第 5 章 软件管家

第 5 章 软件管家模块

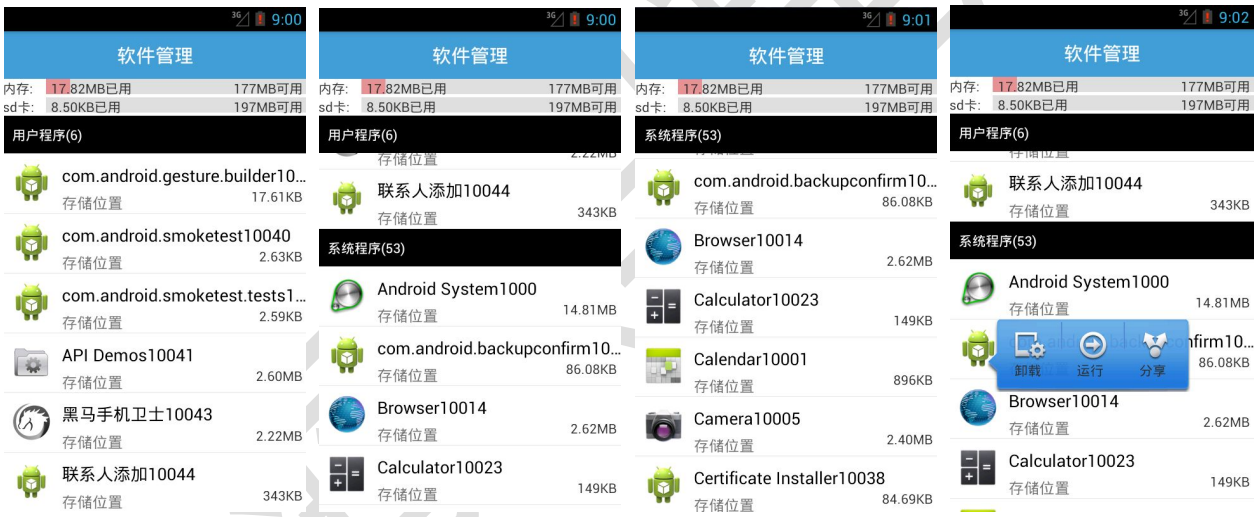
- **了解软件管家模块功能**
- **掌握软件管家模块的开发**

在日常生活中，每个手机都装有很多程序，为了方便管理这些程序，我们开发了软件管家模块，该模块用于管理手机中的所有程序，当选中某个程序时，可以通过弹出的条目对程序进行启动、卸载、分享、设置，本章将针对软件管家模块进行详细地讲解。

5.1 模块概述

软件管家模块用于获取当前手机中的所有应用，包括用户程序和系统程序，以及手机剩余内存、SD 卡剩余内存，如图 5-1（a）-（c）。

当点击某个应用时，下方会浮出一个操作条，可以管理程序的启动、卸载、分享，该功能界面效果如图 5-1（d）所示。



（a） （b） （c） （d）

图 5-1 软件管理界面

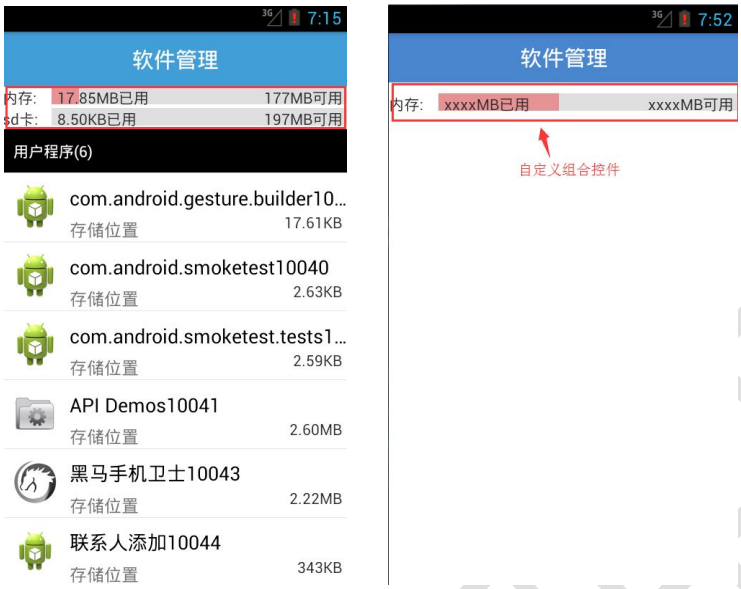
5.2 软件管家 UI

5.2.1 自定义组合控件

首先，我们分析软件管家界面效果图中红圈标注的部分，如图 5-2（a），该部分是用于显示手机内存信息和 SD 卡内存信息，每一部分都包含三个 TextView 和一个 ProgressBar，如果将这些控件的代码都写在主界面的布局文件则会显得代码过于繁多，不利于以后的代码维护。

为解决这个问题，我们可以把这些控件组合在一起，定义一个具有水平进度条和三个 TextView 的组合控件 ProgressDesView，这样在书写主界面布局文件时也会变得更加简便。自定义组合控件的效果如图 6-2

（b）所示。



（a） （b）

图 5-2 软件管家界面和自定义组合控件

关于自定义组合控件 ProgressDesView 的布局文件 progress_des_view.xml 的代码如下所示。

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:orientation="horizontal" >
6.      <TextView
7.          android:id="@+id/tv_title_des"
8.          android:layout_width="45dp"
9.          android:layout_height="wrap_content"
10.         android:text="内存:" />
11.     <RelativeLayout
12.         android:layout_width="match_parent"
13.         android:layout_height="wrap_content" >
14.         <ProgressBar
15.             android:id="@+id/pb_des"
16.             style="?android:attr/progressBarStyleHorizontal"
17.             android:layout_width="match_parent"
18.             android:layout_height="wrap_content"
19.             android:progressDrawable="@drawable/progress_horizontal"
20.             android:progress="40"/>
21.         <TextView
22.             android:id="@+id/tv_left"
23.             android:layout_width="wrap_content"
24.             android:layout_height="wrap_content"
25.             android:layout_marginLeft="5dp"
```

```
26.           android:text="xxxxMB 已用" />
27.        <TextView
28.           android:id="@+id/tv_right"
29.           android:layout_width="wrap_content"
30.           android:layout_height="wrap_content"
31.           android:layout_alignParentRight="true"
32.           android:layout_marginRight="5dp"
33.           android:text="xxxxMB 可用" />
34.    </RelativeLayout>
35. </LinearLayout>
```

上面 ProgressBar 中的 progressDrawable 属性，它所引用的是用于设置进度条的背景层次关系的 progress_horizontal 文件，这是为了进度条的效果更加美观，我们设置其主进度条和副进度条的颜色，具体如文件【5-1】所示。

【文件 5-1】 res/drawable/ProgressDesView.java

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
3.     <!-- layer-list 表示层级关系 -->
4.     <!-- 背景 -->
5.     <item android:id="@android:id/background">
6.        <shape>
7.           <!-- corners 边角 -->
8.           <!-- <corners android:radius="5dip" /> -->
9.           <!--梯度渐变色
10.            android:angle="270" 角度 逆时针旋转 270 度 -->
11.          <!-- <gradient -->
12.          <!-- android:startColor="#ff9d9e9d" -->
13.          <!-- android:centerColor="#ff5a5d5a" -->
14.          <!-- android:centerY="0.75" -->
15.          <!-- android:endColor="#ff747674" -->
16.          <!-- android:angle="270" -->
17.          <!-- /> -->
18.          <!-- solid 固定色 -->
19.          <solid android:color="#22000000" />
20.       </shape>
21.    </item>
22.    <!-- 副进度条 (第二进度条) -->
23.    <item android:id="@android:id/secondaryProgress">
24.       <clip>
25.          <shape>
26.             <!-- <corners android:radius="5dip" /> -->
27.             <!-- <gradient -->
28.             <!-- android:startColor="#80ffd300" -->
29.             <!-- android:centerColor="#80ffb600" -->
30.             <!-- android:centerY="0.75" -->
```

```
31.              <!-- android:endColor="#a0ffcb00" -->
32.              <!-- android:angle="270" -->
33.              <!-- /> -->
34.              <solid android:color="#22ff0000" />
35.          </shape>
36.       </clip>
37.    </item>
38.    <!-- 主进度条 -->
39.    <item android:id="@android:id/progress">
40.       <clip>
41.          <shape>
42.             <!-- <corners android:radius="5dip" /> -->
43.             <!-- <gradient -->
44.             <!-- android:angle="270" -->
45.             <!-- android:centerColor="#ffffb600" -->
46.             <!-- android:centerY="0.75" -->
47.             <!-- android:endColor="#ffffcb00" -->
48.             <!-- android:startColor="#ffffd300" /> -->
49.             <solid android:color="#55ff0000" />
50.          </shape>
51.       </clip>
52.    </item>
53. </layer-list>
```

ProgressDesView 布局文件我们已经写好，下面我们需要在 ProgressDesView 中拿到指定 id 的 View 对象，方便我们使用 ProgressDesView，具体代码如文件【5-2】所示。

【文件 5-2】 com.itheima.mobilesafe_sh3.act.view/ProgressDesView.java

```
1.  public class ProgressDesView extends LinearLayout {
2.      private TextView tv_title_des;
3.      private ProgressBar pb_des;
4.      private TextView tv_left;
5.      private TextView tv_right;
6.      public ProgressDesView(Context context, AttributeSet attrs, int defStyle) {
7.          super(context, attrs, defStyle);
8.      }
9.      public ProgressDesView(Context context, AttributeSet attrs) {
10.         super(context, attrs);
11.         /**
12.          * 第三个参数：表示是否挂载在父类身上
13.          * null: 说明不需要挂载
14.          * this：说明需要挂载
15.          */
16.         View.inflate(context, R.layout.progress_des_view, this);
17.         //标题
```

```
18.         tv_title_des = (TextView) findViewById(R.id.tv_title_des);
19.         //进度条
20.         pb_des = (ProgressBar) findViewById(R.id.pb_des);
21.         //左边的文本
22.         tv_left = (TextView) findViewById(R.id.tv_left);
23.         //右边的文本
24.         tv_right = (TextView) findViewById(R.id.tv_right);
25.     }
26.     public ProgressDesView(Context context) {
27.         this(context, null);
28.     }
29.     /**
30.      * 设置标题
31.      * @param title
32.      */
33.     public void setTitle(String title){
34.         tv_title_des.setText(title);
35.     }
36.     /**
37.      * 设置进度条进度
38.      * @param progress
39.      */
40.     public void setProgress(int progress){
41.         pb_des.setProgress(progress);
42.     }
43.     /**
44.      *  设置已经使用的内存
45.      * @param left
46.      */
47.     public void setTvLeft(String left_text){
48.         tv_left.setText(left_text);
49.     }
50.     /**
51.      * 设置可以使用内存
52.      * @param right_text
53.      */
54.     public void setTvRight(String right_text){
55.         tv_right.setText(right_text);
56.     }
57. }
```

