Android 平台开发媒体盒子

汪永松

摘　要：详尽地介绍在 Android 平台开发一款媒体盒子（Media Box）的应用程序，并对其商业化应用进行了分析。

关键词：媒体盒子；Media Box；Android

1　概述

对于媒体盒子（Media Box），相信读者都不陌生。网络上各种炫酷的音乐盒子、视频盒子等工具，就是媒体盒子程序。打开这些媒体盒子工具，用户可以及时获取到最新的媒体资源、播放排名等信息，还可以方便地对网络中的音乐和视频进行点播、下载、共享等操作。一般地，媒体盒子工具还提供了搜索平台来让用户根据条件（标题名称、艺人姓名、分类信息等）来搜索其所感兴趣的媒体资源。而且，当用户成为社区的注册用户之后，还可以将本地的资源发布到社区中，分享给其他的成员。

相比通过访问媒体网站的网页来获取媒体资源，使用媒体盒子的方式更为简单和方便。用户无需打开浏览器工具，更无需在多个浏览器窗体中切换所要的页面。同时使用媒体盒子产生的网络流量要比使用浏览器少，从而也节约了不少系统资源。

2　分析

2.1　分析思路

使用十六进制读取软件解析音乐盒工具的主程序文件，找出该工具所使用的 API 或组件类的名称（如图 1 所示），再根据这些 API 或组件类的功能说明，并结合工具的使用特征来推断出其应用机制。

```
64 40 40 50 41 55 31 32 40 40 40 00 00 00 00 00 ; d@@PAU12@@@.....
74 CA 5B 00 00 00 00 00 2E 3F 41 56 43 57 65 62 ; t?[......?AVCWeb
42 72 6F 77 73 65 72 32 40 40 00 00 00 00 00 00 ; Browser2@@......
74 CA 5B 00 00 00 00 00 2E 3F 41 56 43 4D 73 67 ; t?[......?AVCMsg
```

图 1　解析音乐盒工具程序文件

2.2　分析结果

使用上述的分析方法，找到该音乐盒工具所使用的 API 或组件类，如表 1 所示。

通过表 1，初步判断该音乐盒工具内嵌了网页浏览器组件和媒体播放器组件，通过从互联网读取媒体资源文件来实现多媒体数据的下载和播放。至此，就会产生如何将网页中的媒体资源的 URL 传递给播放器组件的问题。

表 1　音乐盒工具所使用 API 或组件对象类

序号	API 或组件类	说明
1	CWebBrowser2	Windows 网页浏览器 ActiveX 组件
2	IHTMLDocument2	表示 HTML 文档的接口，可以用于获取文档信息、检查和修改文档中的 HTML 元素和文本
3	IDocHostUIHandler	为托管网页浏览器组件的应用程序提供替换组件的选择菜单、上下文菜单的接口
4	InternetReadFile	从互联网读取文件（WinInet 函数）
5	InternetConnect	连接到互联网（WinInet 函数）
6	CWMPPlayer4	Windows 媒体播放器 ActivieX 组件

结合对该音乐盒工具的使用，作者发现，网页浏览器组件的上下文菜单被屏蔽。当用户点击媒体资源链接（Anchor）时，该资源将自动加入到播放列表当中。从而可以推断，该工具“截获”了网页浏览器组件对网页中链接元素的分析，并把与媒体资源相关的链接“提供”给播放器组件。

2.3　分析结论

至此，相信读者已经大致明白了该音乐盒工具的运作机制：当工具启动时，会通过互联网连接到指定的服务器，并从服务端获取包含媒体资源并以网页形式存储的数据文件，继而将该文件载入到网页浏览器组件中。当用户点击网页中所包含的链接时，工具会对当前所请求的链接进行“截获”。如果判断是媒体文件链接，那么工具会将该项添加到播放列表，并根据链接信息通过互联网获取文件流，继而提供给媒体播放器组件进行播放。如图 2 所示。

