第 6 章　进程管理

第 6 章 进程管理模块 3
6.1 模块概述 3
6.1.1 功能介绍 3
6.2 进程管理之获取内存和进程数 4
6.2.1 界面 UI 4
6.2.2 进程信息的实体类 5
6.2.3 工具类之获取系统信息 6
6.2.4 工具类之获取进程信息 9
6.2.5 主逻辑代码 11
6.3 进程管理之手机进程展示 13
6.3.1 界面 UI 13
6.3.2 进程管理之 ListView 的 Item 布局 14
6.3.3 进程管理之 ListView 的数据适配器 16
6.3.4 进程管理之 ListView 的条目点击事件 18
6.3.5 进程管理界面 ListView 优化 20
6.4 进程管理之手机进程清理 25
6.4.1 全选与反选界面 UI 25
6.4.2 全选与反选之按钮状态 26
6.4.3 进程管理之进程清理 29
6.4.4 头标题中进程数的更新 32
6.5 进程管理之抽屉界面 34
6.5.1 界面 UI 34
6.5.2 抽屉动画 38
6.6 进程管理之抽屉界面功能实现 40
6.6.1 显示系统进程选项的状态更新 40

6.6.2 显示系统进程选项的功能实现.....42
6.6.3 锁屏自动清理选项的状态更新.....42
6.6.4 锁屏自动清理选项的功能实现.....44
6.7 本章小结.....46

第 6 章 进程管理模块

- 了解进程管理模块功能
- 掌握如何获取手机中的进程
- 掌握如何结束正在运行的进程

大家都知道，大部分 Android 手机用的时间越长就会变得越慢甚至卡顿，哪怕是 2G 的运行内存，用的时间长了手机也会变得奇慢无比。通常情况下，造成这种情况的原因有两种，一种是缓存垃圾过多，一种是后台开启的进程过多。在第 6 章中已经针对缓存清理进行了讲解，本章将针对进程管理进行详细讲解，以解决手机运行速度过慢问题。

6.1 模块概述

6.1.1 功能介绍

进程管理模块主要用于查看当前开启多少进程服务，其中包含用户进程个数和系统进程个数，我们可以选择清理某个进程，也可以选择清理所有进程，同时还可以设置是否显示系统进程以及锁屏时是否清理进程，该模块界面效果如图 6-1 所示。

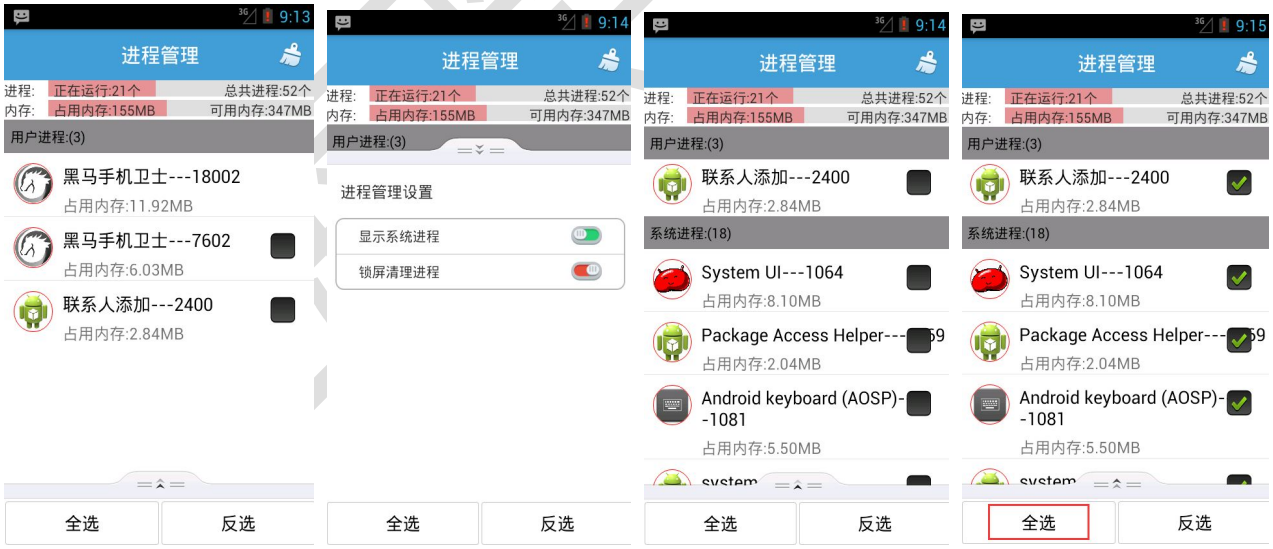

图 6-1 进程管理界面

6.2 进程管理之获取内存和进程数

6.2.1 界面 UI

进程管理模块界面 TaskManagerActivity，可以看出它的界面跟软件管理的界面有很多相似之处，如下图 6-2 中的两张图所示，进程管理界面只是比软件管理界面多了上方右上角一个清理进程的按钮，加上下方的两个按钮和一个抽屉功能。

这里，我们首先实现上方获取进程和内存信息的功能，将软件管理的界面布局的代码直接拿过来。

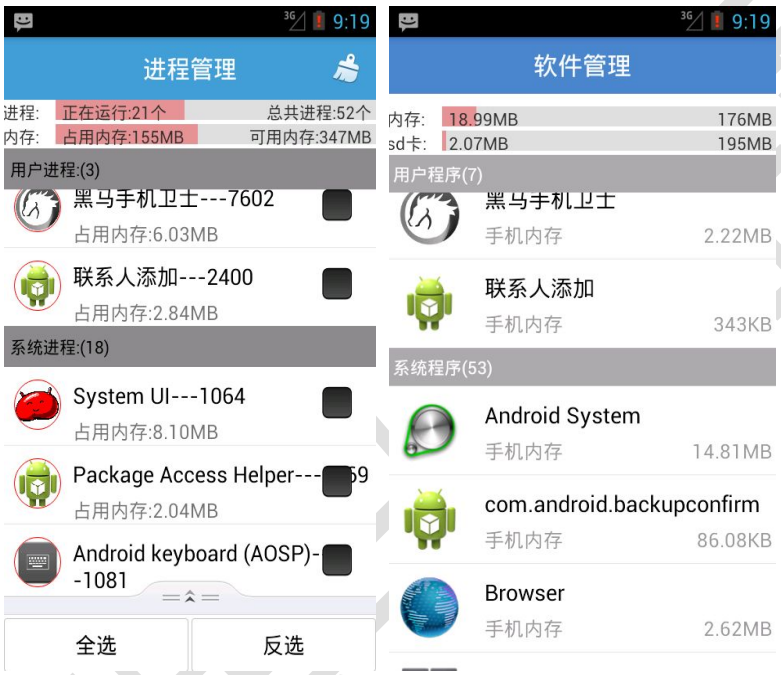

图 6-2 进程管理界面和软件管理界面之间的比较

图 6-2 进程管理界面上方用于显示进程数和内存的布局代码如文件【6-1】所示。

【文件 6-1】 activity_processmanager.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="进程管理" />
9.      <!-- 进程数 -->
10.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
11.         android:id="@+id/pdv_task_count"
12.         android:layout_width="match_parent"
13.         android:layout_height="wrap_content"
14.         android:layout_marginTop="10dp" />
15.     <!-- 内存 rom : 类似电脑上面的一块存储空间
```

```
16.                 ram ： 代表运行内存-->
17.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
18.         android:id="@+id/pdv_ram"
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content" />
21. </LinearLayout>
```

主要由 1 个标题、2 个自定义的进度条组合控件组成。其中，第 1 个进度条组合控件用于显示正在运行的进程个数，第二个进度条组合控件用于显示可用内存和总内存大小。

6.2.2 进程信息的实体类

在 ListView 列表展示的进程信息中，包含应用名称、应用图标、程序包名、占用内存大小、是否是系统进程。如下图 6-3 所示。因此需要定义一个进程的实体类来封装这些信息，具体代码如【文件 6-2】所示。

图 6-3 进程信息的实体类

【文件 6-2】 TaskInfo.java

```
1.  public class TaskInfo {
2.      /**
3.       * 进程图标
4.       */
5.      public Drawable icon;
6.      /**
7.       * apk 的名字
8.       */
9.      public String appName;
10.     /**
11.      * apk 的包名
12.      */
```

```
    public String apkPackageName;
    /**
     * 占用内存大小
     */
    public long memSize;
    /**
     * 判断是否是用户进程
     * true  表示用户进程
     * false 表示系统进程
     */
    public boolean isUserTask;
    /**
     * 用于显示 CheckBox 是否被选中
     */
    public boolean checked;
}
```

6.2.3 工具类之获取系统信息

在主界面逻辑代码中，使用了 TaskUtils 工具类，该类主要用于获取系统信息，它有 5 个方法，分别是获取手机总内存大小、获取可用的内存信息、正在运行的进程数量、总的进程数和低版本下的总内存大小，具体代码如【文件 6-3】所示。

【文件 6-3】TaskUtils.java