5.2.2 软件管家主界面 UI

前面我们已经定义了用于显示手机内存信息和 SD 卡内存信息的组合控件，根据图 4-3 的界面可知，标题栏的下方是两个组合控件，最面是一个帧布局，帧布局中放置一个 ListView 用于展示应用程序信息，ListView 的第一个条目用于说明当前展示的程序是系统程序还是用户程序。需要注意的是，这个 TextView 中的文本信息是在代码中动态设置的。软件管家的图形化主界面如图 5-3 所示。

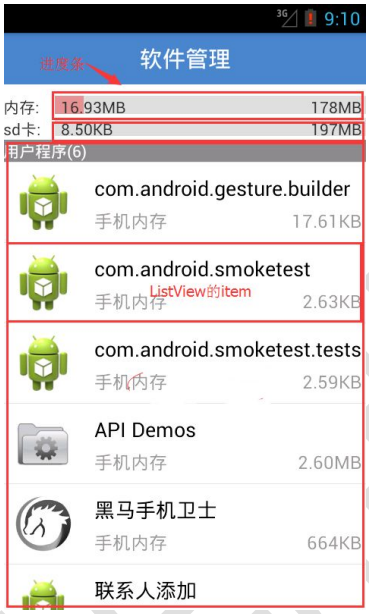


图 5-3 软件管家主界面分析

图 5-3 软件管家界面对应的布局文件如文件【5-3】所示。

【文件 5-3】 res/layout/activity_app_manager.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:orientation="vertical" >
6.      <TextView
7.          style="@style/TitleBarTextView"
8.          android:text="软件管理" />
9.      <!-- 内存 -->
10.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
11.         android:id="@+id/pdv_rom"
12.         android:layout_width="match_parent"
13.         android:layout_height="wrap_content"
14.         android:layout_marginTop="10dp" />
15.     <!-- sd卡 -->
16.    <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
17.         android:id="@+id/pdv_sd"
18.         android:layout_width="match_parent"
```

```
19.         android:layout_height="wrap_content" />
20.    <FrameLayout
21.         android:layout_width="match_parent"
22.         android:layout_height="match_parent" >
23.         <ListView
24.             android:id="@+id/list_view"
25.             android:layout_width="match_parent"
26.             android:layout_height="wrap_content" >
27.         </ListView>
28.         <ProgressBar
29.             android:id="@+id/pb"
30.             android:layout_width="wrap_content"
31.             android:layout_height="wrap_content"
32.             android:layout_gravity="center"
33.             android:indeterminateDrawable="@drawable/progress_medium"
34.             android:visibility="invisible" />
35.     </FrameLayout>
36. </LinearLayout>
```

上述布局文件中的 ProgressBar 主要用于提示用户当前界面正在处于加载状态，当数据获取完毕时该进度条便会消失。

5.2 软件管家逻辑

5.2.1 获取手机内存和 SD 卡内存

布局文件前面我们已经写好，这里，我们首先需要实现获取手机的内存和 SD 卡内存的信息，AppManagerActivity 中的逻辑代码如下所示。

```
1.  public class AppManagerActivity extends Activity {
2.      private ProgressDesView pdv_rom;
3.      private ProgressDesView pdv_sd;
4.      private ListView mListview;
5.       private ProgressBar pb;
6.      @Override
7.      protected void onCreate(Bundle savedInstanceState) {
8.          // TODO Auto-generated method stub
9.          super.onCreate(savedInstanceState);
10.         initView();
11.         initData();
12.     }
13.     private void initData() {
14.         // 获取到剩余内存
15.         long romFreeSpace = Environment.getDataDirectory().getFreeSpace();
```

```
16.         // 获取到总共有多少内存
17.         long romTotalSpace = Environment.getDataDirectory().getTotalSpace();
18.         // 获取到已经使用了的内存
19.         long userSpace = romTotalSpace - romFreeSpace;
20.         // 进度条进度
21.         int progress = (int) (userSpace * 100f / romTotalSpace + 0.5f);
22.         System.out.println("----------------------------------");
23.         System.out.println("剩余内存--- "
24.                 + Formatter.formatFileSize(this, romFreeSpace));
25.         System.out.println("总共内存--- "
26.                 + Formatter.formatFileSize(this, romTotalSpace));
27.         System.out.println("使用内存--- "
28.                 + Formatter.formatFileSize(this, userSpace));
29.         // 设置内存的数据
30.         pdv_rom.setTitle("内存:");
31.         // 已经使用内存
32.         pdv_rom.setTvLeft(Formatter.formatFileSize(this, userSpace));
33.         // 可以使用内存
34.         pdv_rom.setTvRight(Formatter.formatFileSize(this, romFreeSpace));
35.         // 设置进度条
36.         pdv_rom.setProgress(progress);
37.         // 获取到 sd 卡的剩余目录
38.         long sdFreeSpace = Environment.getExternalStorageDirectory()
39.                 .getFreeSpace();
40.         // 获取到 sd 卡总的目录
41.         long sdTotalSpace = Environment.getExternalStorageDirectory()
42.                 .getTotalSpace();
43.         // 获取到 sd 卡的使用目录
44.         long sdUserSpace = sdTotalSpace - sdFreeSpace;
45.         // 获取到 sd 卡进度条进度
46.         int sdProgress = (int) (sdUserSpace * 100f / sdTotalSpace + 0.5f);
47.         // 设置 sd 卡的数据
48.         pdv_sd.setTitle("sd 卡:");
49.         // 已经使用内存
50.         pdv_sd.setTvLeft(Formatter.formatFileSize(this, sdUserSpace));
51.         // 可以使用内存
52.         pdv_sd.setTvRight(Formatter.formatFileSize(this, sdFreeSpace));
53.         // 设置进度条
54.         pdv_sd.setProgress(sdProgress);
55.     }
56.     //初始化 View
57.     private void initView() {
58.         setContentView(R.layout.activity_app_manager);
```

```
59.         // 初始化内存控件
60.         pdv_rom = (ProgressDesView) findViewById(R.id.pdv_rom);
61.         // 初始化 sd 卡
62.         pdv_sd = (ProgressDesView) findViewById(R.id.pdv_sd);
63.         pb = (ProgressBar) findViewById(R.id.pb);
64.         // 展示进度条
65.         pb.setVisibility(View.VISIBLE);
66.     }
67. }
```

- 第 14~36 行获取手机内存的可用空间和全部空间，并通过 Formatter 的 formatFileSize 函数将获取的内存大小转换为 MB 单位，并将其显示在布局中的内存信息。
- 第 37~54 行获取手机 SD 卡的可用空间和全部空间，并通过 Formatter 的 formatFileSize 函数将获取的内存大小转换为 MB 单位，并将其显示在布局中的 SD 卡内存信息。
- 第 57~66 行初始化布局文件中的 View 对象，便于以后使用

此时，运行程序，效果图如图 5-4 所示。

图 5-4 主界面内存信息

5.2.2 应用程序的实体类

在获取应用程序列表之前首先需要创建一个实体类（AppInfo.java），该类用于存储应用程序的相关信息，如程序包名、程序图标、程序大小等，具体代码如【文件 5-4】所示。

【文件 5-4】 AppInfo.java

```
1.  public class AppInfo {
2.      // Drawable 不仅仅表示图片 表示资源文件 表示的范围更广泛
3.      // bitmap 只能单纯表示图片
4.      /**
5.       * 表示应用的图标
6.       */
7.      public Drawable icon;
```

```
8.      /**
9.       * 应用的名字
10.      */
11.     public String appName;
12.     /**
13.      * 存放 app 的位置 手机内存 true 或者是 sd 卡 false
14.      */
15.     public boolean isRom;
16.     /**
17.      * app 的大小
18.      */
19.     public long appSize;
20.     /**
21.      * 是否是用户程序 true 表示用户程序 false 表示系统程序
22.      */
23.     public boolean isUserApp;
24.     /**
25.      * 应用程序的包名
26.      */
27.     public String appPackageName;
28. }
```

5.2.3 应用程序的工具类

在开发软件管家模块时，需要获取手机中所有已安装程序，且 ListView 中每个条目都是手机里的一个安装程序，如图 5-5 所示。由于该功能比较独立，因此，可以将其专门定义一个工具类，以提高代码的可移植性，具体代码如【文件 5-5】所示。

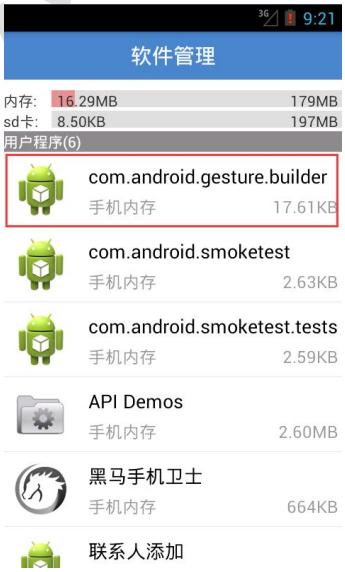


图 5-5　ListView 中应用程序的 Item 信息

【文件 5-5】 AppInfoParser.java

```
1.  public class AppInfoParser {
2.      /**
3.       * 获取到所有的 app 的基本信息
4.       * @return
5.       */
6.      public static List<AppInfo> getAppInfos(Context context) {
7.          // 获取到安装包的管理者
8.          PackageManager packageManager = context.getPackageManager();
9.          // 获取到所有的安装包
10.         // 参数：获取到所有的标记
11.         List<PackageInfo> installedPackages = packageManager
12.                 .getInstalledPackages(0);
13.         List<AppInfo> lists = new ArrayList<AppInfo>();
14.         // 迭代所有的安装包
15.         for (PackageInfo packageInfo : installedPackages) {
16.             AppInfo info = new AppInfo();
17.             // 获取到应用的包名
18.             String appPackageName = packageInfo.packageName;
19.             info.appPackageName = appPackageName;
20.             // 获取到手机的图片
21.             Drawable icon = packageInfo.applicationInfo
22.                     .loadIcon(packageManager);
23.             info.icon = icon;
24.             // 获取到所有的安装包的名字
25.             String appName = packageInfo.applicationInfo.loadLabel(
26.                     packageManager).toString();
27.             info.appName = appName;
28.             // 获取到当前 apk 安装的位置
29.             String sourceDir = packageInfo.applicationInfo.sourceDir;
30.             File file = new File(sourceDir);
31.             // 获取到当前 apk 的大小
32.             long appSize = file.length();
33.             info.appSize = appSize;
34.             System.out.println("---------------------------------");
35.             System.out.println("应用的名字---" + appName);
36.             System.out.println("应用的包名---" + appPackageName);
37.             System.out.println("应用的目录---" + sourceDir);
38.             System.out.println("应用的大小---" + appSize);
39.             // 如果是系统 app /system/app
40.             // 如果是用户 app /data/data/app
41.             if (sourceDir.startsWith("/system")) {
42.                 // 安装的系统的 app
43.                 info.isUserApp = false;
```