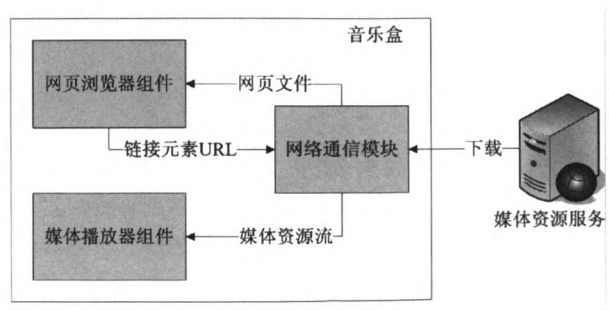

图 2　音乐盒工具的运作示意图

3 设计

3.1 运行机制

通过以上的分析结论，相信读者对于在 Android 平台中开发媒体盒子程序也有了大致的思路。同样的，Android 平台中的媒体盒子程序也有网页浏览器组件（WebView）和媒体播放器组件（VideoView 和 MediaPlayer），而且幸运的是，Android 平台简化了通过网络来获取文件流的过程：网页浏览器组件只需要指定网页资源的 URL 即可实现网页的加载；播放器组件只需指定播放资源的 URL 即可实现自动播放。

对于用户所点选链接的 URL 的“截获”行为，将由网页视图客户端（WebViewClient）接口来进行处理。图 3 是在 Android 平台中的媒体盒子工具的运作示意图。

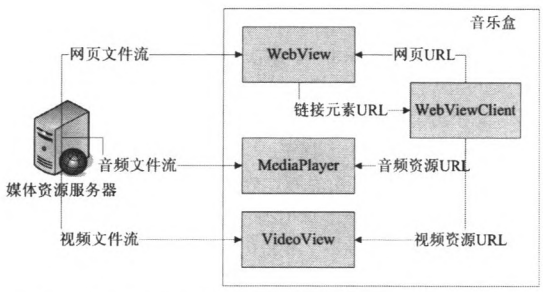

图 3 Android 平台中媒体盒子工具的运作示意图

3.2 界面设计

此外，考虑到手机设备中屏幕资源的“紧缺”，所以 Android 平台中的媒体盒子工具的用户界面不可能像 PC 平台中的那样“平铺直叙”，而是需要“紧凑”一些。这里，选用的主要的界面组件是标签页视图（TabHost），该组件继承于框布局（FrameLayout）。图 4 是该媒体盒子的主界面。

图 4 Android 平台中媒体盒子工具的主界面

3.3 界面板块

通过图 4 可以看出，该媒体盒子工具分为“推荐资源”、“当前播放”、“播放列表”和“本地资源”这 4 个标签页。其中：

（1）“推荐资源”页所展示的是从服务端获取的、包含媒体资源信息的网页，其效果如图 4 所示。该页的主要组件是网页视图（WebView）。

（2）“当前播放”页所展示的是当前所播放的视频内容，如图 5 所示。

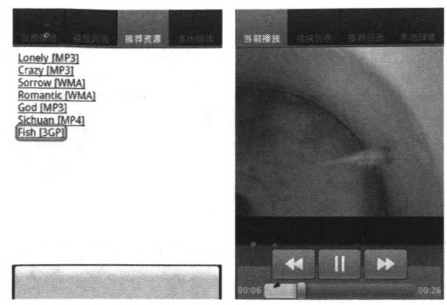

图 5 视频播放界面

该页面主要的组件是视频视图（VideoView）和播放控制器（MediaController）。当用户在“推荐资源”页的网页视图中点击视频资源链接（图 5 中所选取的视频资源类型为 3GP）时，媒体盒子将切换到“当前播放”页，并开始播放视频。

（3）“播放列表”页所展示的是当前添加到播放序列的音频资源列表，如图 6 所示。

图 6 添加音频播放列表

该页面主要的组件是列表视图（ListView）。当用户在“推荐资源”页的网页视图中点击音频资源链接（图 6 中所选取的音频资源类型为 MP3）时，媒体盒子将切换到“播放列表”页，并将当前播放项添加到列表首位，继而开始播放音频。

