宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Beta 0.01 版

By 阳哥

Android 手机卫士-08 进程管理&Widget

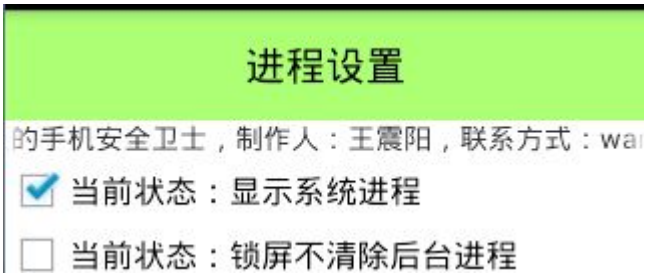
1. 进程管理（★★★★）

1.1 进程管理简介

在应用的主界面点击进程管理图标，进入如下图所示的进程管理主界面。

该界面列出了所有的用户进程和系统进程。我们点击某个进程，那么就会选中该进程，然后可以点击界面下方的“一键清理”按钮将该进程清除掉。

点击进入该界面最下面的设置按钮，进入如下所示进程设置界面，在该界面可以设置是否显示系统进程已经锁屏时是否清除后天进程等选项。



1.2 进程管理界面布局的实现

我们将进程管理的布局分为三大部分，这三个部分放在一个大的线性布局中。

1）最顶端用于运行中进程总数和剩余/总内存大小的部分，该部分是相对布局。

2）中间面积最大用于显示进程的部分，该部分其实是帧布局里面嵌套线性布局，之所以用帧布局是因为我们的进程列表并不是该界面一打开马上就能显示出来，这是一个耗时操作，因此在加载的时候需要显示进度条，数据更新完以后再显示列表

3）最下面的四个按钮是放在线性布局中的

布局文件名为 activity_task_manager.xml，清单如下：

```
<?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:layout_width="match_parent"
        android:layout_height="55dp"
        android:background="#8866ff00"
        android:gravity="center"
        android:text="进程管理"
        android:textColor="#000000"
        android:textSize="20sp" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/tv_run_process"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:text="运行的进程: " />

        <TextView
            android:id="@+id/tv_rom_avail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:text="剩余/总内存: " />
    </RelativeLayout>

    <FrameLayout
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <ListView
            android:id="@+id/lv_task_item"
```

```
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

         <TextView
            android:id="@+id/tv_list_status"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#888888"
            android:text="用户程序(8)"
            android:textColor="#ffffff" />

        <LinearLayout
            android:id="@+id/ll_loading"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:orientation="vertical" >

            <ProgressBar
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="数据加载中" />
        </LinearLayout>
    </FrameLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        >
        <Button
            android:textSize="14sp"
            android:background="@drawable/button"
            android:text="全选"
            android:onClick="selectAll"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            />
```

```
        <Button
            android:textSize="14sp"
            android:background="@drawable/button"
            android:text="反选"
            android:onClick="unSelect"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            />
        <Button
            android:textSize="14sp"
            android:background="@drawable/button"
            android:text="一键清理"
            android:onClick="clearAll"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            />
        <Button
            android:textSize="14sp"
            android:background="@drawable/button"
            android:text="设置"
            android:onClick="reEnterSetting"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            />
    </LinearLayout>

</LinearLayout>
```

1.3 进程设置界面布局的实现

该界面布局比较简单，布局文件名 activity_task_manager_setting.xml，清单如下：

```
<?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:layout_width="match_parent"
        android:layout_height="55dp"
        android:background="#8866ff00"
        android:gravity="center"
        android:text="进程设置"
        android:textColor="#000000"
        android:textSize="20sp" />

    <CheckBox
        android:id="@+id/cb_task_manager_setting"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="当前状态：显示系统进程"
        />
    <CheckBox
        android:id="@+id/cb_task_manager_killprocess"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="锁屏时：清除后台进程"
        />
</LinearLayout>
```