```
public class TaskUtils {
    /**
     * 获取正在运行的进程的个数
     * @param context
     */
    public static int getRunningProcessCount(Context context) {
        // 获取到进程管理器
        ActivityManager am = (ActivityManager) context
                .getSystemService(Context.ACTIVITY_SERVICE);
        // 获取所有正在运行的进程集合
        List<RunningAppProcessInfo> appProcesses = am.getRunningAppProcesses();
        // 正在运行的进程的个数
        return appProcesses.size();
    }
    /**
     * 获取总共有多少个进程
     */
    public static int getRunningTotalCount(Context context) {
        // 获取总共有多少个进程
        PackageManager packageManager = context.getPackageManager();
```

```
21.         List<PackageInfo> installedPackages = packageManager
22.                 .getInstalledPackages(0);
23.         // 初始化 hashset
24.         // 记录一共有多少个进程
25.         int count = 0;
26.         for (PackageInfo packageInfo : installedPackages) {
27.             HashSet<String> set = new HashSet<String>();
28.             // 获取到进程的名字
29.             String processName = packageInfo.applicationInfo.processName;
30.             set.add(processName);
31.             ActivityInfo[] activities = packageInfo.activities;
32.             if (null != activities) {
33.                 for (ActivityInfo activityInfo : activities) {
34.                     processName = activityInfo.processName;
35.                     set.add(processName);
36.                 }
37.             }
38.             ServiceInfo[] services = packageInfo.services;
39.             if (null != services) {
40.                 for (ServiceInfo serviceInfo : services) {
41.                     processName = serviceInfo.processName;
42.                     set.add(processName);
43.                 }
44.             }
45.             ActivityInfo[] receivers = packageInfo.receivers;
46.             if (null != receivers) {
47.                 for (ActivityInfo activityInfo : receivers) {
48.                     processName = activityInfo.processName;
49.                     set.add(processName);
50.                 }
51.             }
52.             ProviderInfo[] providers = packageInfo.providers;
53.             if (null != providers) {
54.                 for (ProviderInfo providerInfo : providers) {
55.                     processName = providerInfo.processName;
56.                     set.add(processName);
57.                 }
58.             }
59.             count += set.size();
60.         }
61.         return count;
62.     }
63.     /**
```

```
64.      * 获取到剩余内存
65.      * @param context
66.      * @return
67.      */
68.     public static long getAvailMem(Context context) {
69.         ActivityManager am = (ActivityManager) context
70.                 .getSystemService(Context.ACTIVITY_SERVICE);
71.         MemoryInfo outInfo = new MemoryInfo();
72.         // 获取到内存的基本信息
73.         am.getMemoryInfo(outInfo);
74.         // 获取到可用内存(剩余内存)
75.         return outInfo.availMem;
76.     }
77.     /**
78.      * 获取到总共有多少内存
79.      * @param context
80.      * @return
81.      */
82.     @SuppressLint("NewApi")
83.     public static long getTotalMem(Context context) {
84.         ActivityManager am = (ActivityManager) context
85.                 .getSystemService(Context.ACTIVITY_SERVICE);
86.         MemoryInfo outInfo = new MemoryInfo();
87.         am.getMemoryInfo(outInfo);
88.         // 判断当前用户的版本号
89.         // 这个方法在 API16 里面才有
90.         // 适配低版本的手机
91.         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
92.             return outInfo.totalMem;
93.         } else {
94.             return getLowTotalMem();
95.         }
96.     }
97.     /**
98.      * 获取到低版本总共有多少内存
99.      */
100.        private static long getLowTotalMem() {
101.    //      总共的内存
102.    //      MemTotal:        513492 kB
103.            File file = new File("/proc/meminfo");
104.            try {
105.                BufferedReader reader = new BufferedReader(new InputStreamReader(new
106.     FileInputStream(file)));
107.                //读取一行
```

```
108.                String readLine = reader.readLine();
109.                //替换
110.                readLine = readLine.replace("MemTotal:", "");
111.                readLine = readLine.replace("kB", "");
112.                readLine = readLine.trim();
113.                //方便格式化
114.                return Long.parseLong(readLine) * 1024;
115.            } catch (Exception e) {
116.                // TODO Auto-generated catch block
117.                e.printStackTrace();
118.            }
119.            return 0;
120.        }
121.    }
```

6.2.4 工具类之获取进程信息

在进程管理界面中，需要获取正在运行的进程，该功能代码相对比较独立，因此将其抽取出来作为一个单独的工具类供其使用，具体代码如【文件 6-4】所示。

【文件 6-4】TaskInfoParser.java

```
1.  public class TaskInfoParser {
2.      private static String processName;
3.      private static TaskInfo taskInfo;
4.      public static List<TaskInfo> getTaskInfos(Context context) {
5.          // 获取到进程管理者
6.          ActivityManager am = (ActivityManager) context
7.                  .getSystemService(Context.ACTIVITY_SERVICE);
8.          // 获取到所有正在运行的进程
9.          List<RunningAppProcessInfo> processes = am.getRunningAppProcesses();
10.         PackageManager packageManager = context.getPackageManager();
11.         // 获取到手机上面所有的包
12.         // 获取到所有的 apk
13.         List<PackageInfo> installedPackages = packageManager
14.                 .getInstalledPackages(0);
15.         List<TaskInfo> lists = new ArrayList<TaskInfo>();
16.         for (int i = 0; i < processes.size(); i++) {
17.             try {
18.                 taskInfo = new TaskInfo();
19.                 // 获取到正在运行当前进程的对象
20.                 RunningAppProcessInfo processInfo = processes.get(i);
21.                 processName = processes.get(i).processName;
22.                 taskInfo.apkPackageName = processName;
23.                 // 不使用 PackageInfo packageInfo = installedPackages.get(i)的原因是
```

```
24.                 // 手机中所有程序并非全部都在运行，应使用下面这种方式
25.                 PackageInfo packageInfo = packageManager.getPackageInfo(
26.                         processName, 0);
27.                 // 获取到当前应用程序的图标
28.                 Drawable icon = packageInfo.applicationInfo.loadIcon(packageManager);
29.                 taskInfo.icon = icon;
30.                 // 获取到应用程序的名字
31.                 String appName = packageInfo.applicationInfo.loadLabel(
32.                         packageManager).toString();
33.                 taskInfo.appName = appName;
34.                 // 获取到进程内存的基本信息
35.                 MemoryInfo[] memoryInfo = am
36.                         .getProcessMemoryInfo(new int[] { processInfo.pid });
37.                 // Dirty 弄脏 表示占用了多少内存
38.                 // * 1024 方便格式化
39.                 long totalPrivateDirty = memoryInfo[0].getTotalPrivateDirty() * 1024;
40.                 taskInfo.memSize = totalPrivateDirty;
41.                 int flags = packageInfo.applicationInfo.flags;
42.                 if ((flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
43.                     taskInfo.isUserTask = false;
44.                 } else {
45.                     taskInfo.isUserTask = true;
46.                 }
47.             } catch (Exception e) {
48.                 // TODO Auto-generated catch block
49.                 e.printStackTrace();
50.                 // taskInfo = new TaskInfo();
51.                 // 考虑到有些应用程序时没有应用图标
52.                 // 如果当前应用程序报错了。那么我们给一个默认的图片
53.                 taskInfo.icon = context.getResources().getDrawable(
54.                         R.drawable.ic_launcher);
55.                 taskInfo.appName = processName;
56.             }
57.             lists.add(taskInfo);
58.         }
59.         return lists;
60.     }
61. }
```

- 第 5~9 行代码用于获取进程管理器，然后得到所有正在运行的进程数，一个包管理器 PackageManager，通过包管理器获取正在运行的程序。
- 第 10~14 行代码用于获取包管理器 PackageManager，通过包管理器获取手机上所有的安装包。
- 第 15 行代码创建了一个 List<TaskInfo>进程集合，用于存储进程对象。
- 第 16~46 行代码用于循环遍历所有正在运行的进程，并获取程序包名、占用内存大小、程序图标、程序名称，最后将进程信息封装到 TaskInfo 对象中。

- 第 48~55 行代码当应用程序获取应用图标失败后，我们自己给一个默认的图片
- 第 57 行代码将当前获取的进程信息放到集合中，

6.2.5 主逻辑代码

上面我们已经将对应的工具类定义好，下面我们开始获取手机中的进程数和内存信息，TaskManagerActivity.java 中对应的逻辑代码如文件【6-5】所示。

【文件 6-5】 com.itheima.mobilesafe_sh2.act/TaskManagerActivity.java