（4）“本地媒体”页将要展示的是通过扫描本地存储器之后形成的媒体资源列表，该功能在这里不予关注。

3.4 后台功能

3.4.1 加载网页

通过网页视图实例的“loadUrl”方法可以载入 URL 所指定的网页资源。

3.4.2 分发资源 URL

当用户点击网页中的链接时，网页客户端（WebViewClient）实例可以通过重载 URL 的加载方法来“截获”目标资源的 URL，再通过判断 URL 所指明的资源类型来决定将 URL 提交给相应的组件（网页视图、媒体播放器或视频视图）。

3.4.3 媒体播放

通过 MediaPlayer 实例的“setDataSource”方法可以设置

URL 所指定的音频媒体为播放的数据源；通过 VideoView 实例的“setVideoPath”方法可以设置所要播放的视频资源路径。

3.4.4 播放列表更新

播放列表的更新包括添加项和调整项顺序。最近播放的项总在整个播放列表的首位。如图 7 中，当选择播放列表中第 2 项后，第 2 项的位置将调整为首位，原第 1 项的位置调整为第 2。

图 7 更新播放列表

播放列表使用的是列表视图，列表项的顺序更新实际上是组件所关联的适配器（Adapter）的数据容器中的元素位置的更新，通过适配器的“notifyDataSetChanged”方法就可以实现界面与数据容器的“同步”显示。

3.4.5 资源下载

当用户长按（相当于 PC 平台的右键菜单）网页视图中链接时，将会弹出上下文菜单（Context Menu），其中的“下载”菜单项用于下载链接所指向的媒体资源，如图 8 所示。

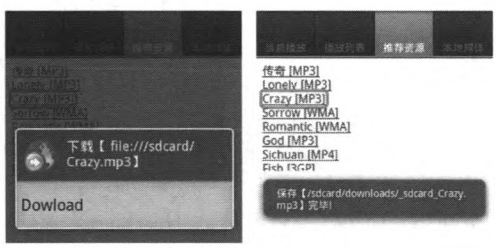

图 8 下载媒体资源

资源下载不仅需要连接到 URL 所指定的连接结点，而且还需要在本地创建文件。通过网络连接接口（URLConnection）从 URL 所描述的结点处读取资源的字节流，再写入到本地文件中，即实现下载。

4 开发

4.1 界面布局定义

代码 1 是图 4 所示的媒体盒子工具的界面布局定义。

代码 1 媒体盒子界面布局定义

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
3   android:orientation="vertical"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent">
6   <TabHost android:id="@android:id/tabhost"
7   android:layout_width="fill_parent"
8   android:layout_height="400sp">
9   <LinearLayout android:orientation="vertical"
10  android:layout_width="fill_parent"
11  android:layout_height="fill_parent">
12  <TabWidget android:id="@android:id/tabs"
13  android:layout_width="fill_parent"
14  android:layout_height="wrap_content" />
15  <FrameLayout android:id = "@android:id/tabcontent"
16  android:layout_width="fill_parent"
17  android:layout_height="fill_parent">
18  <VideoView android:id="@+id/videoView"
19  android:layout_width="fill_parent"
20  android:layout_height="fill_parent" />
21  <ListView android:id="@+id/playListView"
22  android:layout_width="fill_parent"
23  android:layout_height="fill_parent" />
24  <WebView android:id="@+id/webView"
25  android:layout_width="fill_parent"
26  android:layout_height="fill_parent" />
27  <TextView android:id="@+id/localView"
28  android:layout_width="fill_parent"
29  android:layout_height="fill_parent"
30  android:text="本地媒体资源" />
31  </FrameLayout>
32  </LinearLayout>
33  </TabHost>
34  <Button android:id="@+id/controller"
35  android:layout_width="fill_parent"
36  android:layout_height="100sp" />
37  </LinearLayout>
```

代码 1 中第 6 行即为标签页组件（TabHost）的定义，有关标签页组件的用法请参考 Android SDK 的参考。此外，第 34 行中定义的按钮组件用于“绑缚”播放控制器。

4.2 工程清单

代码 2 是媒体盒子工具的工程清单文件（AndroidManifest.xml）内容。

代码 2 媒体盒子工程清单文件内容

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android = "http://schemas.android.com/apk/res/android"
3   package="foolstudio.demo"
4   android:versionCode="1"
5   android:versionName="1.0">
6   <application android:icon="@drawable/icon" android:label="@string/app_name"
7   android:theme="@android:style/Theme.NoTitleBar">
8   <activity android:name = ".MediaBoxAct" android:label="@string/app_name">
9   <intent-filter>
```

```
10      <action android:name="android.intent.action.MAIN" />
11      <category android:name = "android.intent.category.
LAUNCHER" />
12        </intent-filter>
13    </activity>
14   </application>
15  <uses-sdk android:minSdkVersion="8" />
16    <uses-permission android:name="android.permission.
INTERNET"/>
17    <uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE"/>
18  </manifest>
```

