr.java）

```
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);
            String owner = parser.getAttributeValue(null, "owner");

            GalleryItem item = new GalleryItem();
            item.setUrl(smallUrl);
            item.setOwner(owner);
            items.add(item);
        }

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

非常简单，获取图片网页URL的任务完成了。

31.2　简单方式：使用隐式 intent

我们将使用隐式intent来访问图片URL。隐式intent可启动系统默认的浏览器，并在其中打开URL指向的网页。

首先，实现应用对GridView显示项点击事件的监听。由于没有匹配的GridFragment，此处要实现的代码与第9章稍有不同。不再采用在fragment中覆盖onListItemClick(...)方法的方式，我们转而调用GridView的setOnItemClickListener(...)方法，实现图片点击事件的监听。这与按钮点击监听器的处理类似。

完成图片点击事件的监听后，可采用一种简单的方式来实现网页的浏览，即创建并发送隐式intent。如代码清单31-3所示，在PhotoGalleryFragment类中添加以下代码：

代码清单31-3 通过隐式intent实现网页浏览（PhotoGalleryFragment.java）

```
@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();

    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
                long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);

            startActivity(i);
        }
    });

    return v;
}
```

启动PhotoGallery应用，点击任意图片。短暂的进度指示动画后，浏览器应用应该会弹出。

31.3 较难方式：使用 WebView

然而很多时候，我们需要在activity中显示网页内容，而不是打开独立的浏览器应用。我们也许想显示自己生成的HTML，或想以某种方式锁定浏览器的使用。对于大多数包含帮助文档的应用，普遍做法是以网页的形式提供帮助文档，这样会方便后期的更新与维护。打开浏览器应用查看帮助文档，既不够专业，又阻碍了对应用行为的定制，同时也无法将网页整合进自己的用户界面。

如需在自己的用户界面展现网页内容，可使用WebView类。虽然我们将使用WebView类定位成一种较难的实现方式，但实际使用并不困难。（只是相对隐式intent来说，显得困难一些而已。）

首先，创建一个activity以及一个显示WebView的fragment。依惯例先定义一个布局文件，如图31-1所示。

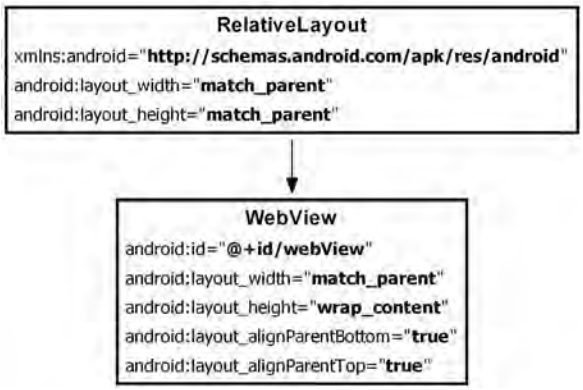

图31-1　初始布局（res/layout/fragment_photo_page.xml）

是不是会认为“这里的RelativeLayout起不了什么作用”？确实如此，本章的后面，我们会添加更多的组件来完善它。

接下来是创建初步的fragment。以上一章创建的VisibleFragment为父类，新建PhotoPageFragment类。实现布局文件的实例化，从中引用WebView，并转发从intent数据中获取的URL。如代码清单31-4所示。

代码清单31-4　创建网页浏览fragment（PhotoPageFragment.java）

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

...

public class PhotoPageFragment extends VisibleFragment {
    private String mUrl;
    private WebView mWebView;

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

        mUrl = getActivity().getIntent().getData().toString();
    }

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

        mWebView = (WebView)v.findViewById(R.id.webView);

        return v;
    }
}
```

当前，PhotoPageFragment类还未完成，暂时先这样，稍后再来完成它。接下来，使用SingleFragmentActivity新建PhotoPageActivity托管类，如代码清单31-5所示。

代码清单31-5 创建显示网页的activity（PhotoPageActivity.java）

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

...

public class PhotoPageActivity extends SingleFragmentActivity {
    @Override
    public Fragment createFragment() {
        return new PhotoPageFragment();
    }
}
```

回到PhotoGalleryFragment类中，弃用隐式intent，改调新建的activity，如代码清单31-6所示。

代码清单31-6 改调新建activity（PhotoGalleryFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...

    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
                long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);
            Intent i = new Intent(getActivity(), PhotoPageActivity.class);
            i.setData(photoPageUri);

            startActivity(i);
        }
    });

    return v;
}
```

最后，在配置文件中声明新建activity，如代码清单31-7所示。

代码清单31-7 在配置文件中声明activity（AndroidManifest.xml）

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

  ...