```
1.  public class TaskManagerActivity extends Activity {
2.      private ProgressDesView pdv_task_count;
3.      private ProgressDesView pdv_ram;
4.       private List<TaskInfo> taskInfos;
5.      @Override
6.      protected void onCreate(Bundle savedInstanceState) {
7.          // TODO Auto-generated method stub
8.          super.onCreate(savedInstanceState);
9.          initView();
10.         initData();
11.     }
12.     private void initView() {
13.         setContentView(R.layout.activity_task_manager);
14.         mListView = (ListView) findViewById(R.id.list_view);
15.         // 进程数
16.         pdv_task_count = (ProgressDesView) findViewById(R.id.pdv_task_count);
17.         // 运行内存
18.         pdv_ram = (ProgressDesView) findViewById(R.id.pdv_ram);
19.         // 初始化进程个数
20.         initTaskCount();
21.         // 初始化内存
22.         initMem();
23.     }
24.
25.   //初始化数据，获取到所有的进程详细信息，用于在主界面中进行展示
26.    private void initData() {
27.         new Thread() {
28.             public void run() {
29.                 // 获取到所有的进程
30.                 taskInfos = TaskInfoParser.getTaskInfos(TaskManagerActivity.this);
31.                 Message msg = Message.obtain();
32.                 msg.what = 0;
33.                 handler.sendMessage(msg);
34.             };
35.         }.start();
```

```
36.     }
37.     private Handler handler = new Handler() {
38.         public void handleMessage(android.os.Message msg) {
39.             switch (msg.what) {
40.             case 0:
41.               System.out.println("进程个数--" + taskInfos.size());
42.                 break;
43.             }
44.         };
45.     };
46.     /**
47.      * 初始化内存
48.      */
49.     private void initMem() {
50.         // 获取到剩余内存
51.         long availMem = TaskUtils.getAvailMem(this);
52.         // 获取总共的内存
53.         long totalMem = TaskUtils.getTotalMem(this);
54.         // 使用的内存
55.         long userMem = totalMem - availMem;
56.         pdv_ram.setTitle("内存");
57.         // 设置占用的内存
58.         pdv_ram.setTvLeft("占用内存" + Formatter.formatFileSize(this, userMem));
59.         // 设置可用内存
60.         pdv_ram.setTvRight("可用内存" + Formatter.formatFileSize(this, availMem));
61.         // 设置进度条
62.         pdv_ram.setProgress((int) (userMem * 100f / totalMem));
63.     }
64.     /**
65.      * 初始化进程个数
66.      */
67.     private void initTaskCount() {
68.         // 获取正在运行的进程个数
69.         int runningProcessCount = TaskUtils.getRunningProcessCount(this);
70.         // 总共的进程个数
71.         int runningTotalCount = TaskUtils.getRunningTotalCount(this);
72.         // 设置标题
73.         pdv_task_count.setTitle("进程:");
74.         // 设置进程个数
75.         pdv_task_count.setTvLeft("正在运行" + runningProcessCount + "个");
76.         // 设置总共有多少个进程
77.         pdv_task_count.setTvRight("总共进程" + runningTotalCount + "个");
78.         // 设置进度
79.         pdv_task_count
```

```
80.                 .setProgress((int) (runningProcessCount * 100f / runningTotalCount));
81.     }
82. }
```

上面的代码书写好之后，运行程序，效果图如图 6-4 所示。

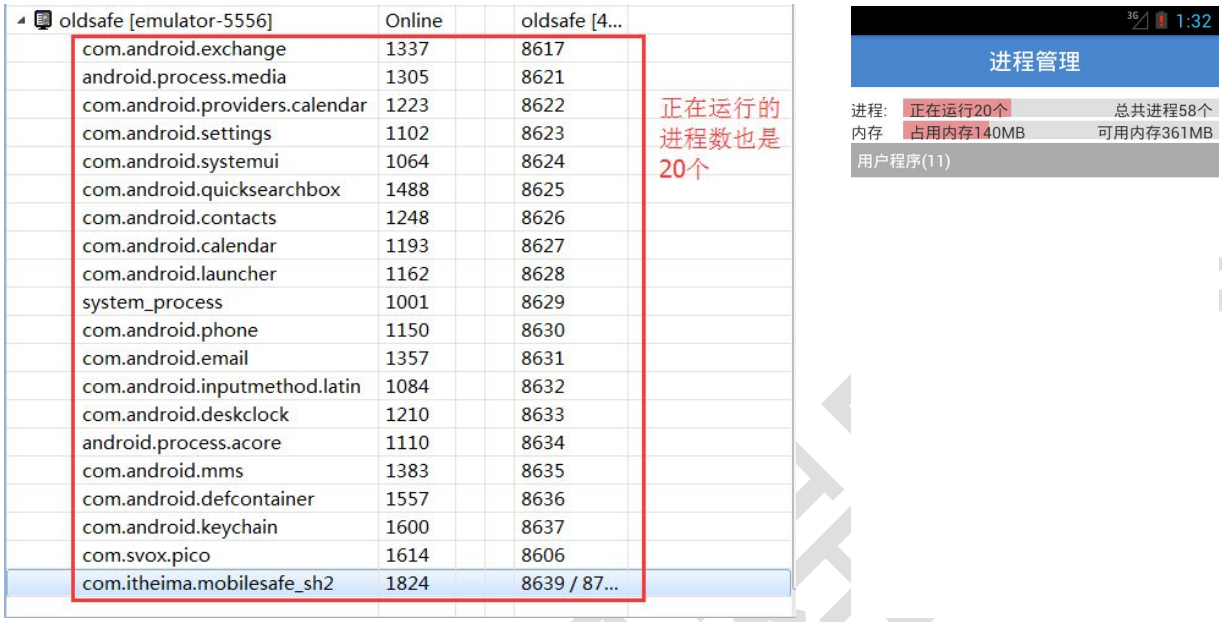

图 6-4 头标题中内存和进程信息

6.3 进程管理之手机进程展示

6.3.1 界面 UI

前面我们已经将手机中所有正在运行的进程信息已经获取到了，下面我们开始在进程管理界面进行展示，与软件管理界面类似，我们也需要使用 ListView 进行数据展示。

首先，修改进程管理界面之前的布局文件，代码修改为如下所示。

```
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="vertical" >
6.      <TextView
7.          style="@style/TitleBarTextView"
8.          android:text="软件管理" />
9.      <!-- 进程数 -->
10.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
11.         android:id="@+id/pdv_task_count"
12.         android:layout_width="match_parent"
13.         android:layout_height="wrap_content"
```

```
14.         android:layout_marginTop="10dp" />
15.     <!-- 内存 rom : 类似电脑上面的一块存储空间
16.                 ram :  代表运行内存-->
17.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
18.         android:id="@+id/pdv_ram"
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content" />
21.     <FrameLayout
22.         android:layout_width="match_parent"
23.         android:layout_height="match_parent" >
24.         <ListView
25.             android:id="@+id/list_view"
26.             android:layout_width="match_parent"
27.             android:layout_height="wrap_content" >
28.         </ListView>
29.        <!-- <TextView
30.             android:id="@+id/tv_head_title"
31.             android:layout_width="match_parent"
32.             android:layout_height="wrap_content"
33.             android:background="@android:color/darker_gray"
34.             android:padding="5dp"
35.             android:text="用户程序(11)"
36.             android:textColor="#fff" />  --!>
37.         <ProgressBar
38.             android:id="@+id/pb"
39.             android:layout_width="wrap_content"
40.             android:layout_height="wrap_content"
41.             android:layout_gravity="center"
42.             android:indeterminateDrawable="@drawable/progress_medium"
43.             android:visibility="invisible" />
44.     </FrameLayout>
45. </LinearLayout>
```

6.3.2 进程管理之 ListView 的 Item 布局

由于进程管理界面中 ListView 展示数据的布局都一样，因此可以定义一个 Item 条目布局用于展示数据，效果图如图 6-5 所示，具体代码如【文件 6-6】所示。

图 6-5 进程管理界面 Item 布局

【文件 6-6】item_processmanager_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:orientation="vertical" >
    <!-- icon -->
    <ImageView
        android:id="@+id/iv_icon"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@drawable/ic_launcher" />
    <!-- app 名字 -->
    <TextView
        android:id="@+id/tv_task_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"
        android:layout_toRightOf="@id/iv_icon"
        android:text="应用的名字"
        android:textSize="18sp" />
    <!-- 占用内存大小 -->
    <TextView
        android:id="@+id/tv_task_mem_size"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/tv_task_name"
```

```
28.         android:layout_marginLeft="5dp"
29.         android:layout_marginTop="3dp"
30.         android:layout_toRightOf="@id/iv_icon"
31.         android:text="占用内存大小"
32.         android:textSize="16sp" />
33.     <CheckBox
34.         android:id="@+id/cb_state"
35.         android:layout_width="wrap_content"
36.         android:layout_height="wrap_content"
37.         android:layout_alignParentRight="true"
38.         android:layout_margin="3dp"
39.         android:button="@drawable/checkbox_selected" />
40. </RelativeLayout>
```