代码 2 中第 7 行设置了该工具的界面主题为无状态栏（“NoTitleBar”），所以工具的主界面（图 4）没有状态栏。从第 8 行读者可以看出，工具的主 Activity 组件是 MediaBoxAct。第 15 行中的 SDK 版本为 8 表示 Android 的版本为 2.2。

特别的，第 16 行和第 17 行声明了 2 个使用许可（Uses-permission），第 16 行的是允许访问互联网，因为有通过互联网获取资源 URL 的需要；第 17 行的是允许往扩展存储器（SD 卡）上写入内容，因为有下载文件到外部存储器的需要。

4.3 功能模块划分

根据功能之间的耦合度，作者将该媒体盒子工程划分为 4 个模块：

（1）配置接口（Config），定义工程中所有配置（例如：下载目录、服务主页等）的常量。

（2）主 Activity（MediaBoxAct），主要负责用户界面与后台的交互。

（3）网页视图客户端实例（MediaBoxWebViewClient），用于“截获”和分发通过网页视图“转发”过来的资源 URL。

（4）下载线程（DownloadThread），用于提供后台下载。

4.4 功能实现

4.4.1 主 Activity 填充界面布局

代码 3 是主 Activity 填充界面布局的关键代码。

代码 3 Activity 填充界面布局

```
1   public void onCreate(Bundle savedInstanceState) {
2   super.onCreate(savedInstanceState);
3   //填充布局资源为内容视图
4   setContentView(R.layout.main);
5
6   //初始化资源文件中所定义的组件
7   mTabHost = getTabHost();
8   mVideoView = (VideoView)findViewById (R.id.
videoView);
9   mListView = (ListView)findViewById(R.id.playListView);
10  mWebView = (WebView)findViewById(R.id.webView);
11
12  //初始化 TabHost
13  mTabHost.addTab(mTabHost.newTabSpec("tab1")
14  .setIndicator("当前播放").setContent(R.id.videoView));
15  mTabHost.addTab(mTabHost.newTabSpec("tab2")
16  .setIndicator("播放列表").setContent(R.id.playListView));
17  mTabHost.addTab(mTabHost.newTabSpec("tab3")
18  .setIndicator("推荐资源").setContent(R.id.webView));
19  mTabHost.addTab(mTabHost.newTabSpec("tab4")
20  .setIndicator("本地媒体").setContent(R.id.localView));
21  //
22  mTabHost.setCurrentTab(2);
23  …
24  }
```

代码 3 中，主 Activity 填充界面布局资源为内容视图（第 4 行）。从第 7 行到第 10 行，初始化资源文件中定义的组件（标签页视图、视频视图、列表视图、网页视图）。从第 13 行到第 22 行，初始化标签页视图中的 4 个标签页内容。

4.4.2 初始化核心组件

（1）播放列表视图

代码 4 是初始化播放列表视图的主要代码。

代码 4 初始化播放列表视图

```
1   //初始化播放列表
2   private void initList() {
3   mItems = new ArrayList<String>();
4   //使用数据容器构造列表视图适配器
5   ListAdapter adapter = new ArrayAdapter<String>(this,
6   android.R.layout.simple_list_item_1, mItems);
7   mListView.setAdapter(adapter);
8   //设置列表视图点击事件侦听器
9   mListView.setOnItemClickListener(this);
10  }
```