1.4 进程管理业务逻辑的实现

Tips：进程管理涉及到的知识点清单如下，这些知识点的使用方法在代码中都会出现。

1）获取系统可用内存空间

```
public static long getAvailRam(Context context) {
      ActivityManager activityManager = (ActivityManager)
context.getSystemService(Context.ACTIVITY_SERVICE);
      MemoryInfo outInfo = new MemoryInfo();
      activityManager.getMemoryInfo(outInfo);
      return outInfo.availMem;
   }
```

2）获取系统总内存空间

```
public static long getTotalRam(Context context) {
       File file = new File("/proc/meminfo");
       StringBuilder sb = new StringBuilder();
       try {
          BufferedReader reader = new BufferedReader(new FileReader(file));
          String string = reader.readLine();
          for (char c : string.toCharArray()) {
             if (c >= '0' && c <= '9') {
                sb.append(c + "");
             }
          }
          reader.close();
       } catch (Exception e) {
          System.err.println(e);
          return 0;
       }
       return Long.valueOf(sb.toString()) * 1024;
    }
```

3）获取系统中运行的所有进程

```
ActivityManager activityManager = (ActivityManager)
context.getSystemService(context.ACTIVITY_SERVICE);
List<RunningAppProcessInfo>                runningAppProcesses                =
activityManager.getRunningAppProcesses();
```