上述布局文件中，ImageView 控件用于展示程序图标，第一个 TextView 控件用于展示程序名称，第二个 TextView 用于展示程序占用内存大小，CheckBox 控件用于选中当前程序。并且 CheckBox 控件还使用了背景选择器，具体如【文件 6-7】所示。

【文件 6-7】res/drawable/green_checkbox_selector.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3.      <!-- 需要注意 checkbox 是设置 state_checked ，不是按压 -->
4.  <item android:drawable="@drawable/ic_security_checkbox_checked" android:state_checked=
5.        "true"></item>
6.      <!-- 这个地方需要注意不要设置为默认的。必须设置成 false -->
7.  <itemandroid:drawable="@drawable/ic_security_checkbox_disable_unchecked"
8.        android:state_checked="false"></item>
9.  </selector>
```

6.3.3 进程管理之 ListView 的数据适配器

当数据获取完之后，我们需要将当前的数据在当前的进程管理界面进行展示，需要写出 ListView 的数据适配器，具体的代码如下所示。

```
1.  private class TaskManagerAdater extends BaseAdapter {
2.          @Override
3.          public int getCount() {
4.              // TODO Auto-generated method stub
5.              return taskInfos.size();
6.          }
7.          @Override
8.          public Object getItem(int position) {
9.              return null;
10.         }
11.         @Override
12.         public long getItemId(int position) {
```

```
13.             // TODO Auto-generated method stub
14.             return position;
15.         }
16.         @Override
17.         public View getView(int position, View convertView, ViewGroup parent) {
18.             // TODO Auto-generated method stub
19.             View view = View.inflate(TaskManagerActivity.this,
20.                     R.layout.item_task_manager, null);
21.             // 图标
22.             ImageView iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
23.             // app 名字
24.             TextView tv_task_name = (TextView) view.findViewById(R.id.tv_task_name);
25.             // 占用内存大小
26.             TextView tv_task_mem_size = (TextView) view
27.                     .findViewById(R.id.tv_task_mem_size);
28.             // 勾选状态
29.             CheckBox cb_state = (CheckBox) view.findViewById(R.id.cb_state);
30.             TaskInfo taskInfo = taskInfos.get(position);
31.             iv_icon.setImageDrawable(taskInfo.icon);
32.             tv_task_name.setText(taskInfo.appName);
33.             tv_task_mem_size.setText(Formatter.formatFileSize(
34.                     TaskManagerActivity.this, taskInfo.memSize));
35.             return view;
36.         }
37.     }
```

运行程序，效果图如图 6-6 所示。

图 6-6 进程管理界面

在图 6-6 的进程界面中，我们点击当前 ListView 的条目时发现右边的 CheckBox 没有被选中，只有鼠标放在 CheckBox 上点击时按钮才会被选中，这是因为类似 Button 或者 CheckBox 的按钮会抢夺当前所在

条目的焦点。因此，我们需要在 CheckBox 上增加两条属性，让当前的 CheckBox 获取不到焦点，处理触摸的事件焦点交给 ListView 的条目，对应修改的代码如下所示。

```
<!-- 是否选中 -->
  <!-- android:focusable="false" 不可触摸 -->
  <!-- android:clickable="false" 不可点击 -->
<!--  需要注意：如果是 CheckBox 自己的选中状态一般不给自己处理。一般都是给 listview 进行处理 -->
  <CheckBox
      android:id="@+id/cb_state"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:layout_margin="3dp"
      android:button="@drawable/checkbox_selected"
      android:clickable="false"
      android:focusable="false" />
```

6.3.4 进程管理之 ListView 的条目点击事件

当进程信息显示出来后，在点击当前条目时右边按钮会显示被选中的状态，反之亦然。首先，我们需要实现监听 ListView 的条目点击事件，在点击事件中进行处理。

定义一个初始化监听的方法 initListener( )，在该方法中定义 ListView 的点击事件，代码如下所示。

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        initView();
        initData();
        //监听 ListView 的条目点击事件
        initListener();
    }
    private void initListener() {
        mListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                // 实际上此处返回的 itemAtPosition 就是 ListView 适配器中 getItem()的返回值，
                // 因此，需要在 ListView 中将当前点击集合中 position 的元素返回。
                Object itemAtPosition = mListView.getItemAtPosition(position);
                if (itemAtPosition != null) {
                    TaskInfo taskInfo = (TaskInfo) itemAtPosition;
                    // 如果当前对象已经勾选。点击之后。变成没有勾选
                    // 如果没有勾选。点击之后变成已经勾选
                    if (taskInfo.checked) {
```

```
                        taskInfo.checked = false;
                    } else {
                        taskInfo.checked = true;
                    }
                    // 刷新界面
                    adater.notifyDataSetChanged();
                }
            }
        });
    }
```

此时，需要在 ListView 适配器的 getView( )方法中设置右边 CheckBox 的选中状态，并在 getItem( )方法中返回当前被点击的 TaskInfo 对象，修改后的代码如下所示。

```
    @Override
      public TaskInfo getItem(int position) {
          TaskInfo info = taskInfos.get(position);
          return info;
      }
      @Override
      public View getView(int position, View convertView, ViewGroup parent) {
          // TODO Auto-generated method stub
          View view = View.inflate(TaskManagerActivity.this,
                  R.layout.item_task_manager, null);
          // 图标
          ImageView iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
          // app 名字
          TextView tv_task_name = (TextView) view.findViewById(R.id.tv_task_name);
          // 占用内存大小
          TextView tv_task_mem_size = (TextView) view
                  .findViewById(R.id.tv_task_mem_size);
          // 勾选状态
          CheckBox cb_state = (CheckBox) view.findViewById(R.id.cb_state);
          TaskInfo taskInfo = taskInfos.get(position);
          iv_icon.setImageDrawable(taskInfo.icon);
          tv_task_name.setText(taskInfo.appName);
          tv_task_mem_size.setText(Formatter.formatFileSize(
                  TaskManagerActivity.this, taskInfo.memSize));
          // 设置 CheckBox 按钮的选中状态
          cb_state.setChecked(taskInfo.checked);
          return view;
      }
```

运行程序，效果图如图 6-7 所示，按钮能够正常响应点击事件。

图 6-7 ListView 条目点击的按钮状态

6.3.5 进程管理界面 ListView 优化

与软件管理界面类似，我们需要将手机中用户进程和系统进程分开进行显示，这就需要我们在获取手机中进程信息的时候就需要进行一些必要的标记操作，然后在界面初始化获取所有进程信息时，根据这个标记将用户进程和系统进程分成两个集合存储。

首先，在工具类 TaskInfoParser.java 中需要对应用程序 TaskInfo 进行标记，判断用户程序还是系统程序，代码如下所示。

```
1.                 int flags = packageInfo.applicationInfo.flags;
2.                 if ((flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
3.                     taskInfo.isUserTask = false;
4.                 } else {
5.                     taskInfo.isUserTask = true;
6.                 }
```

然后，在 TaskManagerActivity.java 的初始化数据时，区分开用户程序和系统程序，并在适配器中进行不同的展示，这方面的实现类似于软件管理界面中 ListView 的不同布局功能的实现，增加了头标题信息，用于标示当前的进程属于用户还是系统。