代码 4 中，使用数组适配器对列表视图（播放列表组件）进行初始化（第 5 行）。

（2）音频播放器

```
//初始化播放器
private void initPlayer()
    mMusicPlayer = new MediaPlayer();
}
```

（3）网页视图

代码 5 是初始化网页视图的主要代码。

代码 5 初始化网页视图

```
1   //初始化网页视图和网页视图客户端组件
2   private void initWeb() {
3   mWebViewClient = new MediaBoxWebViewClient
(this);
4   mWebView.getSettings().setJavaScriptEnabled(true);
5   mWebView.setWebViewClient(mWebViewClient);
6   //设置上下文菜单
7   this.registerForContextMenu(mWebView);
8   //载入网页资源
```

```
9    mWebView.loadUrl("file:///sdcard/index.html");
10   }
```

代码5中，第3行中构造了网页视图客户端实例，并绑定到网页视图（第5行）。第9行中指明网页视图载入指定的网页（HTML文件）。

（4） 视频视图

代码6是初始化视频视图的主要代码。

代码6 初始化视频视图

```
1    //初始化视频视图
2    private void initVideoView() {
3    //构造播放控制器
4    mController = new MediaController(this);
5    mCtrlView = (Button)findViewById(R.id.controller);
6    //将播放控制器绑缚可视组件上
7    mController.setAnchorView(mCtrlView);
8    //设置播放控制器的视频视图
9    mController.setMediaPlayer(mVideoView);
10   //设置视频视图的控制器
11   mVideoView.setMediaController(mController);
12   }
```

代码6中，创建了一个播放控制器实例（第4行），然后将其绑缚到可视组件中（第7行），同时将播放控制器与视频视图相互绑定（第9行和第11行）。

4.4.3 主界面线程消息队列处理器

代码7是初始化主界面线程消息队列（Message Queue）处理器的主要代码。消息队列处理器给下载线程提供了向主Activity发送消息（例如：下载完毕的消息）的接口。

代码7 初始化主界面线程消息队列处理器

```
1    //初始化主界面线程消息队列处理器
2    mHandler = new Handler() {
3    public void handleMessage(Message msg) {
4    super.handleMessage(msg);
5    //获取消息数据包
6    Bundle bundle = msg.getData();
7    //获取数据包中的消息内容
8    String msgStr = bundle.getString("Msg");
9    //显示消息内容
10   Toast.makeText (this, msgStr, Toast.LENGTH_LONG).
show();
11   }
12   };
```

代码7中，定义了消息的处理函数（第3行），一旦主线程消息队列接收到消息，就会获取消息内容（第8行）并以提示条的形式进行显示（第10行）。

4.4.4 网页链接URL分发

当用户在网页视图中点击资源链接时，视图载入对应链接URL的行为将在网页视图所绑定的网页视图客户端来进行。代码8是媒体盒子网页视图客户端的完整定义代码。

代码8 媒体盒子网页视图客户端定义

```
1        package foolstudio.demo;
2
3        import android.webkit.WebView;
4        import android.webkit.WebViewClient;
5
6        public class MediaBoxWebViewClient extends
WebViewClient {
7        private MediaBoxAct mMainAct = null;
8
9        public MediaBoxWebViewClient(MediaBoxAct act)
{
10       mMainAct = act;
11       }
12
13       //重载载入媒体资源的URL的方法(统一管理)
14       @Override
15       public boolean shouldOverrideUrlLoading (Web-
View view, String url) {
16       mMainAct.changeUrl(url);
17       //view.loadUrl(url);
18       return (true);
19       }
20       };
```

代码8中，媒体盒子网页视图客户端重载了URL的载入方法（第15行），并将所获取到的URL转给主Activity来处理（第16行），而不再执行默认的载入行为（第17行）。