```
44.                 System.out.println("系统 app");
45.             } else {
46.                 // 用户 app
47.                 info.isUserApp = true;
48.                 System.out.println("用户 app");
49.             }
50.             // 获取到当前 app 的标记
51.             int flags = packageInfo.applicationInfo.flags;
52.             // 判断当前是否是系统应用
53.             if ((flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
54.                 // 系统应用
55.                 System.out.println("系统 app");
56.             } else {
57.                 // 用户应用
58.                 System.out.println("用户 app");
59.             }
60.             //判断当前的 apk 是否是在外部存储(sd 卡)
61.             if((flags &  ApplicationInfo.FLAG_EXTERNAL_STORAGE) != 0){
62.                 //sd 卡
63.                 info.isRom = false;
64.             }else{
65.                 //机身内存
66.                 info.isRom = true;
67.             }
68.             SystemClock.sleep(10);
69.             lists.add(info);
70.         }
71.         return lists;
72.     }
73. }
```

- 第 8~13 行代码用于获取包管理器，通过包管理器获取手机中已安装的应用程序的包信息，并存储到 List<PackageInfo>集合中。
- 第 14 行代码定义了一个 List<AppInfo>集合，该集合用于存储获取到的应用程序信息。
- 第 17~28 行代码用于遍历 installedPackages 集合对象，并将每个应用程序的包名、图标、应用名称存储到 AppInfo 对象中。
- 第 29~34 行代码用于获取应用程序的大小，首先获取到应用程序的路径，然后通过 File 对象获取到应用程序的大小，并存储到 AppInfo 对象中。
- 第 40~50 行代码用于判断程序的安装位置，通过第 30 行获取到 Apk 的安装路径 sourceDir 的开头，进行判断应用程序是否安装在外部存储中，如果以 system 开头则是系统应用，将 AppInfo 的 isUerApp 属性设置为 false，否则就是用户应用，将 isUserApp 属性设置为 true。
- 第 51~52 行代码用于获取程序的二进制映射的 flags。
- 第 54~60 行的代码是通过 flags 标记来判断当前应用程序是否为系统应用，如果判断条件即(ApplicationInfo.FLAG_SYSTEM&flags)!=0 成立，则是系统应用，将 AppInfo 的 isUerApp 属性设

置为 false，否则就是用户应用，将 isUserApp 属性设置为 true。

- 第 61~68 行判断当前的 apk 是否是在外部存储(sd 卡)。
- 第 70 行代码将 AppInfo 对象添加到 List<AppInfo>集合中，并返回整个集合 lists。

5.2.4 软件管家界面 ListView 的条目布局

观察主界面中 ListView 的展示应用程序信息的条目可知，如图 5-6 所示，这里我们需要定义一个布局文件用于显示这些信息，具体代码如文件【5-6】所示。

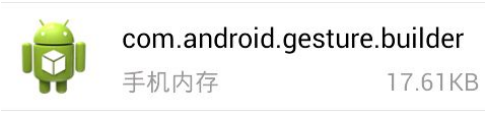


图 5-6　ListView 的 Item 详细信息

【文件 5-6】 res/layout/item_activity_manager.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="match_parent"
4.      android:layout_height="match_parent"
5.      android:orientation="horizontal" >
6.      <!-- icon 图标 -->
7.      <ImageView
8.          android:id="@+id/iv_icon"
9.          android:layout_width="50dp"
10.         android:layout_height="50dp"
11.         android:layout_margin="10dp"
12.         android:src="@drawable/ic_launcher" />
13.
14.     <RelativeLayout
15.         android:layout_width="match_parent"
16.         android:layout_height="wrap_content" >
17.         <!-- 应用的名字 -->
18.         <TextView
19.             android:id="@+id/tv_app_name"
20.             android:layout_width="wrap_content"
21.             android:layout_height="wrap_content"
22.             android:layout_marginLeft="10dp"
23.             android:layout_marginTop="10dp"
24.             android:text="黑马手机卫士"
25.             android:textColor="#000"
26.             android:textSize="18sp" />
27.         <!-- 应用存放的位置 -->
28.         <TextView
29.             android:id="@+id/tv_app_location"
30.             android:layout_width="wrap_content"
```

```
31.             android:layout_height="wrap_content"
32.             android:layout_below="@id/tv_app_name"
33.             android:layout_marginLeft="10dp"
34.             android:layout_marginTop="5dp"
35.             android:text="手机内存"
36.             android:textColor="#66000000"
37.             android:textSize="16sp" />
38.         <!-- 应用的大小 -->
39.         <TextView
40.             android:id="@+id/tv_app_size"
41.             android:layout_width="wrap_content"
42.             android:layout_height="wrap_content"
43.             android:layout_alignParentRight="true"
44.             android:layout_below="@id/tv_app_name"
45.             android:layout_marginTop="5dp"
46.             android:text="18MB"
47.             android:layout_marginRight="5dp"
48.             android:textColor="#66000000"
49.             android:textSize="16sp" />
50.     </RelativeLayout>
51. </LinearLayout>
```

5.2.5 软件管家界面 ListView 的数据集合

这里我们需要获取所有的应用程序，但是手机中的应用程序可分为用户程序和系统程序，如下图 5-7 所示，红色圆圈标注的地方。因此，我们需要在所获取的所有应用程序集合中进行遍历，判断当前获取的是应用程序还是系统程序，并放入对应的集合中，用于 ListView 中不同集合数据的展示。

图 5-7　ListView 中应用程序分类

数据集合的创建应在 initData( )中执行，具体的代码如下所示。

```
1.      private void initData() {
2.      ......
3.      new Thread() {
4.              public void run() {
5.                  // 获取到所有的安装包
6.                  mAppInfos = AppInfoParser.getAppInfos(AppManagerActivity.this);
7.                  // 用户程序集合
8.                  userLists = new ArrayList<AppInfo>();
9.                  systemLists = new ArrayList<AppInfo>();
10.                 // 把所有的程序一分为二
11.                 for (AppInfo info : mAppInfos) {
12.                     if (info.isUserApp) {
13.                         // 用户程序
14.                         userLists.add(info);
15.                     } else {
16.                         // 系统程序
17.                         systemLists.add(info);
18.                     }
19.                 }
20.                 handler.sendEmptyMessage(0);
21.             };
22.         }.start();
23.     }
```

5.2.6 软件管家界面 ListView 的数据适配器

软件管家界面中使用 ListView 控件展示应用程序，因此需要创建一个数据适配器，同时也需要获取手机中的所有应用的数据，由于界面上的应用程序数据分为用户程序和系统程序，那么在数据适配器中，我们将根据下图 5-8 的分析进行对应位置数据的填充，具体代码如下所示。

图 5-8　ListView 中应用程序不同数据集合的展示

```
1.  private class AppManagerAdapter extends BaseAdapter {
2.          @Override
3.          public int getCount() {
4.              // 表示加上 2 个特殊条目
5.              return userLists.size() + 1 + systemLists.size() + 1;
6.          }
7.          @Override
8.          public Object getItem(int position) {
9.              return null;
10.         }
11.         @Override
12.         public long getItemId(int position) {
13.             return 0;
14.         }
15.         @Override
16.         public View getView(int position, View convertView, ViewGroup parent) {
17.             if (position == 0) {
18.                 TextView textView = new TextView(AppManagerActivity.this);
19.                 textView.setText("用户程序(" + userLists.size() + ")");
20.                 textView.setTextColor(Color.WHITE);
21.                 textView.setBackgroundColor(Color.GRAY);
22.                 return textView;
23.             }
24.             if(position == userLists.size() + 1){
25.                 TextView textView = new TextView(AppManagerActivity.this);
26.                 textView.setText("系统程序(" + systemLists.size() + ")");
27.                 textView.setTextColor(Color.WHITE);
28.                 textView.setBackgroundColor(Color.GRAY);
29.                 return textView;
30.             }
31.             View view = View.inflate(parent.getContext(),
32.                     R.layout.item_activity_manager, null);
33.             // 应用的名字
34.             TextView tv_app_name = (TextView) view
35.                     .findViewById(R.id.tv_app_name);
36.             // 应用的位置
37.             TextView tv_app_location = (TextView) view
38.                     .findViewById(R.id.tv_app_location);
39.             // 应用的图标
40.             ImageView iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
41.             // 应用的大小
42.             TextView tv_app_size = (TextView) view
43.                     .findViewById(R.id.tv_app_size);
```

```
44. //          AppInfo appInfo = mAppInfos.get(position);
45.             AppInfo appInfo;
46.             if(position < userLists.size() + 1){
47.                 //获取到所有的用户程序
48.                 appInfo = userLists.get(position - 1);
49.             }else{
50.                 //系统程序
51.                 int location = position - 1 - userLists.size() - 1;
52.                 appInfo = systemLists.get(location);
53.             }
54.             // 设置应用的名字
55.             tv_app_name.setText(appInfo.appName);
56.             // 设置图标的 icon
57.             iv_icon.setImageDrawable(appInfo.icon);
58.             // 设置应用的大小
59.             tv_app_size.setText(Formatter.formatFileSize(
60.                     AppManagerActivity.this, appInfo.appSize));
61.             return view;
62.         }
63.     }
```

- 第 3~6 行的 getCount()方法用于显示当前条目的个数，由于当前条目中添加了两个 TextView 用于显示系统程序和用户程序，因此个数应该为用户应用的个数加上系统应用的个数再加上 2，也就是 userLists.size( ) + systemLists.size( ) + 2。
- 第 17~30 行根据当前的 position 为 0 或者 userLists.size( ) + 1 时，判断当前条目是否为“用户程序”和“系统程序”的 TextView 标签，如果是则指定文本显示内容为用户程序或系统程序的个数。
- 第 31~43 行将 ListView 的条目布局加载进来，并获取布局上的控件对象。
- 第 45~53 行根据当前的 position 获取对应的 appInfo 对象，即用户程序和系统程序信息。
- 第 54~62 行根据获取的 appInfo 对象，将信息显示在界面控件上。

5.2.7 软件管家主逻辑

前面我们在获取数据集合的时候，当数据获取完毕后，会向主线程中创建的 Handle 发送消息，在 Handle 中进行数据的界面展示，关于 Handle 部分的代码如下所示。

```
1.  private Handler handler = new Handler() {
2.          public void handleMessage(android.os.Message msg) {
3.              // 隐藏进度条
4.              pb.setVisibility(View.INVISIBLE);
5.              AppManagerAdapter adapter = new AppManagerAdapter();
6.              mListview.setAdapter(adapter);
7.          };
8.      };
```