此时初始化数据的代码和适配器的代码如下所示。

```
1.     private List<TaskInfo> taskInfos;
2.      private void initData() {
3.          new Thread() {
4.              public void run() {
5.                  // 获取到所有的进程
6.                  taskInfos = TaskInfoParser.getTaskInfos(TaskManagerActivity.this);
7.                  userList = new ArrayList<TaskInfo>();
8.                  systemList = new ArrayList<TaskInfo>();
9.                  // 一份为二
10.                 for (TaskInfo info : taskInfos) {
```

```
                    if (info.isUserTask) {
                        userList.add(info);
                    } else {
                        systemList.add(info);
                    }
                }
                Message msg = Message.obtain();
                msg.what = 0;
                handler.sendMessage(msg);
            };
        }.start();
    }
    private TaskManagerAdater adater;
    //定义 Handle 接收初始化数据的信息，数据获取完毕后就进行展示
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
            case 0:
                System.out.println("进程个数--" + taskInfos.size());
                adater = new TaskManagerAdater();
                mListView.setAdapter(adater);
                break;
            }
        };
    };
    private class TaskManagerAdater extends BaseAdapter {
        private ViewHolder holder;
        @Override
        public int getCount() {
            return userList.size() + 1 + systemList.size() + 1;
        }
        @Override
        public TaskInfo getItem(int position) {
            if (position == 0 || position == userList.size() + 1) {
                return null;
            }
            TaskInfo info;
            if (position < userList.size() + 1) {
                info = userList.get(position - 1);
            } else {
                int location = position - 1 - userList.size() - 1;
                info = systemList.get(location);
            }
```

```
54.             return info;
55.         }
56.         @Override
57.         public long getItemId(int position) {
58.             return position;
59.         }
60.         @Override
61.         public int getItemViewType(int position) {
62.             if (position == 0 || position == userList.size() + 1) {
63.                 return 0;
64.             } else {
65.                 return 1;
66.             }
67.         }
68.         @Override
69.         public int getViewTypeCount() {
70.             return 2;
71.         }
72.         @Override
73.         public View getView(int position, View convertView, ViewGroup parent) {
74.             int type = getItemViewType(position);
75.             switch (type) {
76.             case 0:
77.                 TextView textView = new TextView(TaskManagerActivity.this);
78.                 // 在 android 系统里面所有通过代码去设置大小的地方全部都是像素
79.                 textView.setPadding(5, 5, 5, 5);
80.                 textView.setBackgroundColor(Color.GRAY);
81.                 textView.setTextColor(Color.WHITE);
82.                 if (position == 0) {
83.                     textView.setText("用户进程(" + userList.size() + ")");
84.                 } else {
85.                     textView.setText("系统进程(" + systemList.size() + ")");
86.                 }
87.                 return textView;
88.             case 1:
89.                 if (convertView == null) {
90.                     convertView = View.inflate(TaskManagerActivity.this,
91.                             R.layout.item_task_manager, null);
92.                     holder = new ViewHolder();
93.                     // 图标
94.                     holder.iv_icon = (ImageView) convertView
95.                             .findViewById(R.id.iv_icon);
96.                     // app 名字
97.                     holder.tv_task_name = (TextView) convertView
```

```
98.                             .findViewById(R.id.tv_task_name);
99.                     // 占用内存大小
100.                        holder.tv_task_mem_size = (TextView) convertView
101.                                .findViewById(R.id.tv_task_mem_size);
102.                        // 勾选状态
103.                        holder.cb_state = (CheckBox) convertView
104.                                .findViewById(R.id.cb_state);
105.                        convertView.setTag(holder);
106.                    } else {
107.                        holder = (ViewHolder) convertView.getTag();
108.                    }
109.                    TaskInfo taskInfo = getItem(position);
110.                    holder.iv_icon.setImageDrawable(taskInfo.icon);
111.                    holder.tv_task_name.setText(taskInfo.appName);
112.                    holder.tv_task_mem_size.setText(Formatter.formatFileSize(
113.                            TaskManagerActivity.this, taskInfo.memSize));
114.                    holder.cb_state.setChecked(taskInfo.checked);
115.                    break;
116.                }
117.                return convertView;
118.            }
119.        }
120.        //定义 ViewHolder
121.        static class ViewHolder {
122.            ImageView iv_icon;
123.            TextView tv_task_name;
124.            TextView tv_task_mem_size;
125.            CheckBox cb_state;
126.        }
127.
```

- 第 2~16 行初始化数据，使用工具类 TaskInfoParser 获取到所有的进程信息，然后遍历数据集合，根据封装 TaskInfo 中的 isUserTask 属性区分开用户程序还是系统程序。
- 第 17~19 行获取 Message 信息，向 Handle 发送消息。
- 第 25~35 行创建 Handle 对象，收到子线程发送的消息后进行数据展示。
- 第 36~119 行是进程管理界面的适配器 TaskManagerAdater 代码，其中第 43~55 行代码用于根据不同的 position 返回不同的条目布局填充数据。第 61~67 行代码用于返回当前 ListView 中根据 position 定义的两个不同条目布局类型。第 73~119 行代码是 getView( )方法，在该方法中根据不同的布局类型，在 ListView 中显示不同的数据。
- 第 120~126 代码定义了 ViewHolder，用于保存当前布局中 View 对象。

数据适配器中的代码修改好之后，运行程序，效果图如图 6-8 所示。

图 6-8 进程管理界面的优化

考虑到清理进程的时候不可能将当前黑马手机卫士的进程清理掉。因此，就需要我们将当前的黑马手机卫士进程的按钮给隐藏掉，这个代码需要在适配器的 getView 方法中进行设置。添加的代码如下所示。

```
1.                  // 通过包名进行判断。如果是自己的 app。那么把当前的选择框隐藏
2.                  if (taskInfo.apkPackageName.equals(getPackageName())) {
3.                      holder.cb_state.setVisibility(View.INVISIBLE);
4.                  } else {
5.                      holder.cb_state.setVisibility(View.VISIBLE);
6.                  }
```

修改代码之后，运行程序，效果图如图 6-9 所示，当前进程的按钮已被隐藏。

图 6-9 隐藏当前黑马手机卫士进程的按钮

6.4 进程管理之手机进程清理

6.4.1 全选与反选界面 UI

前面我们已经手机中的所有进程显示出来，下面我们要开始进行进程的清理操作，这里我们需要在界面的下边添加两个按钮，即全选和反选。

首先，布局需要进行修改，需要注意的是，要在之前的帧布局中添加一个 android:layout_weight="1"，否则当前添加的按钮显示不了，将会被挤出去。对应的代码如文件【6-8】所示。

【文件 6-8】 res/layout/activity_task_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.      <!-- 进程数 -->
10.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
11.         android:id="@+id/pdv_task_count"
12.         android:layout_width="match_parent"
13.         android:layout_height="wrap_content"
14.         android:layout_marginTop="10dp" />
15.     <!-- 内存 rom : 类似电脑上面的一块存储空间
16.                 ram :  代表运行内存-->
17.     <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
18.         android:id="@+id/pdv_ram"
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content" />
21.
22.     <FrameLayout
23.         android:layout_width="match_parent"
24.         android:layout_height="match_parent"
25.         android:layout_weight="1">
26.         <ListView
27.             android:id="@+id/list_view"
28.             android:layout_width="match_parent"
29.             android:layout_height="wrap_content" >
30.         </ListView>
31.         <ProgressBar
32.             android:id="@+id/pb"
```

```
33.            android:layout_width="wrap_content"
34.            android:layout_height="wrap_content"
35.            android:layout_gravity="center"
36.            android:indeterminateDrawable="@drawable/progress_medium"
37.            android:visibility="invisible" />
38.     </FrameLayout>
39.
40.     <LinearLayout
41.        android:layout_width="match_parent"
42.        android:layout_height="wrap_content"
43.        android:orientation="horizontal" >
44.        <Button
45.           android:layout_width="0dp"
46.           android:layout_height="wrap_content"
47.           android:layout_weight="1"
48.           android:background="@drawable/dg_button_cancel_selected"
49.           android:onClick="allSelect"
50.           android:text="全选" />
51.        <Button
52.           android:layout_width="0dp"
53.           android:layout_height="wrap_content"
54.           android:layout_weight="1"
55.           android:background="@drawable/dg_button_cancel_selected"
56.           android:onClick="oppsiteSelect"
57.           android:text="反选" />
58.     </LinearLayout>
59. </LinearLayout>
```

其中，dg_button_cancel_selected.xml 是全选和反选按钮的背景选择器，对应的代码如文件【6-9】所示。

【文件 6-9】 Res/drawable/dg_button_cancel_selected.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3.  <item  android:drawable="@drawable/dg_button_cancel_select" android:state_pressed=
4.       "true"></item>
5.  <item  android:drawable="@drawable/dg_button_cancel_normal"></item>
6.  </selector>
```

6.4.2 全选与反选之按钮状态

为了更好的观察当前所有的进程是否被全选或者反选的效果图，这里将之前用于隐藏当前进程的代码注释掉。

首先，实现布局中全选按钮的点击事件，同时需要注意，避免将当前的黑马手机卫士进程选中，我们需要进行判断，具体的代码如下所示。

```
1.      /**
2.       * 全部选择
3.       * @param view
4.       */
5.      public void allSelect(View view) {
6.          // 迭代用户集合
7.          for (TaskInfo info : userList) {
8.              // 判断当前用户集合的表名是否等于自己的包名
9.              if (info.apkPackageName.equals(getPackageName())) {
10.                 continue;
11.             }
12.             info.checked = true;
13.         }
14.         for (TaskInfo info : systemList) {
15.             info.checked = true;
16.         }
17.         //用于适配器的刷新
18.         adater.notifyDataSetChanged();
19.     }
```

运行程序，效果图如图 6-10 所示。

图 6-10 进程管理之全选按钮的实现

接着，实现反选按钮的点击事件，具体的代码如下所示。

```
1.      /**
2.       * 反选
3.       * @param view
4.       */
5.      public void oppsiteSelect(View view) {
```

```
6.          // 迭代用户集合
7.          for (TaskInfo info : userList) {
8.              // 判断当前用户集合的表名是否等于自己的包名
9.              if (info.apkPackageName.equals(getPackageName())) {
10.                 continue;
11.             }
12.             if (info.checked) {
13.                 info.checked = false;
14.             } else {
15.                 info.checked = true;
16.             }
17.         }
18.         for (TaskInfo info : systemList) {
19.             if (info.checked) {
20.                 info.checked = false;
21.             } else {
22.                 info.checked = true;
23.             }
24.         }
25.         adater.notifyDataSetChanged();
26.     }
```