代码9是主Activity对从网页视图客户端传递过来的资源URL进行判断和分发的主要代码。

代码9 代码标题

```
1    //当资源Url改变时(回调函数)
2    public void changeUrl(String url) {
3    //重置播放状态
4    stopVideo();
5    stopAudio();
6
7    if(url.endsWith(".mp4") || url.endsWith(".3gp") ) { //视频资源
8    playVideo(url);
9    mTabHost.setCurrentTab(0); //跳转到播放页
10   }
11   else if(url.endsWith(".mp3") || url.endsWith(".wma") ) {
//音频资源
12   playAudio(url);
13   mTabHost.setCurrentTab(1); //跳转到播放列表页
14   }
15   }
```

代码9中，先对播放状态进行了重置（第4行和第5行），然后在通过URL判断媒体资源类型是视频还是音频，从而调用对应的播放方法（第8行或第12行）。

4.4.5 播放音频资源

代码 10 是停止和启动播放音频资源的主要代码。

代码 10 停止和启动播放音频资源

```
1   //停止播放音频
2   private void stopAudio() {
3   if(mMusicPlayer.isPlaying() ) { //判断当前是否正在播放
4         mMusicPlayer.stop();
5   }
6   }
7
8   //播放音频
9   private void playAudio(String url) {
10  //Url 检查
11  String url2 = url.replaceFirst("file://", "");
12
13  //更新播放列表顺序
14  updateList(url2);
15
16  try {
17        mMusicPlayer.reset();
18        mMusicPlayer.setDataSource(url2);
19        mMusicPlayer.prepare();
20        mMusicPlayer.start();
21  } catch(IOException e) {
22        e.printStackTrace();
23  }
24  }
```

代码 10 中，第 11 行是对包含“file://”前缀（文件传输）的 URL 进行调整，应该去掉协议模式部分（有关 URL 的组成请参考有关资料），对“http://”前缀是无需调整的。第 18 行，通过媒体播放器的“setDataSource”方法来指定所要播放的音频资源的 URL。

4.4.6 播放视频资源

代码 11 是停止和启动播放视频资源的主要代码。

代码 11 停止和启动播放视频资源

```
1   //停止播放视频
2   private void stopVideo() {
3   if(mVideoView.isPlaying() ) { //判断当前是否正在播放
4         mVideoView.stopPlayback();
5   }
6   }
7
8   //播放视频
9   private void playVideo(String url) {
10  mVideoView.setVideoPath(url); //设置视频路径
11  mVideoView.start();
12  mVideoView.requestFocus();
13  }
```

代码 11 中，通过视频视图的“setVideoPath”方法来指定所要播放的视频资源的路径（第 10 行）。

4.4.7 资源下载

代码 12 是当用户点选上下文菜单中“下载”项进行资源下载（见图 8）的主要代码。

代码 12 启动资源下载

```
1   //当选择上下文菜单项时回调
2   public boolean onContextItemSelected(MenuItem item) {
3   switch(item.getItemId() ) {
4     case R.id.MI_DOWNLOAD: {
5       HitTestResult htr = this.mWebView.getHitTestResult
();
6             String url = htr.getExtra();
7
8             //下载资源
9             if(url ! = null) {
10                  downloadUrl(url);
11            }
12
13            break;
14        }
15  }
16  return super.onContextItemSelected(item);
17  }
18
19  //下载资源
20  private void downloadUrl(String url) {
21  //启动下载线程
22  DownloadThread t = new DownloadThread (url, mHan-
dler);
23  t.start();
24  }
```

代码 12 中，当用户点选网页视图的上下文菜单项后，可以通过视图的“getHitTestResult”方法来获取该点击测试结果（第 5 行），并获取有关的资源 URL（第 6 行）。获取到目标资源的 URL 之后，就可以以此来启动下载线程（第 22 行）。