4）杀死进程

```
activityManager.killBackgroundProcesses(info.getPackName());
```

5）在 ListView 中显示状态

1.4.1 TaskManagerActivity.java

进程管理对应的 Activity 为 TaskManagerActivity.java，代码清单如下：

```
/**
 * 对应进程管理主界面
 */
public class TaskManagerActivity extends Activity {
```

```
    // 用于显示系统可用存储空间
    private TextView tv_rom_avail;
    // 用于显示系统中运行的进程数
    private TextView tv_run_process;
    // 声明个 ActivityManager 引用，这个引用也是管理 Activity 的核心类
    private ActivityManager activityManager;
    private ListView lv_task_item;
    private LinearLayout ll_loading;
    private List<TaskInfo> taskinfos;
    private List<TaskInfo> usertaskinfos;
    private List<TaskInfo> systemtaskinfos;
    private TextView tv_list_status;
    // 自定义的 ListView 的 Adapter
    private TaskManagerAdapter adapter;
    private SharedPreferences sp;
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            // 接收到消息后代表 ListView 数据已经初始化完成，因此隐藏进度条
            ll_loading.setVisibility(View.INVISIBLE);
            adapter = new TaskManagerAdapter();
            // 给 ListView 设置 Adapter
            lv_task_item.setAdapter(adapter);
        };
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_task_manager);
        sp = getSharedPreferences("config", MODE_PRIVATE);
        // 初始化控件对象
        tv_list_status = (TextView) findViewById(R.id.tv_list_status);
        tv_rom_avail = (TextView) findViewById(R.id.tv_rom_avail);
        tv_run_process = (TextView) findViewById(R.id.tv_run_process);
        // 通过自定义工具类获取运行中的进程
        tv_run_process.setText("运行中的进程(" +
SystemInfoUtil.getRunProgressCount(this) + ")");
        // 获取可用内存和总内存
        String avail = Formatter.formatFileSize(this,
SystemInfoUtil.getAvailRam(this));
        String total = Formatter.formatFileSize(this,
SystemInfoUtil.getTotalRam(this));
        // 显示可用内存使用信息
```

```
        tv_rom_avail.setText("剩余/总内存: " + avail + "/" + total);
        lv_task_item = (ListView) findViewById(R.id.lv_task_item);
        ll_loading = (LinearLayout) findViewById(R.id.ll_loading);
        // 获取 ActivityManager 对象
        activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
        // 给 ListView 控件填充数据，获取数据是耗时操作，因此该函数里面的代码放在子线程中进行
        fillData();
        // 给 ListView 设置条目的点击事件
        lv_task_item.setOnItemClickListener(new OnItemClickListener() {

            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
                // 获取点击的对象
                Object obj = lv_task_item.getItemAtPosition(position);
                // 判断点击的对象类型 这里只对 TaskInfo 类型进行处理
                if (obj instanceof TaskInfo) {
                    TaskInfo info = (TaskInfo) obj;
                    // 如果被选中则不选，如果没选中则选中
                    CheckBox cb = (CheckBox) view.findViewById(R.id.cb_task_status);
                    if (info.getPackName().equals(getPackageName())) {
                        return;
                    }
                    if (info.isChecked()) {
                        info.setChecked(false);
                        cb.setChecked(false);
                    } else {
                        cb.setChecked(true);
                        info.setChecked(true);
                    }
                }
            }

        });
        // 给 ListView 设置滚动监听
        lv_task_item.setOnScrollListener(new OnScrollListener() {

            @Override
            public void onScrollStateChanged(AbsListView view, int scrollState) {
            }
```

```
        @Override
        public void onScroll(AbsListView view, int firstVisibleItem, int
visibleItemCount, int totalItemCount) {
            if (systemtaskinfos == null || usertaskinfos == null) {
                return;
            }
            // 如果 ListView 中第一个显示的脚标>用户进程的中长度则在 TextView 中显示系
统进程状态，否则显示用户进程状态
            if (firstVisibleItem > usertaskinfos.size()) {
                tv_list_status.setText("系统进程(" + systemtaskinfos.size() + ")");
            } else {
                tv_list_status.setText("用户进程(" + usertaskinfos.size() + ")");
            }
        }
    });
}

// 给 ListView 初始化数据
private void fillData() {
    new Thread(new Runnable() {

        @Override
        public void run() {
            try {
                // 获取所有的进程信息，将这些进程信息氛围用户进程和系统进程
                taskinfos =
TaskInfoProvider.getTaskinfos(TaskManagerActivity.this);
                usertaskinfos = new ArrayList<TaskInfo>();
                systemtaskinfos = new ArrayList<TaskInfo>();
                for (TaskInfo info : taskinfos) {
                    if (info.isUser()) {
                        usertaskinfos.add(info);
                    } else {
                        systemtaskinfos.add(info);
                    }
                }
                // 数据初始化完成以后通过 handler 发送消息
                handler.sendEmptyMessage(0);
            } catch (NameNotFoundException e) {
                System.out.println("加载数据错误" + e);
            }
        }
    }).start();
```

```
    }

    /*
     * 自定义 BaseAdapter 类
     */
    class TaskManagerAdapter extends BaseAdapter {

        @Override
        public int getCount() {
            //根据系统设置查看是否需要显示系统进程
            boolean showSystem = sp.getBoolean("showSystem", true);
            if (showSystem) {
                return taskinfos.size() + 2;
            } else {
                return usertaskinfos.size() + 1;
            }
        }

        @Override
        public Object getItem(int position) {
            TaskInfo taskInfo = null;
            //ListView 的第一个条目和第用户进程数+1 个条目返回 null，因为这些脚标位置放置
的是 TextView 对象，我们不需要使用，因此直接返回 null 就行了
            if (position == 0) {
                return null;
            } else if (position == usertaskinfos.size() + 1) {
                return null;
            } else if (position <= usertaskinfos.size()) {
                //返回用户进程的对象，之所以减 1 是因为 ListView 第一个位置是 TextView
                taskInfo = usertaskinfos.get(position - 1);
            } else {
                //返回系统进程对象，之所以减 2 是因为 ListView 第一个和系统进程前分别有个
TextView
                taskInfo = systemtaskinfos.get(position - usertaskinfos.size() - 2);
            }
            return taskInfo;
        }

        @Override
        public long getItemId(int position) {
            return 0;
        }
```