运行程序，效果图如图 6-11 所示。

图 6-11 进程管理之反选按钮的实现

另外，为了让当前的黑马手机卫士进程不会被用户点击选中，仅仅隐藏按钮是不可以的，这里需要在 ListView 的点击事件中进行相应的处理，修改的代码如下所示。

```
1.  mListView.setOnItemClickListener(new OnItemClickListener() {
2.              @Override
3.              public void onItemClick(AdapterView<?> parent, View view,
4.                      int position, long id) {
```

```
5.                  Object itemAtPosition = mListView.getItemAtPosition(position);
6.                  if (itemAtPosition != null) {
7.                      TaskInfo taskInfo = (TaskInfo) itemAtPosition;
8.                      // 让用户点击当前黑马手机卫士的事件为无效
9.                      if (taskInfo.apkPackageName.equals(getPackageName())) {
10.                         return;
11.                     }
12.                     // 如果当前对象已经勾选。点击之后。变成没有勾选
13.                     // 如果没有勾选。点击之后变成已经勾选
14.                     if (taskInfo.checked) {
15.                         taskInfo.checked = false;
16.                     } else {
17.                         taskInfo.checked = true;
18.                     }
19.                     // 刷新界面
20.                     adater.notifyDataSetChanged();
21.                 }
22.             }
23.         });
```

运行程序，效果图如图 6-12 所示。

图 6-12 进程管理之点击当前进程后按钮未选中效果

6.4.3 进程管理之进程清理

（1）清理按钮的界面 UI

这里，我们在进程管理头标题的右侧定义清理进程的图片按钮，当点击该按钮时将清理界面中被选中的进程。首先，需要修改进程管理界面布局文件中头标题部分的代码，具体如下所示。

```
<RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <TextView
            style="@style/TitleBarTextView"
            android:text="进程管理" />
        <!-- 清理进程 -->
        <ImageButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_margin="10dp"
            android:background="@android:color/transparent"
            android:onClick="killProcess"
            android:src="@drawable/clean_selected" />
    </RelativeLayout>
```

其中，clean_selected.xml 是清理进程图片按钮的背景选择器，具体的代码如文件【6-10】所示。

【文件 6-10】 res/drawable/clean_selected.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- selector 选择器 -->
    <!-- android:state_pressed="true" 手指按压 -->
   <item android:drawable="@drawable/clean_pressed" android:state_pressed="true"></item>
    <!-- 默认 -->
   <item android:drawable="@drawable/clean_normal"></item>
</selector>
```

运行程序，效果图如图 6-13 所示。

图 6-13 进程管理之清理进程按钮的点击效果

（2）清理按钮的功能实现

接下来，我们需要实现清理按钮的点击事件，添加的代码如下所示。

```
1.     /**
2.       * 清理进程
3.       *
4.       * @param view
5.       */
6.      public void killProcess(View view) {
7.          ActivityManager am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
8.          //为了防止造成并发修改异常，这里另外定义了一个需要清理的进程集合
9.          List<TaskInfo> lists = new ArrayList<TaskInfo>();
10.         for (TaskInfo info : userList) {
11.             if (info.checked) {
12.                 lists.add(info);
13.             }
14.         }
15.
16.         for (TaskInfo info : systemList) {
17.             if (info.checked) {
18.                 lists.add(info);
19.             }
20.         }
21.
22.         for (TaskInfo info : lists) {
23.             if (info.isUserTask) {
24.                 userList.remove(info);
25.                 am.killBackgroundProcesses(info.apkPackageName);
26.             } else {
27.                 systemList.remove(info);
28.                 am.killBackgroundProcesses(info.apkPackageName);
29.             }
30.         }
31.         adater.notifyDataSetChanged();
32.     }
```

另外，需要注意的是，清理手机进程需要添加一个权限，如下所示。

```
1.  <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES" />
```

运行程序，点击全选按钮，然后点击右上角的清理进程按钮，效果图如图 6-14 所示。

图 6-14 进程管理之清理进程效果

6.4.4 头标题中进程数的更新

观察之前的效果图，如图 6-15 所示，多选几个进程点击清理之后，发现上方的进程数和占用内存没有变化，这样就会让用户认为清理进程失败。因此，我们需要在清理进程后更新当前的进程数。

图 6-15 锁屏清理时头标题中进程数未更新

这部分的逻辑需要在清理进程的 killProcess 方法中实现，具体代码如下所示。

```
1.      /**清理进程
2.       * @param view
3.       */
4.      public void killProcess(View view) {
5.          ActivityManager am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
6.          List<TaskInfo> lists = new ArrayList<TaskInfo>();
7.          // 表示进程的数量
8.          int count = 0;
```

```
9.          // 表示占用内存
10.         int mem = 0;
11.         for (TaskInfo info : userList) {
12.             if (info.checked) {
13.                 lists.add(info);
14.                 count++;
15.                 mem += info.memSize;
16.             }
17.         }
18.         for (TaskInfo info : systemList) {
19.             if (info.checked) {
20.                 lists.add(info);
21.                 count++;
22.                 mem += info.memSize;
23.             }
24.         }
25.         for (TaskInfo info : lists) {
26.             if (info.isUserTask) {
27.                 userList.remove(info);
28.                 am.killBackgroundProcesses(info.apkPackageName);
29.             } else {
30.                 systemList.remove(info);
31.                 am.killBackgroundProcesses(info.apkPackageName);
32.             }
33.         }
34.         runningProcessCount -= count;
35.         availMem += mem;
36.         userMem -= mem;
37.         // 设置进程个数
38.         pdv_task_count.setTvLeft("正在运行" + runningProcessCount + "个");
39.         // 设置总共有多少个进程
40.         pdv_task_count.setTvRight("总共进程" + runningTotalCount + "个");
41.         // 设置进度
42.         pdv_task_count
43.                 .setProgress((int) (runningProcessCount * 100f / runningTotalCount));
44.         // 设置占用的内存
45.         pdv_ram.setTvLeft("占用内存" + Formatter.formatFileSize(this, userMem));
46.         // 设置可用内存
47.         pdv_ram.setTvRight("可用内存" + Formatter.formatFileSize(this, availMem));
48.         // 设置进度条
49.         pdv_ram.setProgress((int) (userMem * 100f / totalMem));
50.         adater.notifyDataSetChanged();
51.     }
```

运行程序，选择 5 个应用进程进行清理，效果图如图 6-16 所示。

图 6-16 进程清理后头标题中的数据更新

6.5 进程管理之抽屉界面

6.5.1 界面 UI

这里我们将要使用 SlingDrawer 来实现抽屉效果，具体的代码如文件【6-11】所示。

【文件 6-11】 res/layout/activity_task_managerxml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.      android:orientation="vertical" >
7.
8.      <RelativeLayout
9.          android:layout_width="match_parent"
10.         android:layout_height="wrap_content" >
11.         <TextView
12.             style="@style/TitleBarTextView"
13.             android:text="进程管理" />
14.         <!-- 清理进程 -->
15.         <ImageButton
16.             android:layout_width="wrap_content"
17.             android:layout_height="wrap_content"
18.             android:layout_alignParentRight="true"
19.             android:layout_margin="10dp"
20.             android:background="@android:color/transparent"
```

```
21.           android:onClick="killProcess"
22.           android:src="@drawable/clean_selected" />
23.    </RelativeLayout>
24.
25.    <!-- 进程数 -->
26.    <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
27.        android:id="@+id/pdv_task_count"
28.        android:layout_width="match_parent"
29.        android:layout_height="wrap_content"
30.        android:layout_marginTop="10dp" />
31.    <!--内存 rom ：类似电脑上面的一块存储空间
32.             ram ： 代表运行内存
33.    -->
34.    <com.itheima.mobilesafe_sh2.act.view.ProgressDesView
35.        android:id="@+id/pdv_ram"
36.        android:layout_width="match_parent"
37.        android:layout_height="wrap_content" />
38.
39.   <FrameLayout
40.        android:layout_width="match_parent"
41.        android:layout_height="match_parent"
42.        android:layout_weight="1" >
43.        <ListView
44.           android:id="@+id/list_view"
45.           android:layout_width="match_parent"
46.           android:layout_height="wrap_content" >
47.        </ListView>
48.        <!-- 抽屉 -->
49.        <SlidingDrawer
50.           android:id="@+id/slidingDrawer"
51.           android:layout_width="match_parent"
52.           android:layout_height="match_parent"
53.           android:content="@+id/content"
54.           android:handle="@+id/handle" >
55.           <!-- 抽屉的把手 -->
56.           <RelativeLayout
57.              android:id="@+id/handle"
58.              android:layout_width="match_parent"
59.              android:layout_height="wrap_content"
60.              android:background="@drawable/drawer_bg" >
61.              <LinearLayout
62.                 android:layout_width="match_parent"
63.                 android:layout_height="wrap_content"
```

```
64.                 android:layout_marginTop="20dp"
65.                 android:gravity="center_horizontal"
66.                 android:orientation="vertical" >
67.                 <!-- 抽屉的箭头 -->
68.                 <ImageView
69.                     android:id="@+id/drawer_arrow_up1"
70.                     android:layout_width="wrap_content"
71.                     android:layout_height="wrap_content"
72.                     android:src="@drawable/drawer_arrow_up" />
73.                 <!-- 抽屉的箭头 -->
74.                 <ImageView
75.                     android:id="@+id/drawer_arrow_up2"
76.                     android:layout_width="wrap_content"
77.                     android:layout_height="wrap_content"
78.                     android:layout_below="@id/drawer_arrow_up1"
79.                     android:src="@drawable/drawer_arrow_up" />
80.             </LinearLayout>
81.         </RelativeLayout>
82.
83.         <!-- 抽屉的内容 -->
84.         <LinearLayout
85.             android:id="@+id/content"
86.             android:layout_width="match_parent"
87.             android:layout_height="match_parent"
88.             android:background="#fff"
89.             android:clickable="true"
90.             android:focusable="true"
91.             android:orientation="vertical" >
92.             <TextView
93.                 android:layout_width="match_parent"
94.                 android:layout_height="wrap_content"
95.                 android:layout_marginLeft="10dp"
96.                 android:layout_marginTop="10dp"
97.                 android:text="进程管理设置" />
98.
99.           <com.itheima.mobilesafe_sh2.act.view.SettingItemView
100.                 android:id="@+id/siv_show_system_process"
101.                 android:layout_width="match_parent"
102.                 android:layout_height="wrap_content"
103.                 android:layout_marginTop="10dp"
104.                 itheima:itbackground="first"
105.                 itheima:title="显示系统进程" />
106.             <com.itheima.mobilesafe_sh2.act.view.SettingItemView
107.                 android:id="@+id/siv_screen_lock_process"
```

```
108.                    android:layout_width="match_parent"
109.                    android:layout_height="wrap_content"
110.                    itheima:itbackground="last"
111.                    itheima:title="锁屏自动清理" />
112.            </LinearLayout>
113.        </SlidingDrawer>
114.        <ProgressBar
115.            android:id="@+id/pb"
116.            android:layout_width="wrap_content"
117.            android:layout_height="wrap_content"
118.            android:layout_gravity="center"
119.            android:indeterminateDrawable="@drawable/progress_medium"
120.            android:visibility="invisible" />
121.    </FrameLayout>
122.
123.    <LinearLayout
124.        android:layout_width="match_parent"
125.        android:layout_height="wrap_content"
126.        android:orientation="horizontal" >
127.        <Button
128.            android:layout_width="0dp"
129.            android:layout_height="wrap_content"
130.            android:layout_weight="1"
131.            android:background="@drawable/dg_button_cancel_selected"
132.            android:onClick="allSelect"
133.            android:text="全选" />
134.        <Button
135.            android:layout_width="0dp"
136.            android:layout_height="wrap_content"
137.            android:layout_weight="1"
138.            android:background="@drawable/dg_button_cancel_selected"
139.            android:onClick="oppsiteSelect"
140.            android:text="反选" />
141.    </LinearLayout>
142.  </LinearLayout>
```