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".PhotoGalleryActivity"
```

```
        android:launchMode="singleTop"
        android:label="@string/title_activity_photo_gallery" >
        ...
    </activity>
    <activity
        android:name=".PhotoPageActivity" />
    <service android:name=".PollService" />
    <receiver android:name=".StartupReceiver">
        ...
    </receiver>
  </application>
</manifest>
```

运行PhotoGallery应用，点击任意图片，可看到弹出的一个空白activity。

好了，现在来处理关键部分，让fragment发挥作用。WebView要成功显示Flickr图片网页，需完成三项任务。

- 告诉WebView要打开的URL。
- 启用JavaScript。JavaScript默认是禁用的。虽然不一定总是需要启用它，但Flickr网站需要。启用JavaScript后，Android Lint会提示警告信息（担心跨网站的脚本攻击），因此需禁止Lint的警告。
- 覆盖WebViewClient类的shouldOverrideUrlLoading(WebView,String)方法，并返回false值。
- 添加如代码清单31-8所示代码。然后，我们来详细解读PhotoPageFragment类。

代码清单31-8　添加更多的实例变量（PhotoPageFragment.java）

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    mWebView = (WebView)v.findViewById(R.id.webView);

    mWebView.getSettings().setJavaScriptEnabled(true);

    mWebView.setWebViewClient(new WebViewClient() {
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            return false;
        }
    });

    mWebView.loadUrl(mUrl);

    return v;
}
```

加载URL网页必须等WebView配置完成后进行，因此这一操作最后完成。在此之前，首先调用getSettings()方法获得WebSettings实例，再调用WebSettings.setJavaScriptEnabled(true)方法，从而完成JavaScript的启用。WebSettings是修改WebView配置的三种途径之一。它还有其他一些可设置属性，如用户代理字符串和显示文字大小。

然后，是配置WebViewClient。WebViewClient是一个事件接口。通过提供自己实现的WebViewClient，可响应各种渲染事件。例如，可检测渲染器何时开始从特定URL加载图片，或决定是否需要向服务器重新提交POST请求。

WebViewClient有多个方法可供覆盖，其中大多数用不到。然而，我们必须覆盖它的shouldOverrideUrlLoading(WebView,String)默认方法。当有新的URL加载到WebView（譬如说点击某个链接），该方法会决定下一步的行动。如返回true值，意即“不要处理这个URL，我自己来。”如返回false值，意即“WebView，去加载这个URL，我不会对它做任何处理。”

如本章前面的做法，默认的实现发送了附有URL数据的隐式intent。对于图片页面来说，这是个严重的问题。Flickr首先重定向到移动版本的网址。使用默认的WebViewClient，意味着会使用用户的默认浏览器。这不是我们想要的。

解决方法很简单，只需覆盖默认的实现方法并返回false值即可。

运行PhotoGallery应用，应该可看到自己的WebView。

31.3.1 使用WebChromeClient优化WebView的显示

既然花时间初步实现自己的WebView，接下来开始我们的优化，为它添加一个标题视图和一个进度条。打开fragment_photo_page.xml，添加如图31-2所示的更新代码：

引用并显示ProgressBar和TextView视图非常简单。但要将它们与WebView关联起来，还需使用WebView: WebChromeClient的第二个回调方法。如果说WebViewClient是响应渲染事件的接口，那么WebChromeClient就是一个响应那些改变浏览器中装饰元素的事件接口。这包括JavaScript警告信息、网页图标、状态条加载，以及当前网页标题的刷新。

在onCreateView(...)方法中，编写代码实现WebChromeClient的关联使用，如代码清单31-9所示。

代码清单31-9 使用WebChromeClient（PhotoPageFragment.java）

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    final ProgressBar progressBar = (ProgressBar)v.findViewById(R.id.progressBar);
    progressBar.setMax(100); // WebChromeClient reports in range 0-100
    final TextView titleTextView = (TextView)v.findViewById(R.id.titleTextView);

    mWebView = (WebView)v.findViewById(R.id.webView);

    mWebView.getSettings().setJavaScriptEnabled(true);

    mWebView.setWebViewClient(new WebViewClient() {
        ...
    });

    mWebView.setWebChromeClient(new WebChromeClient() {
        public void onProgressChanged(WebView webView, int progress) {
            if (progress == 100) {
```