上面的代码书写好之后，运行程序，效果图如图 5-9 所示。

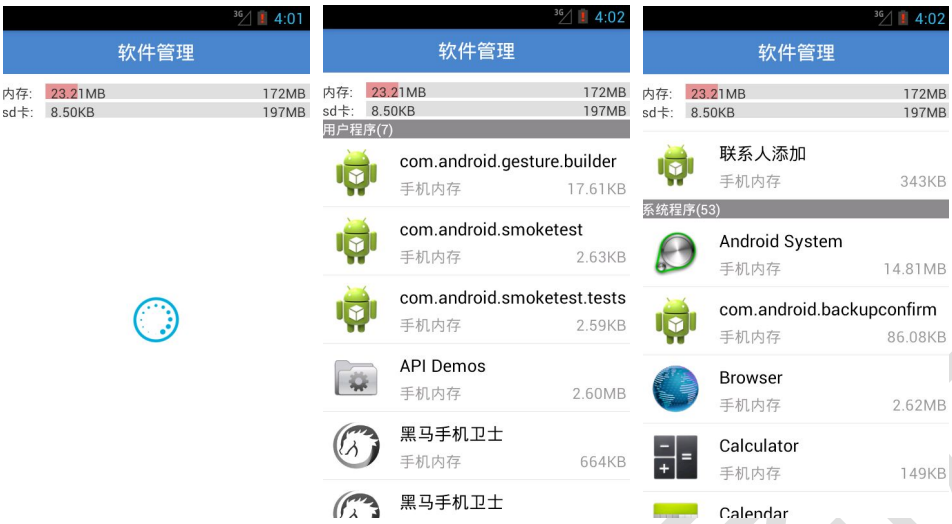


图 5-9 软件管家效果图

5.3 软件管家界面的逻辑优化

5.3.1 ListView 的数据适配器优化之一

（1） getItem( )方法使用

前面，我们是在 ListView 的数据适配器的 getView 方法中根据当前 position 而从数据集合中获取当前的 AppInfo 对象，这里我们考虑使用另外一种方式，那就是利用适配器中的 getItem( )方法，此方法有一个返回值，我们可以在此方法的逻辑中根据 position 的值返回对应的 AppInfo 对象，从而在 getView( )中调用。

正如前面所述，我们可以将 getView( )中根据位置返回 AppInfo 对象的逻辑放到 getItem( )中实现，具体适配器的修改代码如下所示。

```
1.  private class AppManagerAdapter extends BaseAdapter {
2.          @Override
3.          public int getCount() {
4.              // 表示加上 2 个特殊条目
5.              return userLists.size() + 1 + systemLists.size() + 1;
6.          }
7.          @Override
8.          public Object getItem(int position) {
9.              // 判断当前的位置是用户程序还是系统程序 2 个特殊条目
10.             if (position == 0 || position == userLists.size() + 1) {
11.                 return null;
12.             }
13.             AppInfo appInfo;
14.             if (position < userLists.size() + 1) {
15.                 appInfo = userLists.get(position - 1);
16.                 return appInfo;
17.             } else {
```

```
18.                 // 系统程序
19.                 int location = position - 1 - userLists.size() - 1;
20.                 appInfo = systemLists.get(location);
21.                 return appInfo;
22.             }
23.         }
24.         @Override
25.         public long getItemId(int position) {
26.             return 0;
27.         }
28.         @Override
29.         public View getView(int position, View convertView, ViewGroup parent) {
30.             if (position == 0) {
31.                 TextView textView = new TextView(AppManagerActivity.this);
32.                 textView.setText("用户程序(" + userLists.size() + ")");
33.                 textView.setTextColor(Color.WHITE);
34.                 textView.setBackgroundColor(Color.GRAY);
35.                 return textView;
36.             }
37.             if(position == userLists.size() + 1){
38.                 TextView textView = new TextView(AppManagerActivity.this);
39.                 textView.setText("系统程序(" + systemLists.size() + ")");
40.                 textView.setTextColor(Color.WHITE);
41.                 textView.setBackgroundColor(Color.GRAY);
42.                 return textView;
43.             }
44.             Appinfo appInfo = getItem(position);
45.             View view = View.inflate(parent.getContext(),
46.                     R.layout.item_activity_manager, null);
47.             // 应用的名字
48.             TextView tv_app_name = (TextView) view
49.                     .findViewById(R.id.tv_app_name);
50.             // 应用的位置
51.             TextView tv_app_location = (TextView) view
52.                     .findViewById(R.id.tv_app_location);
53.             // 应用的图标
54.             ImageView iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
55.             // 应用的大小
56.             TextView tv_app_size = (TextView) view
57.                     .findViewById(R.id.tv_app_size);
58.             // 设置应用的名字
59.             tv_app_name.setText(appInfo.appName);
60.             // 设置图标的 icon
61.             iv_icon.setImageDrawable(appInfo.icon);
```

```
62.             // 设置应用的大小
63.             tv_app_size.setText(Formatter.formatFileSize(
64.                     AppManagerActivity.this, appInfo.appSize));
65.             return view;
66.         }
67.     }
```

运行程序，效果图如图 5-10 所示，与前面效果一样。

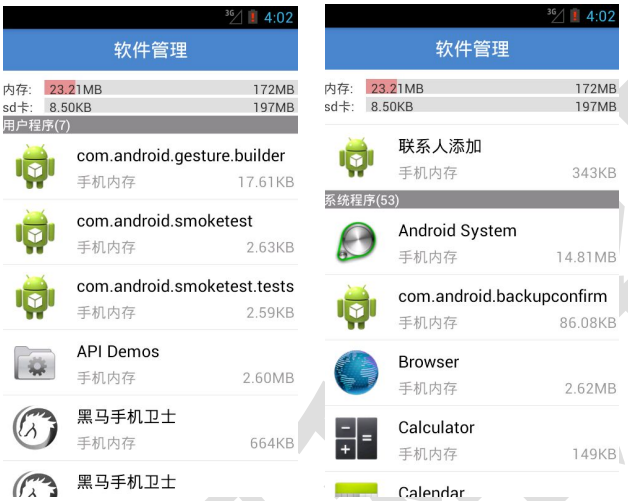


图 5-10 软件管理主界面

（2） getItemViewType( )和 getViewTypeCount 方法使用

常常我们可以看到，在 ListView 中显示的条目布局是不一样的，正如本项目中存在用户程序或者系统程序的条目布局，另外还存在应用程序的条目布局一样。这种不同布局的展示，前面我们是在 getView( ) 根据 position 是否为 0 和 userLists.size() + 1 来进行判定的，为了简化逻辑，我们在这里将使用适配器的另外两种方法来实现，即 getItemViewType( )和 getViewTypeCount 方法，前者是根据不同的 position 返回不同的数据类型，后者返回总共有几种数据类型。

观察下图我们项目的界面效果图可知，在我们的 ListView 的条目布局中，存在两种不同的布局类型，如图 5-11 所示。

图 5-11 软件管理主界面布局类型分析

想要通过上述的两个方法进行不同条目布局的展示，具体的代码如下所示。

```
1.  private class AppManagerAdapter extends BaseAdapter {
2.          @Override
3.          public int getCount() {
4.              // 表示加上 2 个特殊条目
5.              return userLists.size() + 1 + systemLists.size() + 1;
6.          }
7.          @Override
8.          public AppInfo getItem(int position) {
9.              // 判断当前的位置是用户程序还是系统程序 2 个特殊条目
10.             if (position == 0 || position == userLists.size() + 1) {
11.                 return null;
12.             }
13.             AppInfo appInfo;
14.             if (position < userLists.size() + 1) {
15.                 appInfo = userLists.get(position - 1);
16.                 return appInfo;
17.             } else {
18.                 // 系统程序
19.                 int location = position - 1 - userLists.size() - 1;
20.                 appInfo = systemLists.get(location);
21.                 return appInfo;
22.             }
23.         }
24.         @Override
25.         public long getItemId(int position) {
26.             // TODO Auto-generated method stub
27.             return position;
28.         }
29.         // 表示 2 种数据类型 数据类型必须从 0 开始
30.         @Override
31.         public int getItemViewType(int position) {
32.             // 特殊条目
33.             if (position == 0 || position == userLists.size() + 1) {
34.                 return 0;
35.             } else {
36.                 // 普通的条目
37.                 return 1;
38.             }
39.         }
40.         // 返回 2 种数据类型
41.         @Override
42.         public int getViewTypeCount() {
43.             // TODO Auto-generated method stub
```

```
44.             return 2;
45.         }
46.         @Override
47.         public View getView(int position, View convertView, ViewGroup parent) {
48.             View view = null;
49.             int type = getItemViewType(position);
50.             switch (type) {
51.             // 表示特殊类型
52.             case 0:
53.                  if (position == 0) {
54.                     TextView textView = new TextView(AppManagerActivity.this);
55.                     textView.setText("用户程序(" + userLists.size() + ")");
56.                     textView.setTextColor(Color.WHITE);
57.                     textView.setBackgroundColor(Color.GRAY);
58.                     return textView;
59.                  }
60.                  if (position == userLists.size() + 1) {
61.                     TextView textView = new TextView(AppManagerActivity.this);
62.                     textView.setText("系统程序(" + systemLists.size() + ")");
63.                     textView.setTextColor(Color.WHITE);
64.                     textView.setBackgroundColor(Color.GRAY);
65.                     return textView;
66.                  }
67.                 break;
68.             case 1:
68.                 Appinfo appInfo = getItem(position);
69.                  View view = View.inflate(parent.getContext(),
70.                           R.layout.item_activity_manager, null);
71.                 // 应用的名字
72.                 TextView tv_app_name = (TextView) view
73.                     .findViewById(R.id.tv_app_name);
74.                 // 应用的位置
75.                 TextView tv_app_location = (TextView) view
76.                     .findViewById(R.id.tv_app_location);
77.                 // 应用的图标
78.                 ImageView iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
79.                 // 应用的大小
80.                 TextView tv_app_size = (TextView) view
81.                     .findViewById(R.id.tv_app_size);
82.                 // 设置应用的名字
83.                 tv_app_name.setText(appInfo.appName);
84.                 // 设置图标的 icon
85.                 iv_icon.setImageDrawable(appInfo.icon);
```

```
86.                 // 设置应用的大小
87.                 tv_app_size.setText(Formatter.formatFileSize(
88.                     AppManagerActivity.this, appInfo.appSize));
89.                 return view;
69.         }
70.     }
71. }
```