运行程序，效果图如图 6-17 所示。

图 6-17 进程管理之抽屉效果

6.5.2 抽屉动画

（1）抽屉把手上的渐变动画

为了让抽屉的页面看着更加美观，我们需要给抽屉的把手上面的那两个箭头图片添加一个动画，首先需要在 initView( )中初始化 View 对象，代码如下所示。

```
1.  private void initView() {
2.          ........
3.          // 箭头 1
4.          drawer_arrow_up1 = (ImageView) findViewById(R.id.drawer_arrow_up1);
5.          // 箭头 2
6.          drawer_arrow_up2 = (ImageView) findViewById(R.id.drawer_arrow_up2);
7.          // 初始化一个向上的动画
8.          initUpAnimator();
9.      }
10.     /**
11.      * 箭头向上的动画
12.      */
13.     private void initUpAnimator() {
14.         // 初始化动画的图片
15.         drawer_arrow_up1.setImageResource(R.drawable.drawer_arrow_up);
16.         drawer_arrow_up2.setImageResource(R.drawable.drawer_arrow_up);
17.         // 初始化透明度动画
18.         AlphaAnimation animation1 = new AlphaAnimation(0.2f, 1.0f);
19.         // 设置动画的次数
20.         // 虚线循环
21.         animation1.setRepeatCount(AlphaAnimation.INFINITE);
22.         // 设置动画时间
```

```
23.          animation1.setDuration(800);
24.          drawer_arrow_up1.startAnimation(animation1);
25.          // 初始化透明度动画
26.          AlphaAnimation animation2 = new AlphaAnimation(1.0f, 0.2f);
27.          // 设置动画的次数
28.          // 虚线循环
29.          animation2.setRepeatCount(AlphaAnimation.INFINITE);
30.          // 设置动画时间
31.          animation2.setDuration(800);
32.          drawer_arrow_up2.startAnimation(animation2);
33.      }
```

运行程序，观察效果图，如图 6-18 所示。

图 6-18 抽屉动画效果

（2）抽屉打开与关闭状态的监听

观察完整的效果图可知，当抽屉打开时，把手上图片的动画效果是消失的，这就需要对抽屉的开关状态需要监听，然后做出对应的处理，代码如下所示。

首先，拿到抽屉 View 对象，代码放在 iniview()中，如下所示。

```
1.          // 抽屉
2.          slidingDrawer = (SlidingDrawer) findViewById(R.id.slidingDrawer);
```

然后，在初始化监听的 initListener()方法中，添加抽屉的开关状态的监听事件，具体代码如下所示。

```
1.          // 打开抽屉
2.          slidingDrawer.setOnDrawerOpenListener(new OnDrawerOpenListener() {
3.              @Override
4.              public void onDrawerOpened() {
5.                  initDownAnimator();
6.              }
7.          });
```

```
8.          // 关闭抽屉
9.          slidingDrawer.setOnDrawerCloseListener(new OnDrawerCloseListener() {
10.             @Override
11.             public void onDrawerClosed() {
12.                 initUpAnimator();
13.             }
14.         });
```

上述代码中，initDownAnimator()是当打开抽屉的时候，图片箭头向下的动画效果，代码如下所示。

```
1.      /**
2.       * 箭头向下的动画
3.       */
4.      protected void initDownAnimator() {
5.          // 去掉动画
6.          drawer_arrow_up1.clearAnimation();
7.          drawer_arrow_up2.clearAnimation();
8.          drawer_arrow_up1.setImageResource(R.drawable.drawer_arrow_down);
9.          drawer_arrow_up2.setImageResource(R.drawable.drawer_arrow_down);
10.     }
```

另外，initUpAnimator()是之前已经定义的，当关闭抽屉时图片箭头向上的动画效果。

运行程序，效果图如图 6-19 所示。

图 6-19 抽屉打开与关闭时的动画效果

6.6 进程管理之抽屉界面功能实现

6.6.1 显示系统进程选项的状态更新

抽屉界面中，有一个显示系统进程的选项，它的开关决定了进程管理界面是否显示系统进程的信息，这里首先我们需要拿到该选项的点击事件，改变右边按钮的选中状态，最后需要处理对应的回显状态展示。

在 iniview()拿到对应的 View 对象，并在初始化监听 initListener( )方法中实现点击事件，改变按钮状态，并使用 SharedPreferences 保存当前的选中状态，具体的代码如下所示。

```
1.          // 展示系统进程
2.          siv_show_system_process.setOnClickListener(new OnClickListener() {
3.              @Override
4.              public void onClick(View v) {
5.                  // 如果开关是打开的。那么就关闭
6.                  // 如果开关是关闭的。那么就打开
7.                  if (siv_show_system_process.isToggle()) {
8.                      siv_show_system_process.setToggle(false);
9.                  } else {
10.                     siv_show_system_process.setToggle(true);
11.                 }
12.                 // 保存是否展示系统进程的标记
13.                 SharedPreferencesUtils.saveBoolean(TaskManagerActivity.this,
14.                 Constants.SHOW_SYSTEM_PROCESS,siv_show_system_process.isToggle());
15.             }
16.         });
```

其中，SHOW_SYSTEM_PROCESS 是用于 Constants 常量类中标记系统进程的常量。

然后，在初始化 View 的 initView( )方法中回显当前按钮的状态，具体的代码如下所示。

```
1    // 显示系统进程
2   siv_show_system_process = (SettingItemView) findViewById(R.id.siv_show_system_process);
3    // 返回是否展示系统进程
4   boolean isShowSystemProcess = SharedPreferencesUtils.getBoolean(this,
5               Constants.SHOW_SYSTEM_PROCESS, false);
6   if (isShowSystemProcess) {
7      siv_show_system_process.setToggle(true);
8    } else {
9      siv_show_system_process.setToggle(false);
10    }
```