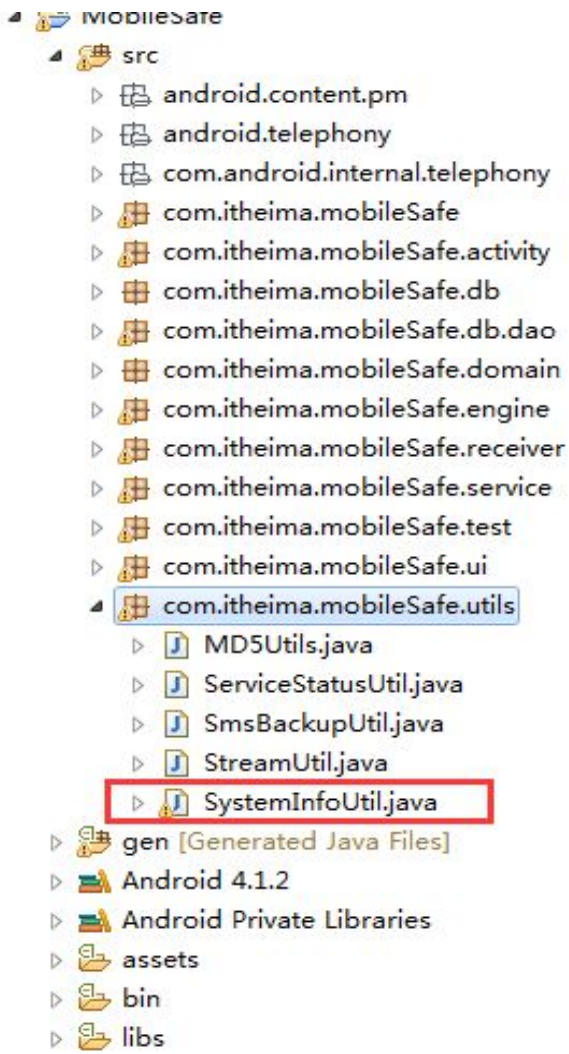
```
        /**
         * ListView
         */
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            TaskInfo taskInfo = null;
            //如果 是 ListView 的第一个位置显示 TextView 对象，在 TextView 中显示用户进程总
数
            if (position == 0) {
                TextView textView = new TextView(getApplicationContext());
                textView.setText("用户进程(" + usertaskinfos.size() + ")");
                textView.setBackgroundColor(Color.GRAY);
                return textView;
            } else if (position == usertaskinfos.size() + 1) {
                //用户进程数+1 处用 TextView 显示系统进程总数
                TextView textView = new TextView(getApplicationContext());
                textView.setText("系统进程(" + systemtaskinfos.size() + ")");
                textView.setBackgroundColor(Color.GRAY);
                return textView;
            } else if (position <= usertaskinfos.size()) {
                taskInfo = usertaskinfos.get(position - 1);
            } else {
                taskInfo = systemtaskinfos.get(position - usertaskinfos.size() - 2);
            }
            if (taskInfo == null) {
                Toast.makeText(getApplicationContext(), "appInfo 为 null", 0).show();
                return convertView;
            }
            //下面的是对 ListView 的优化处理
            View view;
            ViewHolder viewHolder;
            if (convertView != null && convertView instanceof RelativeLayout) {
                view = convertView;
                viewHolder = (ViewHolder) view.getTag();
            } else {
                view = View.inflate(getApplicationContext(), R.layout.list_task_item,
null);
                viewHolder = new ViewHolder();
                viewHolder.iv_icon = (ImageView) view.findViewById(R.id.iv_icon);
                viewHolder.tv_task_memsize = (TextView)
view.findViewById(R.id.tv_task_memsize);
                viewHolder.tv_app_name = (TextView)
view.findViewById(R.id.tv_app_name);
```

```
            viewHolder.cb_task_status = (CheckBox)
view.findViewById(R.id.cb_task_status);
            view.setTag(viewHolder);
         }
         viewHolder.iv_icon.setImageDrawable(taskInfo.getIcon());
         viewHolder.tv_app_name.setText(taskInfo.getName());

   viewHolder.tv_task_memsize.setText(Formatter.formatFileSize(TaskManagerActivi
ty.this, taskInfo.getMemesize()));
         viewHolder.cb_task_status.setChecked(taskInfo.isChecked());
         return view;
      }

   }

   static class ViewHolder {
      TextView tv_app_name;
      TextView tv_task_memsize;
      ImageView iv_icon;
      CheckBox cb_task_status;
   }
   /**
    * 对应界面全选按钮
    * @param view
    */
   public void selectAll(View view) {
      //逻辑很简单，就是遍历所有的用户进程和系统进程然后给这些数据设置成选中的属性
      for (TaskInfo info : usertaskinfos) {
         if (info.getPackName().equals(getPackageName())) {
            continue;
         }
         info.setChecked(true);
      }
      for (TaskInfo info : systemtaskinfos) {
         info.setChecked(true);
      }
      //通知 ListView 跟新数据
      adapter.notifyDataSetChanged();
   }
   /**
    * 对应界面反选按钮
    * @param view
    */
```

```
    public void unSelect(View view) {
        for (TaskInfo info : usertaskinfos) {
            if (info.getPackName().equals(getPackageName())) {
                continue;
            }
            info.setChecked(!info.isChecked());
        }
        for (TaskInfo info : systemtaskinfos) {
            info.setChecked(!info.isChecked());
        }
        adapter.notifyDataSetChanged();
    }
    /**
     * 对应界面的一键清理按钮
     * @param view
     */
    public void clearAll(View view) {
        //遍历所有的进程信息，如果被选中，则执行清除进程方法
        for (TaskInfo info : usertaskinfos) {
            if (info.isChecked()) {
                // 杀死后台进程
                activityManager.killBackgroundProcesses(info.getPackName());
            }
        }
        for (TaskInfo info : systemtaskinfos) {
            if (info.isChecked()) {
                // 杀死后台进程
                activityManager.killBackgroundProcesses(info.getPackName());
            }
        }
        //进程被清理后重新给 ListView 初始化数据
        fillData();
    }
    /**
     * 对应界面设置按钮
     * @param view
     */
    public void reEnterSetting(View view) {
        //通过 Intent 打开一个进程管理设置界面
        Intent intent = new Intent(this, TaskManagerSettingActivity.class);
        startActivityForResult(intent, 1);// (intent);
    }
```

```
/**
     * 接收设置 Activity 返回来的消息
     */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode==1) {
            //更新 ListView 数据
            adapter.notifyDataSetChanged();
        }
    }
}
```