5.3.2 ListView 的数据适配器优化之二

在 ListView 进行条目布局对象获取时，通常都是通过 findViewById 的方式获取 View 对象，但是这种方式是非常耗时的，为避免系统多次进行 findViewById 操作，这里我们使用定义的 ViewHolder 来保存这些 View 对象，当系统缓存为空时通过 findViewById 找到 ViewHolder 中的 View 对象，然后将当前的 ViewHolder 与缓存对象 convertView 进行绑定，当系统缓存对象不为空时从 convertView 中获取 ViewHolder 对象，考虑到本文中存在应用程序标题和详细信息两种类型的布局，这里将定义两个 ViewHolder。

根据上述分析，对应修改后的适配器代码如下所示。

```
1.  private class AppManagerAdapter extends BaseAdapter {
2.          @Override
3.          public int getCount() {
4.              // 表示加上 2 个特殊条目
5.              return userLists.size() + 1 + systemLists.size() + 1;
6.          }
7.          @Override
8.          public AppInfo getItem(int position) {
9.              // 判断当前的位置是用户程序还是系统程序 2 个特殊条目
10.             if (position == 0 || position == userLists.size() + 1) {
11.                 return null;
12.             }
13.             AppInfo appInfo;
14.             if (position < userLists.size() + 1) {
15.                 appInfo = userLists.get(position - 1);
16.                 return appInfo;
17.             } else {
18.                 // 系统程序
19.                 int location = position - 1 - userLists.size() - 1;
20.                 appInfo = systemLists.get(location);
21.                 return appInfo;
22.             }
23.         }
24.         @Override
25.         public long getItemId(int position) {
26.             // TODO Auto-generated method stub
27.             return position;
```

```
28.         }
29.         /**
30.          * 表示 2 种数据类型 数据类型必须从 0 开始
31.          */
32.         @Override
33.         public int getItemViewType(int position) {
34.             // 特殊条目
35.             if (position == 0 || position == userLists.size() + 1) {
36.                 return 0;
37.             } else {
38.                 // 普通的条目
39.                 return 1;
40.             }
41.             // return super.getItemViewType(position);
42.         }
43.         /**
44.          * 返回 2 种数据类型
45.          */
46.         @Override
47.         public int getViewTypeCount() {
48.             // TODO Auto-generated method stub
49.             return 2;
50.         }
51.         @Override
52.         public View getView(int position, View convertView, ViewGroup parent) {
53.             View view = null;
54.             // 特殊条目的 holder
55.             HeadViewHolder headViewHolder = null;
56.             // 普通条目的 holder
57.             ViewHolder viewHolder = null;
58.             // 获取到 item 数据类型
59.             int type = getItemViewType(position);
60.             switch (type) {
61.             // 表示特殊类型
62.             case 0:
63.                 if (position == 0 || position == userLists.size() + 1) {
64.                     if (convertView == null) {
65.                         convertView = View.inflate(AppManagerActivity.this,
66.                                 R.layout.item_app_manager_head_view, null);
67.                         headViewHolder = new HeadViewHolder();
68.                         headViewHolder.tv_head_view_title = (TextView) convertView
69.                                 .findViewById(R.id.tv_head_view_title);
70.                         convertView.setTag(headViewHolder);
```

```
71.                     } else {
72.                         headViewHolder = (HeadViewHolder) convertView.getTag();
73.                     }
74.                     if (position == 0) {
75.                         headViewHolder.tv_head_view_title.setText("用户程序("
76.                                 + userLists.size() + ")");
77.                     } else {
78.                         headViewHolder.tv_head_view_title.setText("系统程序("
79.                                 + systemLists.size() + ")");
80.                     }
81.                 }
82.                 break;
83.             case 1:
84.                 AppInfo appInfo = getItem(position);
85.                 if (convertView == null) {
86.                     convertView = View.inflate(parent.getContext(),
87.                             R.layout.item_activity_manager, null);
88.                     viewHolder = new ViewHolder();
89.                     // 应用的名字
90.                     viewHolder.tv_app_name = (TextView) convertView
91.                             .findViewById(R.id.tv_app_name);
92.                     // 应用的位置
93.                     viewHolder.tv_app_location = (TextView) convertView
94.                             .findViewById(R.id.tv_app_location);
95.                     // 应用的图标
96.                     viewHolder.iv_icon = (ImageView) convertView
97.                             .findViewById(R.id.iv_icon);
98.                     // 应用的大小
99.                     viewHolder.tv_app_size = (TextView) convertView
100.                            .findViewById(R.id.tv_app_size);
101.                    convertView.setTag(viewHolder);
102.                } else {
103.                    viewHolder = (ViewHolder) convertView.getTag();
104.                }
105.                // 设置应用的名字
106.                viewHolder.tv_app_name.setText(appInfo.appName);
107.                // 设置图标的 icon
108.                viewHolder.iv_icon.setImageDrawable(appInfo.icon);
109.                // 设置应用的大小
110.                viewHolder.tv_app_size.setText(Formatter.formatFileSize(
111.                        AppManagerActivity.this, appInfo.appSize));
112.                // 判断当前 app 是否安装到 sd 卡还是机身内存
113.                if (appInfo.isRom) {
114.                    // 表示机身内存
```

```
                        viewHolder.tv_app_location.setText("手机内存");
                    } else {
                        // 表示 SD 卡内存
                        viewHolder.tv_app_location.setText("SD 卡");
                    }
                    break;
                }
                return convertView;
            }
        }
        /**
         * 头部分的标题
         */
        static class HeadViewHolder {
            // 头标题
            TextView tv_head_view_title;
        }
        /**
         * 普通的 holder
         */
        static class ViewHolder {
            TextView tv_app_name;
            TextView tv_app_location;
            ImageView iv_icon;
            TextView tv_app_size;
        }
```

上述代码中，当 item 数据类型为 0 时，即为头标题，它的布局文件为 item_app_manager_head_view.xml，该布局中只有一个 TextView，具体代码如文件【5-7】所示。

【文件 5-7】 res/item_app_manager_head_view.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/tv_head_view_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="5dp"
        android:textColor="#fff"
        android:background="@android:color/darker_gray"/>
</LinearLayout>
```

ListView 的适配器代码修改好之后，运行程序，效果图如图 5-12 所示，与之前效果一样。

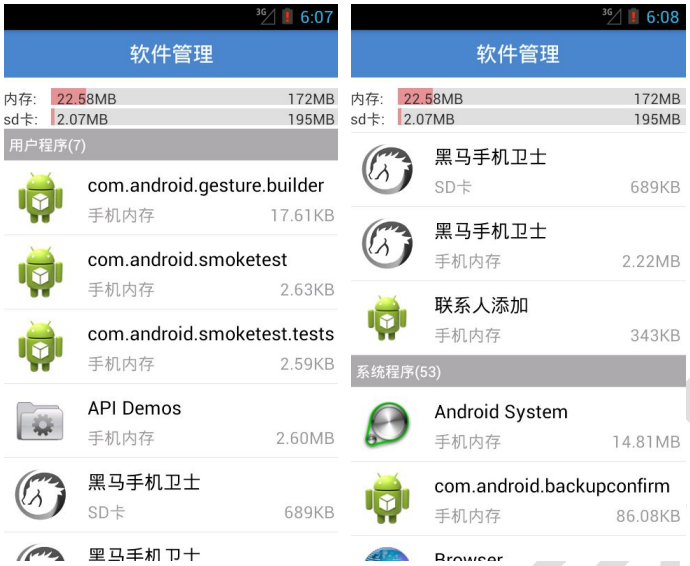


图 5-12 软件管家界面效果

5.3.3 ListView 的滚动事件

根据完整效果图如图 5-13 所示，ListView 上方的标题的固定的，并且能够当 ListView 滚动到系统程序到顶部时，顶部的标题会变成系统程序的字样。

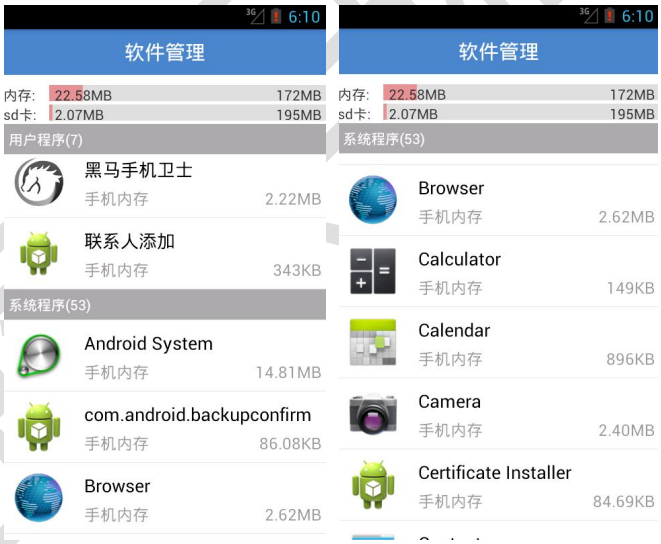


图 5-13 软件管理界面完整效果

首先，我们需要在当前 ListView 的布局之上添加一个头标题，因为包裹 ListView 的是帧布局，默认左上角对齐，那么修改的布局代码如下所示。

```
1.  <FrameLayout
2.          android:layout_width="match_parent"
3.          android:layout_height="match_parent" >
4.          <ListView
5.              android:id="@+id/list_view"
6.              android:layout_width="match_parent"
7.              android:layout_height="wrap_content" >
```

```
        </ListView>
        <TextView
           android:id="@+id/tv_head_title"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:background="@android:color/darker_gray"
           android:padding="5dp"
           android:text="用户程序(11)"
           android:textColor="#fff" />
        <ProgressBar
           android:id="@+id/pb"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_gravity="center"
           android:indeterminateDrawable="@drawable/progress_medium"
           android:visibility="invisible" />
    </FrameLayout>
```

此外，我们需要设置 ListView 的滚动监听事件，判断当前 ListView 条目的显示情况，根据第一个可视的位置，判断其是用户程序或者系统程序来对应地修改头标题中的显示信息，这部分逻辑在软件管理主逻辑代码中进行修改，如下所示。

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        initView();
        initData();
        initListener();
    }
        // listview 的滚动监听
    mListview.setOnScrollListener(new OnScrollListener() {
            @Override
            public void onScrollStateChanged(AbsListView view, int scrollState) {
                // TODO Auto-generated method stub
            }
            /**
             * 第一个参数：listview        第二个参数：第一条可见的条目
               第三个参数：第一页可见的条目    第四个参数：总共有多少个条目
             */
            @Override
            public void onScroll(AbsListView view, int firstVisibleItem,
                    int visibleItemCount, int totalItemCount) {
                // 获取到 listview 第一个可见的条目
                // int firstVisiblePosition =
```

```
24.                 // mListview.getFirstVisiblePosition();
25.                 System.out.println("firstVisibleItem---" + firstVisibleItem);
26.                 // System.out.println("visibleItemCount---" + visibleItemCount);
27.                 // System.out.println("totalItemCount---" + totalItemCount);
28.                 // 判断程序是否有值
29.                 if (userLists != null && systemLists != null) {
30.                     if (firstVisibleItem > userLists.size() + 1) {
31.                         // 展示系统程序
32.                         tv_head_title.setText("系统程序(" + systemLists.size()
33.                                 + ")");
34.                     } else {
35.                         // 展示用户程序
36.                         tv_head_title.setText("用户程序(" + userLists.size() + ")");
37.                     }
38.                 }
39.             }
40.         });
```