```
                progressBar.setVisibility(View.INVISIBLE);
            } else {
                progressBar.setVisibility(View.VISIBLE);
                progressBar.setProgress(progress);
            }
        }

        public void onReceivedTitle(WebView webView, String title) {
            titleTextView.setText(title);
        }
    });

    mWebView.loadUrl(mUrl);

    return v;
}
```

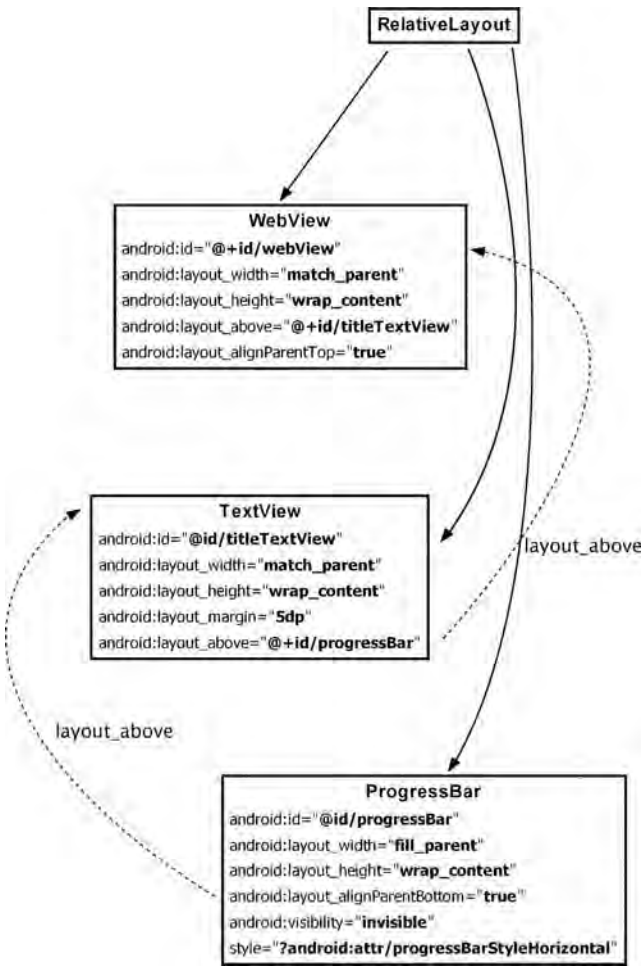

图31-2　添加标题和进度条（fragment_photo_page.xml）

进度条和标题栏的更新都有各自的回调方法，即onProgressChanged(WebView,int)和onReceivedTitle(WebView,String)方法。从onProgressChanged(WebView,int)方法收到的网页加载进度是一个从0到100的整数值。如果值是100，说明网页已完成加载，因此需设置进度条可见性为View.INVISIBLE，将ProgressBar视图隐藏起来。

运行PhotoGallery应用，测试刚才的代码更新。

31.3.2 处理WebView的设备旋转问题

尝试旋转设备屏幕。尽管应用工作如常，但WebView必须重新加载网页。这是因为WebView包含了太多的数据，以至无法在onSaveInstanceState(...)方法内保存所有数据。因此每次设备旋转，它都必须重头开始加载网页数据。

对于一些类似的类（如VideoView），Android文档推荐让activity自己处理设备配置变更。也就是说，无需销毁重建activity可直接调整自己的视图以适应新的屏幕尺寸。这样，WebView也就不必重新加载全部数据了。（干脆都这样处理好了？对不起，这种处理方式并不适用于所有视图。现实还真是残酷。）

为通知PhotoPageActivity自己处理设备配置调整，可在AndroidManifest.xml配置文件中做如下调整，如代码清单31-10所示：

代码清单31-10 配置activity自己处理设备配置更改（AndroidManifest.xml）

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

  ...

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
    <activity
        android:name=".PhotoPageActivity"
        android:configChanges="keyboardHidden|orientation|screenSize" />
    ...
  </application>

</manifest>
```

android:configChanges属性表明，如果因键盘开关、屏幕方向改变、屏幕大小改变（也包括Android 3.2之后的屏幕方向变化）而发生设备配置更改，那么activity应自己处理配置更改。

运行应用，再次尝试旋转设备，这次一切都应完美了。

31.4　深入学习：注入 JavaScript 对象

我们已经知道如何使用WebViewClient和WebChromeClient类响应发生在WebView里的特定事件。然而，通过注入任意JavaScript对象到WebView本身包含的文档中，我们还可以做到更多。查阅 <http://developer.android.com/reference/android/webkit/WebView.html> 文档网页，找到addJavascriptInterface(Object,String)方法。使用该方法，可注入任意JavaScript对象到指定文档中：

```
mWebView.addJavascriptInterface(new Object() {
    public void send(String message) {
        Log.i(TAG, "Received message: " + message);
    }
}, "androidObject");
```

然后按如下方式调用：

```
<input type="button" value="In WebView!"
    onClick="sendToAndroid('In Android land')" />

<script type="text/javascript">
    function sendToAndroid(message) {
        androidObject.send(message);
    }
</script>
```

这可能有风险，因为一些可能的问题网页能够与应用直接接触。安全起见，最好能掌控有问题的HTML，要么就严格控制不要暴露自己的接口。