Tips：在上面的代码中我们使用到了 SystemInfoUtil.java 工具类，该类是我自己创建的用于获取系统内存相关的信息的。我们将该类创建在工程 src 目录下的 com.itheima.mobileSafe.utils 包下。工程目录如下所示。



1.4.2 SystemInfoUtil.java 工具类

该类的代码清单如下：

```
public class SystemInfoUtil {
    /**
     * 获取运行进程个数
     *
     * @param context
     * @return
     */
    public static int getRunProgressCount(Context context) {
        ActivityManager activityManager = (ActivityManager)
context.getSystemService(Context.ACTIVITY_SERVICE);
        return activityManager.getRunningAppProcesses().size();
    }

    /**
     * 获取系统可用内存
     */
    public static long getAvailRam(Context context) {
        ActivityManager activityManager = (ActivityManager)
context.getSystemService(Context.ACTIVITY_SERVICE);
        MemoryInfo outInfo = new MemoryInfo();
        activityManager.getMemoryInfo(outInfo);
        return outInfo.availMem;
    }

    /**
     * 获取系统总内存 对于 Linux 系统,系统中内存信息时存储在系统的如下目录:/proc/meminfo
     *  因此我们想获取系统中内存只需从该文件中读取信息即可
     *  虽然通过 ActivityManager 的 getMemoryInfo()方法也可以获取总内存大小，但是在低版
本的手机上是没有这个 API 的，因此为了兼容性推荐直接读取 Linux 配置文件
     */
    public static long getTotalRam(Context context) {
        File file = new File("/proc/meminfo");
        StringBuilder sb = new StringBuilder();
        try {
            //meminfo 文件有多行内容，但是总内存大小存储在第一行，因此我们只 readLine 一次
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String string = reader.readLine();
            for (char c : string.toCharArray()) {
```

```
                if (c >= '0' && c <= '9') {
                    sb.append(c + "");
                }
            }
            reader.close();
        } catch (Exception e) {
            System.err.println(e);
            return 0;
        }
        return Long.valueOf(sb.toString()) * 1024;
    }
}
```