代码 13 是下载线程的核心代码。

代码 13 下载线程核心代码

```
1   public void run() {
2   try {
3         URL url = new URL(mUrl);
4
5         String fileName = url.getFile();
6         //将路径名替换成下划线(防止文件名重名)
7       fileName = fileName.replace(File.separatorChar, '_');
8
9         //建立 URL 连接
10        URLConnection conn = url.openConnection();
11        InputStream is = conn.getInputStream();
12        BufferedInputStream bis = new BufferedInput-
Stream(is);
```

```
13          //创建文件输出流
14          FileOutputStream fos = new FileOutputStream
(Config.DOWNLOAD_DIR+
15    File.separatorChar+fileName);
16          BufferedOutputStream bos = new BufferedOut-
putStream(fos);
17
18          int aByte = -1;
19          //开始复制字节
20          while( (aByte=bis.read()) ! = -1) {
21                bos.write(aByte);
22          }
23
24          //关闭输出流
25          bos.close();
26          fos.close();
27
28          //关闭输入流
29          bis.close();
30          is.close();
31
32          showResponse ("保存【" +Config.DOWN-
LOAD_DIR+
33                File.separatorChar+fileName + "】完毕! ");
34    } catch (MalformedURLException e) {
35          e.printStackTrace();
36    } catch (IOException e) {
37          e.printStackTrace();
38    }
39    }
```

代码 13 中，通过 URL 来建立连接接口（第 10 行），并获取接口的输入流（第 11 行），用于从 URL 所指定的结点处获取字节流。同时根据文件名来创建本地文件系统的输出流（第 14 行）。通过从远程输入流到本地文件输出流的字节拷贝即完成下载过程（第 20 行到第 22 行）。

当下载线程下载完毕时，该线程通过主界面线程消息队列处理器来向主 Activity 传递消息（第 32 行），再由主 Activity 显示消息内容（见代码 7）。代码 14 是下载线程向主 Activity 发送消息的核心代码。

代码 14　下载线程向主 Activity 发送消息

```
1     //向主界面线程消息队列发送消息
2     private void showResponse(String data) {
3     Bundle bundle = new Bundle();
4     //设置消息内容
5     bundle.putString("Msg", data);
6     Message msg = new Message();
7     //设置消息数据包
8     msg.setData(bundle);
9     //发送消息
10    mHandler.sendMessage(msg);
11    }
```

读者可以看出，代码 14 中的过程与代码 7 中的是逆向的：代码 14 中是将消息内容添加到数据包（第 5 行）中并发送（第 10 行）；而代码 7 中是接收消息，并提取数据包中的消息内容。

4.4.8　更新播放列表

代码 15 是更新播放列表的主要代码。

代码 15　更新播放列表

```
1     private void updateList(String url) {
2     //更新列表适配器的数据容器
3     if(mItems.size() > 0) {
4           int index = mItems.indexOf(url);
5
6           if(index ! = -1) { //如果存在重复项
7                 mItems.remove(index);
8           }
9
10          //将新增项放置为首位
11          mItems.add(0, url);
12    }
13    else { //容器内为空时直接添加
14          mItems.add(url);
15    }
16
17    //通知更新数据集
18    ((ArrayAdapter <String > )mListView.getAdapter ()).notify-
DataSetChanged();
19    }
```

代码 15 中，首先要对新增项进行是否重复的判断（第 4 行），如果存在重复项（第 6 行），则需要先删除已存在项对应的元素（第 7 行）。总而言之，新增项都会放置到列表的首位（第 11 行）。

当数据集更新之后，还需要通过列表适配器来“通知”列表视图重绘内容（第 18 行）。

5　结语

通过上述的介绍，相信读者已经大致理解在 Android 平台下开发媒体盒子程序的功能框架和过程细节。而且，只要读者有一定的 Android 平台开发经验，甚至可以依葫芦画瓢地开发出一款定制的媒体盒子工具。但是从商业应用的角度而言，运行于移动设备上的媒体盒子工具需要无线网络和内容管理平台的支持。通过 GPRS 或 Wi-Fi 接入的方式，连接到无线互联网服务商所提供的无线网络，这样才能与内容管理服务器进行通信，从服务端获取媒体资源信息。

可以预见，随着无线互联网的使用成本日益降低，支持移动设备的媒体盒子必将成为众多手机用户不可缺少的娱乐工具，拭目以待吧。

（收稿日期：2010-08-18）