代码修改好之后，运行程序，效果图如图 5-14 所示。

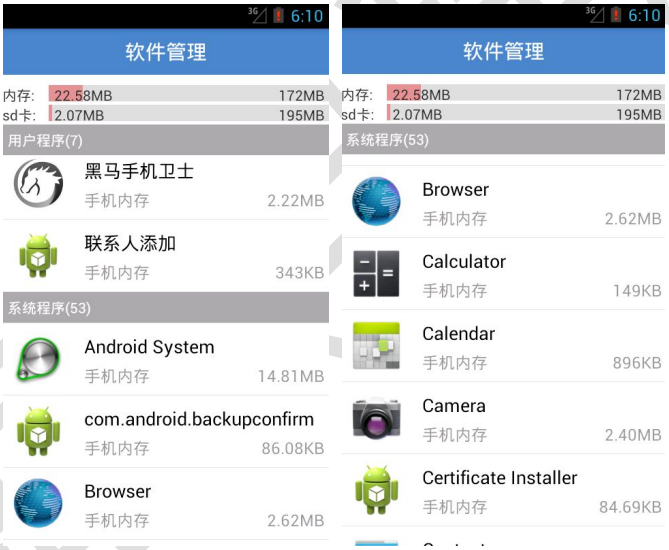


图 5-14 软件管理界面效果图

5.3.4 ListView 条目的小气泡

(1) 小气泡的布局

当选中 ListView 的某一个应用程序时，程序的下方会浮出一个小气泡，提供卸载、运行、分享三个选项可以对程序进行操作，该条目的布局如图 6-15 所示。

图 5-15 小气泡布局效果图

图 5-15 中小气泡对应的布局文件如【文件 5-8】所示。

【文件 5-8】 res/layout/item_appmanager_list.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:background="@drawable/local_popup_bg"
6.      android:orientation="horizontal" >
7.      <TextView
8.          android:id="@+id/tv_unintall"
9.          android:layout_width="wrap_content"
10.         android:layout_height="wrap_content"
11.         android:layout_marginBottom="5dp"
12.         android:layout_marginLeft="15dp"
13.         android:layout_marginRight="15dp"
14.         android:layout_marginTop="5dp"
15.         android:drawableTop="@drawable/img1"
16.         android:text="卸载" />
17.     <View
18.         android:layout_width="0.5dp"
19.         android:layout_height="20dp"
20.         android:background="@drawable/local_popup_divider" />
21.     <TextView
22.         android:id="@+id/tv_run"
23.         android:layout_width="wrap_content"
24.         android:layout_height="wrap_content"
25.         android:layout_marginBottom="5dp"
26.         android:layout_marginLeft="15dp"
27.         android:layout_marginRight="15dp"
28.         android:layout_marginTop="5dp"
29.         android:drawableTop="@drawable/img2"
30.         android:text="运行" />
31.     <View
32.         android:layout_width="0.5dp"
33.         android:layout_height="20dp"
```

```
34.         android:background="@drawable/local_popup_divider" />
35.     <TextView
36.         android:id="@+id/tv_share"
37.         android:layout_width="wrap_content"
38.         android:layout_height="wrap_content"
39.         android:layout_marginBottom="5dp"
40.         android:layout_marginLeft="15dp"
41.         android:layout_marginRight="15dp"
42.         android:layout_marginTop="5dp"
43.         android:drawableTop="@drawable/img3"
44.         android:text="分享" />
45. </LinearLayout>
```

上述布局文件中，主要使用了一个线性布局，线性布局中放置了 3 个 TextView 控件和 2 个 View 控件，分别用于展示这三个按钮的文字提示、功能图标以及功能图标左侧分割线。此外，线性布局的背景使用一个点 9 图，可以进行横向纵向的拉伸效果，

(2) ListView 的点击事件

当点击 ListView 的条目时，会弹出小气泡，因此需要监听其点击事件，该代码需要写在 initListener()中，具体的点击事件代码如下所示。

```
1.  mListview.setOnItemClickListener(new OnItemClickListener() {
2.              /**
3.               * 第一个参数：listview      第二个参数：表示孩子的根节点
4.                 第三个参数：表示位置        第四个参数：表示应用程序 id
5.               */
6.              @Override
7.              public void onItemClick(AdapterView<?> parent, View view,
8.                      int position, long id) {
9.                  // 获取到点击对象
10.                 //如果用户点击的是特殊条目。直接跳出
11.                 if(position == 0 || position == userLists.size() + 1){
12.                     return ;
13.                 }
14.                 mClickAppInfo = (AppInfo) mListview.getItemAtPosition(position);
15.                 // 展示气泡控件
16.                 showPopupWindow(view, parent);
17.             }
18.         });
```

(3) 小气泡的逻辑实现

前面我们已经将小气泡的布局文件已写好，这里将要实现小气泡的弹出逻辑代码，即 showPopupWindow()方法，具体如下所示。

```
1.     /**
2.       * 展示气泡控件
3.       *
```

```
4.       * @param view
5.       * @param parent
6.       */
7.
8.      protected void showPopupWindow(View view, AdapterView<?> parent) {
9.          View contentView = View.inflate(this, R.layout.item_popup_window, null);
10.         // 初始化气泡控件
11.         // 第一个参数:需要展示的 view
12.         popupWindow = new PopupWindow(contentView, LayoutParams.WRAP_CONTENT,
13.                 LayoutParams.WRAP_CONTENT);
14.
15.         // side 旁边 点击旁边让小气泡消失
16.         popupWindow.setOutsideTouchable(true);
17.         // 让小气泡获取焦点
18.         popupWindow.setFocusable(true);
19.         // 设置小气泡的背景。是一个透明色
20.         popupWindow.setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
21.
22.         // 第一个参数：展示 在某某控件的下方
23.         // 第二个参数：表示 x 轴的偏移量
24.         // 第三个参数：表示 y 轴的偏移量
25.         popupWindow.showAsDropDown(view, 80, -view.getHeight());
26.         // 展示在某某位置
27.         // popupWindow.showAtLocation(parent, Gravity.LEFT | Gravity.TOP, 100,
28.         // 100);
29.         // int[] arrayOfInt = new int[2];
30.         // view.getLocationInWindow(arrayOfInt);
31.         // popupWindow.showAtLocation(view, 51, 60 + arrayOfInt[0], arrayOfInt[1]);
32.     }
33.
34.     @Override
35.     protected void onDestroy() {
36.         // TODO Auto-generated method stub
37.         super.onDestroy();
38.         // 如果小气泡已经初始化。那么让它消失
39.         if (null != popupWindow) {
40.             popupWindow.dismiss();
41.             popupWindow = null;
42.         }
43.     }
```

上述代码写好之后，运行程序，效果图如图 5-16 所示，发现小气泡能正常弹出。

图 5-16 小气泡效果图

(4) 小气泡中的逻辑功能实现

由上可知，小气泡中有三个功能，即卸载、运行、分享功能，这里我们将要实现这三个功能，对应地就是要实现其点击事件，为了代码更加简洁，这里让当前 AppManagerActivity 实现 OnClickListener 接口，然后重写 onClick()方法，在该方法中根据对应的 id 进行相应的逻辑处理。

首先，需要在 showPopupWindow()方法中获取对应的 view 对象并设置点击事件，代码如下所示。

```
1.          // 卸载
2.          TextView tv_unintall = (TextView) contentView.findViewById(R.id.tv_unintall);
3.          // 运行
4.          TextView tv_run = (TextView) contentView.findViewById(R.id.tv_run);
5.          // 分享
6.          TextView tv_share = (TextView) contentView.findViewById(R.id.tv_share);
7.          tv_unintall.setOnClickListener(this);
8.          tv_run.setOnClickListener(this);
9.          tv_share.setOnClickListener(this);
```

然后重写 onClick()方法，代码如下所示。

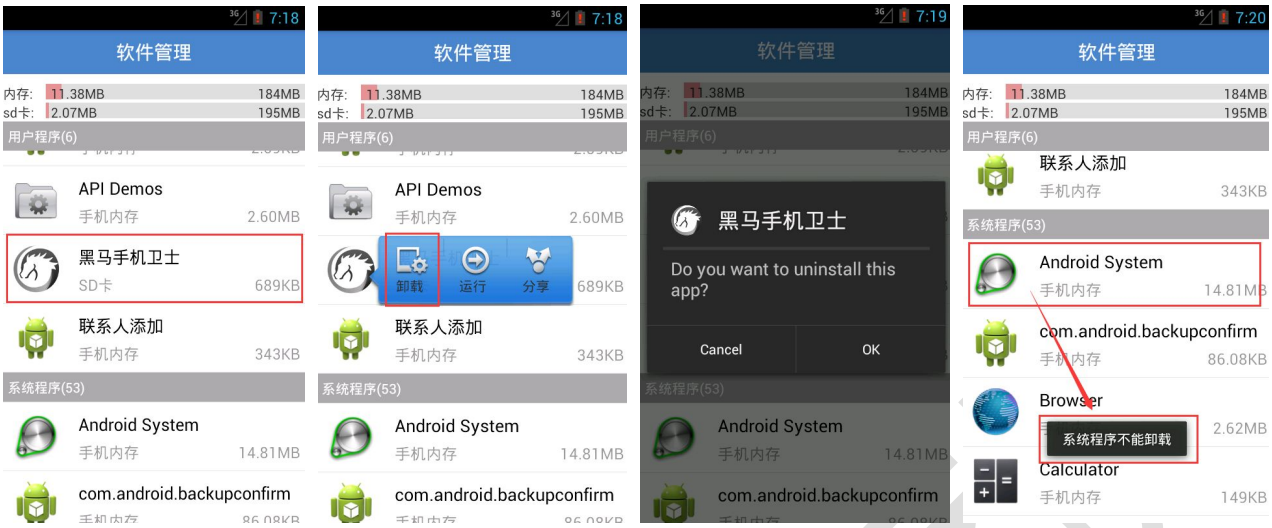
```
1.  @Override
2.      public void onClick(View v) {
3.          switch (v.getId()) {
4.          case R.id.tv_unintall:
5.              // ToastUtils.showSafeToast(this, "卸载");
6.              initUnInstall();
7.              break;
8.
9.          case R.id.tv_run:
10.             // ToastUtils.showSafeToast(this, "运行");
11.             initRun();
```