Tips：Linux 系统 meminfo 文件内容：

```
MemTotal:         507480 kB
MemFree:           10800 kB
Buffers:           34728 kB
Cached:            98852 kB
SwapCached:          128 kB
Active:           304248 kB
Inactive:          46192 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         507480 kB
LowFree:           10800 kB
SwapTotal:        979956 kB
SwapFree:         941296 kB
```

```
Dirty:                32 kB
Writeback:             0 kB
AnonPages:        216756 kB
Mapped:            77560 kB
Slab:              22952 kB
SReclaimable:      15512 kB
SUnreclaim:         7440 kB
PageTables:         2640 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
CommitLimit:     1233696 kB
Committed_AS:     828508 kB
VmallocTotal:     516088 kB
```

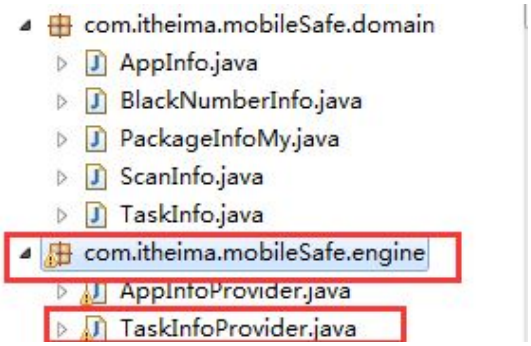
部分属性含义如下（拓展，非必要掌握知识）：

- MemTotal: 所有可用 RAM 大小 （即物理内存减去一些预留位和内核的二进制代码大小）
- MemFree: LowFree 与 HighFree 的总和，被系统留着未使用的内存
- Buffers: 用来给文件做缓冲大小
- Cached: 被高速缓冲存储器（cache memory）用的内存的大小（等于 diskcache minus SwapCache ）.
- SwapCached:被高速缓冲存储器（cache memory）用的交换空间的大小已经被交换出来的内存，但仍然被存放在 swapfile 中。用来在需要的时候很快的被替换而不需要再次打开 I/O 端口。
- Active: 在活跃使用中的缓冲或高速缓冲存储器页面文件的大小，除非非常必要否则不会被移作他用.

- Inactive: 在不经常使用中的缓冲或高速缓冲存储器页面文件的大小，可能被用于其他途径.
- HighFree: 该区域不是直接映射到内核空间。内核必须使用不同的手法使用该段内存。
- LowFree: 低位可以达到高位内存一样的作用，而且它还能够被内核用来记录一些自己的数据结构。
- SwapTotal: 交换空间的总大小
- SwapFree: 未被使用交换空间的大小
- Dirty: 等待被写回到磁盘的内存大小。
- Writeback: 正在被写回到磁盘的内存大小。
- AnonPages：未映射页的内存大小
- Mapped: 设备和文件等映射的大小。
- Slab: 内核数据结构缓存的大小，可以减少申请和释放内存带来的消耗。
- SReclaimable:可收回 Slab 的大小
- SUnreclaim：不可收回 Slab 的大小（SUnreclaim+SReclaimable = Slab）
- PageTables：管理内存分页页面的索引表的大小。
- NFS_Unstable:不稳定页表的大小

1.4.3 TaskInfoProvider.java

在 TaskManagerActivity.java 中我们用到了 TaskInfoProvider 类，该类用于封装进程信息。在 com.itheima.mobileSafe.engine 包下，目录结构如下图：