运行程序，效果图如图 6-20 所示。

图 6-20 抽屉打开与关闭时显示系统进程按钮的回显效果

6.6.2 显示系统进程选项的功能实现

当用户点击显示系统进程状态，即开启状态时，需要显示当前手机中的系统进程，反之不显示。对应的代码我们需要在数据适配器的 getCount( )方法中实现，具体如下所示。

```
1.          @Override
2.          public int getCount() {
3.              // 返回缓存的数据。是否展示系统进程
4.              boolean result = SharedPreferencesUtils.getBoolean(
5.                      TaskManagerActivity.this, Constants.SHOW_SYSTEM_PROCESS,
6.                      false);
7.              // 如果当前的值为 true。那么就需要展示系统进程。如果是 false。那么不展示系统进程
8.              if (result) {
9.                  return userList.size() + 1 + systemList.size() + 1;
10.             } else {
11.                 return userList.size() + 1;
12.             }
13.         }
```

运行程序，效果图如图 6-21 所示。

图 6-21 抽屉之显示系统进程效果图

6.6.3 锁屏自动清理选项的状态更新

当设置界面中的锁屏清理进程按钮开启时，就会打开进程清理的服务，在该服务中注册了监听屏幕锁屏的广播接收者，当屏幕锁屏时该广播接收到屏幕锁屏的消息后会自动清理进程。

首先，在初始化监听 initListener( )方法中实现监听锁屏自动清理的点击事件，并创建一个服务类 LockScreenService 用于注册清理系统进程，具体代码如下所示。

```
1.          // 锁屏清理进程
2.          siv_screen_lock_process.setOnClickListener(new OnClickListener() {
3.              @Override
4.              public void onClick(View v) {
5.                  Intent intent = new Intent(TaskManagerActivity.this,
```

```
                        LockScreenService.class);
                // 是否开启服务
                if (ServiceStateUtils.serviceRunning(TaskManagerActivity.this,
                        LockScreenService.class)) {
                    siv_screen_lock_process.setToggle(false);
                    stopService(intent);
                } else {
                    siv_screen_lock_process.setToggle(true);
                    startService(intent);
                }
            }
        });
```

考虑到锁屏自动清理选项的回显状态，这里我们在 TaskManagerActivity 的 onStart( )方法中进行处理，具体代码如下所示。

```
    @Override
    protected void onStart() {
        // TODO Auto-generated method stub
        super.onStart();
        // 是否开启锁屏的服务
        boolean result = ServiceStateUtils.serviceRunning(this,
                LockScreenService.class);
       //回显是否开启锁屏的服务
        if (result) {
            siv_screen_lock_process.setToggle(true);
        } else {
            siv_screen_lock_process.setToggle(false);
        }
    }
```

注意，需要在清单文件中注册服务，运行程序，点击锁屏自动清理选项，效果图如图 6-22 所示。

图 6-22 抽屉之锁屏自动清理状态的更新效果图

6.6.4 锁屏自动清理选项的功能实现

上面我们创建了一个服务 LockScreenService，在该服务中我们注册锁屏的广播并进行进程清理操作，具体的代码如文件【6-12】所示。

【文件 6-12】 com.itheima.mobilesafe_sh2.service/LockScreenService.java

```
1.  public class LockScreenService extends Service {
2.      @Override
3.      public IBinder onBind(Intent intent) {
4.          // TODO Auto-generated method stub
5.          return null;
6.      }
7.
8.      /**
9.       * 锁屏清理进程
10.      * @author mwqi
11.      */
12.     private class LockScreenReceiver extends BroadcastReceiver {
13.         @Override
14.         public void onReceive(Context context, Intent intent) {
15.             System.out.println("屏幕被锁定了。。。。。。。");
16.             ActivityManager am = (ActivityManager)getSystemService(Context.
17.                     ACTIVITY_SERVICE);
18.             List<RunningAppProcessInfo> runningAppProcesses = am
19.                     .getRunningAppProcesses();
20.             for (RunningAppProcessInfo runningAppProcessInfo : runningAppProcesses) {
21.                 am.killBackgroundProcesses(runningAppProcessInfo.processName);
22.             }
23.         }
24.     }
25.
26.     @Override
27.     public void onCreate() {
28.         // TODO Auto-generated method stub
29.         super.onCreate();
30.         // 初始化一个锁屏的广播
31.         LockScreenReceiver receiver = new LockScreenReceiver();
32.         // 初始化一个锁屏的 action
33.         // Intent.ACTION_SCREEN_OFF 锁屏的动作
34.         // Intent.ACTION_SCREEN_ON 解锁
35.         IntentFilter filter = new IntentFilter(Intent.ACTION_SCREEN_OFF);
36.         // 注册锁屏的广播
37.         registerReceiver(receiver, filter);
38. //      // 初始化一个定时器
```

```
39. //      // Timer timer = new Timer();
40. //      // TimerTask task = new TimerTask() {
41. //      // @Override
42. //      // public void run() {
43. //      // System.out.println("哥被调用了。。。。。");
44. //      // }
45. //      // };
46. //      // // 定时调度，每隔多长时间调用一次
47. //      // // 每隔 2 秒钟调用一次
48. //      // // 第一个参数：定时任务
49. //      // // 第二个参数：从什么时候开始
50. //      // // 第三个参数：多长时间调用一次
51. //      // timer.schedule(task, 0, 2000);
52.     }
53.     @Override
54.     public void onDestroy() {
55.         // TODO Auto-generated method stub
56.         super.onDestroy();
57.         unregisterReceiver(receiver);
58.         receiver = null;
59.     }
60. }
61.
```

- 第 11~20 行创建了一个锁屏的广播接收者 LockScreenReceiver ，在该广播接收者中获取正在运行的进程，并将其杀死。
- 第 28~35 行的 onCreate()方法中动态的注册了一个广播接收者，用于接收屏幕关闭的的广播。
- 第 52~56 行的 onDestroy()方法中将广播接收者注销并设置为 null。

其中，Timer 是一个定时器，它可以实现调度让某个任务在多少时间之后执行，并在多长时间之后再次调用。

接下来在 AndroidManifest.xml 文件中配置清理进程服务，具体代码如下所示：

```
<!-- 锁屏自动清理进程 -->
<service
    android:name= "cn.itcast.mobliesafe.chapter07.service.AutoKillProcessService" >
</service>
```

运行程序，开启锁屏清理，然后按 F7 让模拟器锁屏，观察 Logcat 输出信息，效果图如图 6-23 所示。

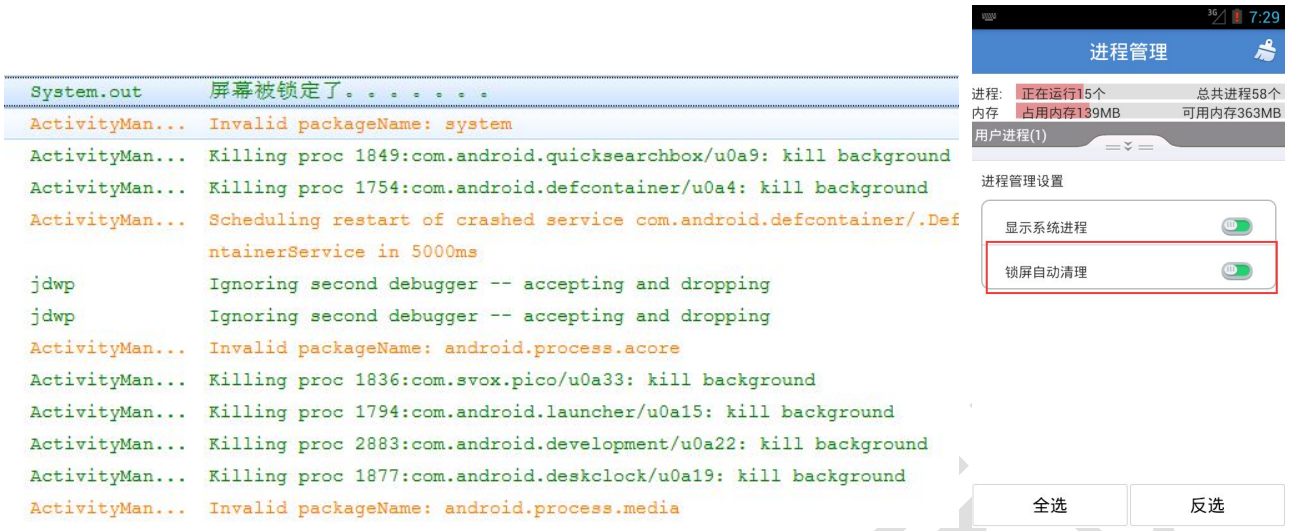

图 6-23 锁屏清理功能实现

6.7 本章小结

本章主要针对进程管理模块进行讲解，首先针对该模块功能进行介绍，然后讲解如何将进程信息展示到 ListView 中以及需要使用的工具类，最后讲解了如何设置是否显示系统进程以及锁屏清理进程。通过该模块的学习编程者可以更熟练的掌握如何管理系统中的进程以及杀死进程。