```
12.             break;
13.         case R.id.tv_share:
14.             // ToastUtils.showSafeToast(this, "分享");
15.             initShare();
16.             break;
17.         }
18.         // 关闭小气泡
19.         popupWindowDismiss();
20.     }
21.
22.     /**
23.      * 点击任意一个功能后，都关闭小气泡
24.      */
25.     private void popupWindowDismiss() {
26.         if (null != popupWindow) {
27.             popupWindow.dismiss();
28.             popupWindow = null;
29.         }
30.
31.     }
```

首先，initUnInstall()方法是用于卸载当前软件的方法，具体的代码逻辑如下所示。

```
1.      /**
2.       * 卸载
3.       */
4.      private void initUnInstall() {
5.
6.          if (mClickAppInfo.isUserApp) {
7.              Intent localIntent = new Intent("android.intent.action.DELETE",
8.                      Uri.parse("package:" + mClickAppInfo.appPackageName));
9.              startActivity(localIntent);
10.         } else {
11.             // root 手机才能卸载
12.             ToastUtils.showSafeToast(this, "系统程序不能卸载");
13.         }
14.     }
```

运行程序，效果图如下所示，当卸载 SD 内存中的应用如黑马手机卫士时可以正常卸载，如图 5-17（a）—（c），但是当卸手机内存中的应用时就会弹出土司，指示当前程序不可卸载，需要 root 之后才可以，如下图 5-17（d）所示。



（a） （b） （c） （d）

图 5-17 小气泡卸载功能展示效果图

第二个 initRun()方法是用于启动当前应用的，它的逻辑代码如下所示。

```
1.    /**
2.       * 运行 */
3.      private void initRun() {
4.          Intent localIntent = AppManagerActivity.this.getPackageManager()
5.                  .getLaunchIntentForPackage(mClickAppInfo.appPackageName);
6.          if (null != localIntent) {
7.              startActivity(localIntent);
8.          } else {
9.              ToastUtils.showSafeToast(this, "当前应用程序没有入口");
10.         }
11.     }
```

运行程序，点击运行模拟器中的另外一个程序，效果图如图 5-18 所示。

图 5-18 小气泡运行功能展示效果图

第三个 initShare()方法是用于分享当前应用程序信息的，具体代码如下所示。

```
/**分享 https://play.google.com/store/apps/details?id=com.tencent.mobileqq
 */
private void initShare() {
    // PackageInfo localPackageInfo = this.val$_PackageInfo;
    Intent localIntent = new Intent("android.intent.action.SEND");
    localIntent.setType("text/plain");
    localIntent.putExtra("android.intent.extra.SUBJECT", "f 分享");
    localIntent.putExtra("android.intent.extra.TEXT", "Hi！推荐您使用软件："
            + mClickAppInfo.appName + "下载地址:"
            + "https://play.google.com/store/apps/details?id="
            + mClickAppInfo.appPackageName);
    startActivity(Intent.createChooser(localIntent, "分享"));
}
```

运行程序，点击小气泡中的分享功能，效果图如图 5-19 所示。

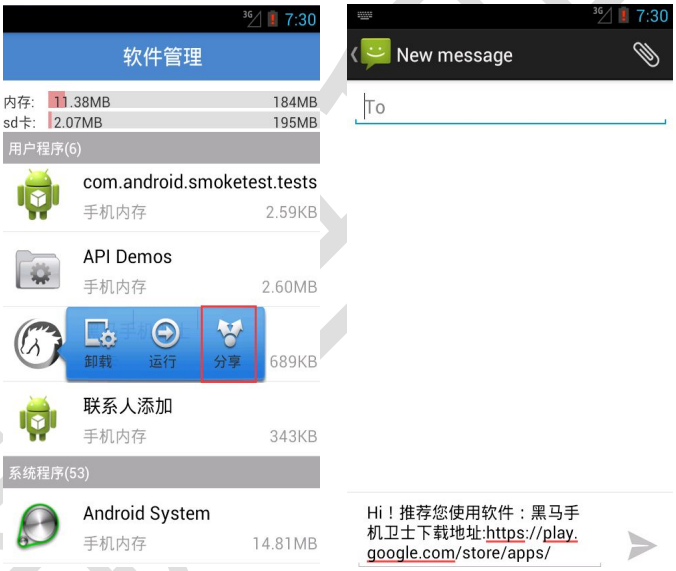


图 5-19 小气泡分享功能展示效果图

(5) 小气泡的动画

这里为了小气泡的效果更加美观，给它的进入和弹出都添加一个动画效果，在 showPopupWindow( ) 中进行修改。

首先，在 showPopupWindow( )中添加以下代码。

```
// 设置动画
popupWindow.setAnimationStyle(R.style.PopupAnimationDialog);
```

其中，PopupAnimationDialog 的动画样式是在 styles.xml 中定义的，具体代码如下所示。

```
<!--    小气泡动画 -->
<style name="PopupAnimationDialog" parent="@android:style/Animation.Dialog">
    <item name="android:windowEnterAnimation">@anim/popup_method_enter</item>
    <item name="android:windowExitAnimation">@anim/popup_method_exit</item>
</style>
```

上面的 popup_method_enter 是小气泡进来时的动画效果，具体代码如文件【5-9】所示。

【文件 5-9】res/anim/popup_method_enter.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false" >
    <translate
        android:duration="@android:integer/config_shortAnimTime"
        android:fromXDelta="100%p"
        android:toXDelta="10%" />
    <alpha
        android:duration="@android:integer/config_shortAnimTime"
        android:fromAlpha="0.5"
        android:interpolator="@interpolator/decelerate_cubic"
        android:toAlpha="1.0" />
</set>
```

上面的 popup_method_exit 是小气泡退出时的动画效果，具体代码如文件【5-10】所示。

【文件 5-10】res/anim/popup_method_exit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false" >
    <translate
        android:duration="@android:integer/config_shortAnimTime"
        android:fromXDelta="10%"
        android:toXDelta="100%p" />
    <alpha
        android:duration="@android:integer/config_shortAnimTime"
        android:fromAlpha="1.0"
        android:interpolator="@interpolator/accelerate_cubic"
        android:toAlpha="0.0" />
</set>
```

5.4 常用工具之短信备份

5.4.1 常用工具界面

这里，我们开始讲述常用工具中的短信备份功能，此时常用工具的界面布局如下所示。

```
   <?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
```

```
8.          style="@style/TitleBarTextView"
9.          android:text="常用工具" />
10.
11.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
12.         android:id="@id/rl_lock_pattern_view"
13.         android:layout_width="match_parent"
14.         android:layout_height="wrap_content"
15.         android:layout_marginTop="10dp"
16.         itheima:isToggle="true"
17.         itheima:itbackground="first"
18.         itheima:title="图案密码" />
19.
20.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
21.         android:id="@+id/rl_number_query"
22.         android:layout_width="match_parent"
23.         android:layout_height="wrap_content"
24.         itheima:isToggle="true"
25.         itheima:itbackground="last"
26.         itheima:title="查询归属地" />
27.
28. <com.itheima.mobilesafe_sh2.act.view.SettingItemView
29.         android:id="@+id/siv_sms_backup"
30.         android:layout_width="match_parent"
31.         android:layout_height="wrap_content"
32.         android:layout_marginTop="10dp"
33.         itheima:isToggle="true"
34.         itheima:itbackground="first"
35.         itheima:title="短信备份" />
36.
37.     <ProgressBar
38.         android:id="@+id/progressBar1"
39.         style="?android:attr/progressBarStyleHorizontal"
40.         android:layout_width="match_parent"
41.         android:layout_height="wrap_content" />
42. </LinearLayout>
```

运行程序，效果图如图 5-20 所示，上面的 ProgressBar 是用于显示短信备份进度的。

图 5-20 常用工具短信备份效果图

5.4.2 短信备份工具类

要将短信备份在本地（内存或 SD 卡）有多种方法，在这里使用 XML 的形式保存短信内容。Android 手机中的短信是在 data/data/com.android.provider.telephony 应用 database 目录下的 mmssms.db 数据库中，如图 5-21 所示。

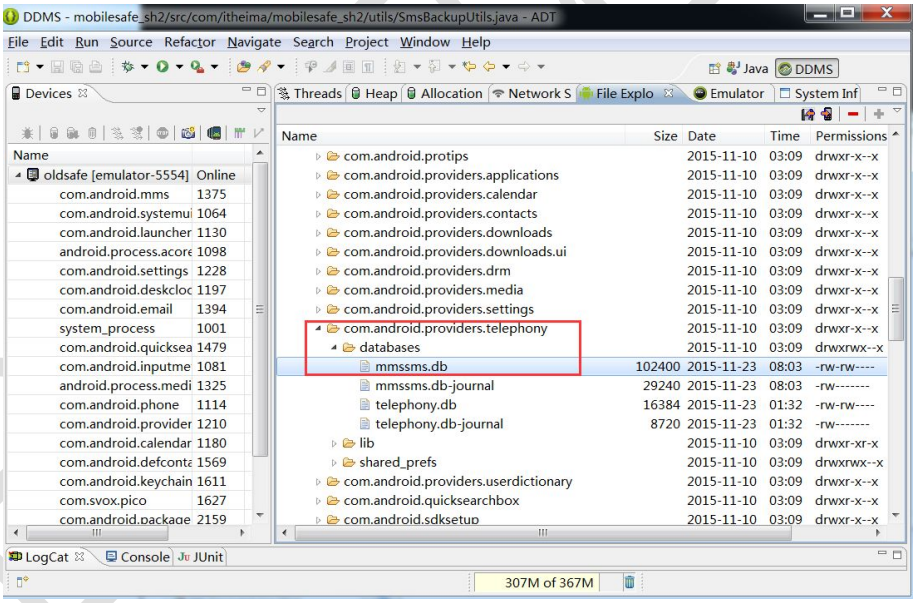


图 5-21 mmssms.db 数据库

通过 SQLiteExpert 工具打开 mmssms.db 数据库中的 sms 表，可以看到手机中的短信内容，具体如图 5-22 所示。

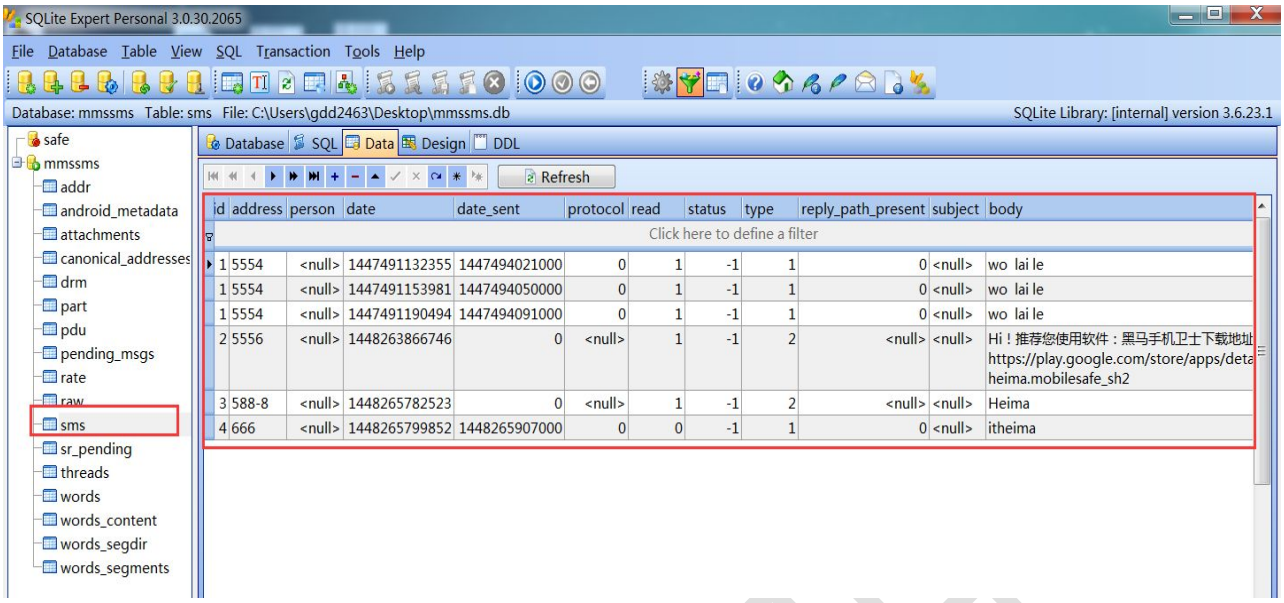


图 5-22　mmssms.db-sms

在图 5-22 的 sms 表中，只需要关心其中的几项数据即可，如 address 代表短信地址；type 代表短信类型：2 代表发送，1 代表接收；date 代表短信时间；body 代表短信内容。要将短信备份为 XML 文件，只需将这几项数据以节点形式写入 XML 文件即可。

在获取短信内容时，不直接操作数据库，而是使用 ContentResolver 得到短信数据库中的这几项数据，具体代码如【文件 5-11】所示。

【文件 5-11】 SmsBackUpUtils.java