TaskInfoProvider.java 的代码清单如下：

```
/**
 * 用于获取进程信息
 *
 */
public class TaskInfoProvider {
   public static List<TaskInfo> getTaskinfos(Context context) throws
NameNotFoundException {
      List<TaskInfo> taskInfos = new ArrayList<TaskInfo>();
      // 获取 ActivityManager 对象
      ActivityManager activityManager = (ActivityManager)
context.getSystemService(Context.ACTIVITY_SERVICE);
      // 获取运行中的进程
      List<RunningAppProcessInfo> runningAppProcesses =
activityManager.getRunningAppProcesses();
      PackageManager pm = context.getPackageManager();
      // 遍历 RunningAppProcessInfo 将从 RunningAppProcessInfo 对象中获取的对象封装在
TaskInfo 中
      for (RunningAppProcessInfo info : runningAppProcesses) {

         TaskInfo taskInfo = new TaskInfo();
         // 获取 pid
         String packname = info.processName;
         try {
            // 获取进程 ID
            int[] pids = { info.pid };
            //通过 ActivityManager 获取进程内存信息，返回的是数组，数组中每个信息对应一
个 pid，这里只取第一个对象，因为在通常情况下一个应用只有一个 pid
            MemoryInfo processMemoryInfo =
activityManager.getProcessMemoryInfo(pids)[0];
            /*
             * Return total private dirty memory usage in kB.
             * 返回该应用使用的内存，单位是 kb，我们乘以 1024 转换成 byte。
             */
            long memesize = processMemoryInfo.getTotalPrivateDirty() * 1024;
            //通过 PackageManager 根据报名信息获取 PackageInfo 对象
            PackageInfo packageInfo = pm.getPackageInfo(packname, 0);
            //获取 icon
            Drawable icon = packageInfo.applicationInfo.loadIcon(pm);
            //获取应用名称
            String name = packageInfo.applicationInfo.loadLabel(pm).toString();
            taskInfo.setMemesize(memesize);
```

```
            taskInfo.setIcon(icon);
            taskInfo.setPackName(packname);
            taskInfo.setName(name);
            /*
             * 通过 applicationInfo.flags 判断该应用是否属于系统应用，进行的是&运算，如果当 flags 跟 ApplicationInfo.FLAG_SYSTEM 进行&
             * 运算为 1 时则代表属于系统进程，否则属于用户进程
             */

            if ((packageInfo.applicationInfo.flags & ApplicationInfo.FLAG_SYSTEM) == 0) {
                // 用户进程
                taskInfo.setUser(true);
            } else {
                taskInfo.setUser(false);
            }
            taskInfos.add(taskInfo);
        } catch (Exception e) {
            taskInfo.setName(packname);

    taskInfo.setIcon(context.getResources().getDrawable(R.drawable.app));
            taskInfos.add(taskInfo);
        }
      }
      return taskInfos;
   }
}
```

1.4.4 TaskInfo.java

在上面的代码中我们将进程信息封装在 TaskInfo 类中，该类位于 com.itheima.mobileSafe.domain 包下。清单如下：

```
public class TaskInfo {
   private boolean isChecked;
   private Drawable icon;
   private String name;
   private String packName;
```

```
    private boolean isUser;
    private long memesize;//byte
    //省略 setter&getter 方法
```