```
1.  public class SmsBackupUtils {
2.      private static FileOutputStream fos;
3.  //   public interface OnClickListener {
4.  //         /**
5.  //          * Called when a view has been clicked.
6.  //          *
7.  //          * @param v The view that was clicked.
8.  //          */
9.  //         void onClick(View v);
10. //      }
11.     public interface OnSmsListener{
12.         //在短信备份之前需要设置最大值
13.         void beforSms(int count);
14.         //在短信备份中需要设置进度
15.         void onSms(int progress);
16.     }
17.     /**
18.      * 短信备份
19.      * @param context
20.      * @param dialog
21.      * @param progressBar1
```

```
22.      * @return
23.      */
24.     public static boolean backupSms(Context context,OnSmsListener onSmsListener) {
25.         // 把短信备份到 sd 卡
26.         // 判断当前用户是否有 sd 卡
27.         if (Environment.getExternalStorageState().equals(
28.                 Environment.MEDIA_MOUNTED)) {
29.             try {
30.                 // 初始化 xml 的序列化器
31.                 XmlSerializer serializer = Xml.newSerializer();
32.                 File file = new File(Environment.getExternalStorageDirectory(),
33.                         "backup.xml");
34.                 fos = new FileOutputStream(file);
35.                 ContentResolver resolver = context.getContentResolver();
36.                 Uri uri = Uri.parse("content://sms/");
37.                 Cursor cursor = resolver.query(uri, new String[] { "address",
38.                         "date", "type", "body" }, null, null, null);
39.                 serializer.setOutput(fos, "utf-8");
40.                 // 设置开始的节点
41.                 serializer.startDocument("utf-8", true);
42.                 serializer.startTag(null, "smss");
43.                 //表示一共有多少条短信
44.                 int count = cursor.getCount();
45.                 //设置进度条的最大值
46. //              dialog.setMax(count);
47. //              progressBar1.setMax(count);
48.                 //短信备份之前
49.                 onSmsListener.beforSms(count);
50.               //初始化进度
51.                 int progress = 0;
52.                 while (cursor.moveToNext()) {
53.                     serializer.startTag(null, "sms");
54.                     String address = cursor.getString(0);
55.                     String date = cursor.getString(1);
56.                     String type = cursor.getString(2);
57.                     String body = cursor.getString(3);
58.                     serializer.startTag(null, "address");
59.                     serializer.text(cursor.getString(0));
60.                     serializer.endTag(null, "address");
61.                     serializer.startTag(null, "date");
62.                     serializer.text(cursor.getString(1));
63.                     serializer.endTag(null, "date");
64.                     serializer.startTag(null, "type");
65.                     serializer.text(cursor.getString(2));
```

```
66.                     serializer.endTag(null, "type");
67.                     serializer.startTag(null, "body");
68.                     serializer.text(cursor.getString(3));
69.                     serializer.endTag(null, "body");
70.                     serializer.endTag(null, "sms");
71.                     System.out.println("----------------------");
72.                     System.out.println("address---" + address);
73.                     System.out.println("date---" + date);
74.                     System.out.println("type---" + type);
75.                     System.out.println("body---" + body);
76.                     progress++;
77.                     //设置进度条
78. //                  dialog.setProgress(progress);
79. //                  progressBar1.setProgress(progress);
80.                     onSmsListener.onSms(progress);
81.                     SystemClock.sleep(50);
82.                 }
83.                 serializer.endTag(null, "smss");
84.                 serializer.endDocument();
85.                 cursor.close();
86.                 fos.flush();
87.                 fos.close();
88.                 return true;
89.             } catch (Exception e) {
90.                 // TODO Auto-generated catch block
91.                 e.printStackTrace();
92.             }
93.         }
94.         return false;
95.     }
96. }
```

- 第 11~16 行代码定义一个回调接口 OnSmsListener，接口中定义两个方法，分别在短信备份之前调用和短信备份过程中调用。其中 beforSms()方法在得到短信总条数的时候进行设置，参数为接收短信的总条数，该方法用于主界面调用时进行是否备份短信的判断，如果总条数是 0 则不备份；onSms()方法在短信备份过程中进行设置，参数接收备份短信的进度，该方法用于主界面调用时的进度条的设置。
- 第 24~95 行的 backupSms()方法用于备份短信，首先创建一个备份文件 backup.xml，然后获取到短信内容，并对其进行序列化，将短信内容存储在 backup.xml 文件中。其中第 35~38 行代码使用 ContentResolver 得到短信数据。

5.4.3 短信备份主逻辑

当短信备份功能所使用的工具类和布局都定义好之后，下面开始编写短信备份主逻辑代码，主要控制短信的备份和备份进度。

首先，需要在 AToolsActivity.java 中获取短信备份的 View 对象，如下所示。

```
1.  // 短信备份
2.  SettingItemView siv_sms_backup = (SettingItemView) findViewById(R.id.siv_sms_backup);
```

然后在 onclick( )点击事件中进行逻辑处理，具体代码如【文件 5-12】所示。

【文件 5-12】 AToolsActivity.java 中的部分代码

```
1.  case R.id.siv_sms_backup:
2.              dialog = new ProgressDialog(this);
3.              dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
4.              dialog.setTitle("提示");
5.              dialog.setMessage("稍安勿躁.短信备份中...");
6.              dialog.setIcon(R.drawable.ic_launcher);
7.              new Thread(){
8.                  public void run() {
9.                      boolean result = SmsBackupUtils.backupSms(AToolsActivity.this,new
10.                     OnSmsListener() {
11.                         @Override
12.                         public void onSms(int progress) {
13.                             progressBar1.setProgress(progress);
14.                             dialog.setProgress(progress);
15.                         }
16.                         @Override
17.                         public void beforSms(int count) {
18.                             progressBar1.setMax(count);
19.                             dialog.setMax(count);
20.                         }
21.                     });
22.                     if(result){
23.                         //展示一个安全的土司
24.                         ToastUtils.showSafeToast(AToolsActivity.this, "短信备份成功");
25.                     }else{
26.                         ToastUtils.showSafeToast(AToolsActivity.this, "短信备份失败");
27.                     }
28.                     dialog.dismiss();
29.                 };
30.             }.start();
31.             dialog.show();
32.             break;
```

- 第 2~6 行的用于获取对话框，并进行设置标题和图标。
- 第 7~32 行代码在开启子线程中调用短信备份工具类 smsBackUpUtils 的 backUpSms()方法开始进

行短信备份。并实现该类中定义回调接口的两个方法，其中在 onSms()方法中进行进度条的设置，在 beforSms()方法中根据参数中的短信条数进行设置最大刻度。需要注意的是，result 是备份成功的 boolean 值，只要当 result 为 true 时表示短信备份操作成功，并会弹出一个短信备份成功的吐司，反之就会弹出一个短信备份失败的土司。

运行程序，效果图如图 5-23 所示。

图 5-23 短信备份操作

至此短信备份的功能全部完成，在高级工具模块点击短信备份按钮，这时会弹出一个对话框，当显示短信备份成功时。此时打开 DDMS，点击 File Explorer 打开文件浏览器，然后打开 SD 卡找到 backup.xml 文件说明短信备份已经成功。打开备份文件，内容如下图所 5-24 所示。

backup.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<smss>
    <sms>
        <address>666</address>
        <date>1448265799852</date>
        <type>1</type>
        <body>itheima</body>
    </sms>
    <sms>
        <address>588-8</address>
        <date>1448265782523</date>
        <type>2</type>
        <body>Heima </body>
    </sms>
    <sms>
        <address>5556</address>
        <date>1448263866746</date>
        <type>2</type>
        <body>Hi! 推荐您使用软件：黑马手机卫士下载地址:https://play.google.com/store/apps/details?id=com.itheima.mobilesafe_sh2</body>
        </sms>
    <sms>
        <address>5554</address>
        <date>1447491190494</date>
        <type>1</type>
        <body>wo  lai le  </body>
    </sms>
    <sms>
        <address>5554</address>
        <date>1447491153981</date>
        <type>1</type>
        <body>wo  lai le  </body>
    </sms>
    <sms>
        <address>5554</address>
        <date>1447491132355</date>
        <type>1</type>
        <body>wo  lai le  </body>
    </sms>
</smss>
```

图 5-24 短信备份成功后的文件内容

5.5 本章小结

本章主要是针对手机卫士中的软件管家模块进行讲解，首先针对该模块功能、代码结构进行介绍，然后讲解了软件管家的 UI 界面，最后实现软件管家中的逻辑代码。通过该模块可以学习到 ListView 的优化、怎样获取应用程序信息。该模块功能较为简单，很容易理解，编程者只需要按照步骤完成模块的开发即可。