1.5 进程设置业务逻辑的实现

在进程设置里面我们可以将设置是否显示系统进程选项保存在 SharedPreference 里面，同时在这里我们可以开启锁屏杀死后台进程服务。进程设置的 TaskManagerSettingActivity.java 代码清单如下：

```
public class TaskManagerSettingActivity extends Activity {
    private CheckBox cb_task_manager_setting;
    private SharedPreferences sp;
    private CheckBox cb_task_manager_killprocess;
    private Intent killProceeIntent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_task_manager_setting);
        cb_task_manager_killprocess = (CheckBox)
findViewById(R.id.cb_task_manager_killprocess);
        sp = getSharedPreferences("config", MODE_PRIVATE);
        cb_task_manager_setting = (CheckBox)
findViewById(R.id.cb_task_manager_setting);
        //从 sp 中获取这两个 Boolean 类型的参数，用于数据回显
        boolean showSystem = sp.getBoolean("showSystem", true);
        boolean killProcess = sp.getBoolean("killProcess", false);
        cb_task_manager_killprocess.setChecked(killProcess);
        cb_task_manager_setting.setChecked(showSystem);
        killProceeIntent = new Intent(this,KillProcessService.class);;
        if (showSystem) {
            cb_task_manager_setting.setText("当前状态：显示系统进程");
        }else {
            cb_task_manager_setting.setText("当前状态：隐藏系统进程");
        }
        //如果开启了锁屏清除进程选项，则启动 KillProcessService 服务,否则关闭
        if (killProcess) {
            cb_task_manager_killprocess.setText("当前状态:锁屏清除后台进程");
            startService(killProceeIntent);
```

```
        }else {
            cb_task_manager_killprocess.setText("当前状态：锁屏不清除后台进程");
            stopService(killProceeIntent);
        }
        //给 CheckBox 设置状态改变监听
        cb_task_manager_setting.setOnCheckedChangeListener(new
OnCheckedChangeListener() {

            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
                //当状态改变时将最近的状态保存在 sp 中，同时修改 checkBox 的文本内容
                Editor editor = sp.edit();
                if (isChecked) {
                    cb_task_manager_setting.setText("当前状态：显示系统进程");
                    editor.putBoolean("showSystem", true);
                } else {
                    editor.putBoolean("showSystem", false);
                    cb_task_manager_setting.setText("当前状态：隐藏系统进程");
                }
                editor.commit();
            }
        });
        cb_task_manager_killprocess.setOnCheckedChangeListener(new
OnCheckedChangeListener() {

            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
                Editor editor = sp.edit();
                if (isChecked) {
                    cb_task_manager_killprocess.setText("当前状态：锁屏清除进程");
                    //如果开启了锁屏清除进程选项，则启动 KillProcessService 服务
                    startService(killProceeIntent);
                    editor.putBoolean("killProcess", true);
                } else {
                    editor.putBoolean("killProcess", false);
                    cb_task_manager_killprocess.setText("当前状态：锁屏不清除进程");
                    //如果关闭了锁屏清除进程选项，则关闭该服务
                    stopService(killProceeIntent);
                }
                editor.commit();
            }
```

```
        });
    }
}
```

Tips：上面我们用到了 KillProcessService 类，在该类中我们开启了一个后台服务用于监听屏幕关闭时系统发出的广播，当监听到屏幕关闭以后，根据 SharedPreference 中记录的参数来决定是否杀死后台进程。

1.6 锁屏杀死后台进程

KillProcessService 类位于 com.itheima.mobileSafe.receiver 包下，目录如下图所示。

KillProcessService.java 代码清单如下：

```
public class KillProcessService extends Service {

    private InnerScreenReceiver receiver;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        //自定义 receiver 用于监听屏幕关闭广播
        receiver = new InnerScreenReceiver();
        IntentFilter filter = new IntentFilter();
        //监听屏幕变暗
        filter.addAction(Intent.ACTION_SCREEN_OFF);
        //动态注册广播接收者
        registerReceiver(receiver, filter);
```

```
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        //当该 service 销毁的时候将广播接收者也销毁
        unregisterReceiver(receiver);
        receiver = null;
    }
    //定义一个广播接收者类
    private class InnerScreenReceiver extends BroadcastReceiver{

        @Override
        public void onReceive(Context context, Intent intent) {
            //当接收到广播后将后台进程全部杀掉
            ActivityManager am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
            List<RunningAppProcessInfo> processes = am.getRunningAppProcesses();
            for(RunningAppProcessInfo info : processes){
                am.killBackgroundProcesses(info.processName);
            }
        }
    }
}
```

2. Widget 入门（★★★）

2.1 Widget 简介

App Widget 是应用程序窗口小部件（Widget）是微型的应用程序视图，它可以被嵌入到其它应用程序中（比如桌面）并接收周期性的更新。你可以通过一个 AppWidgetProvider 来发布一个 Widget。

AppWidgetProvider 继承自 BroadcastReceiver，它能接收 widget 相关的广播，例如 widget 的更新、删除、开启和禁用等。

在我们的这个项目中，我们最终的 Widget 效果如下图所示，它显示了当前运行的软件，可用内存等状态信息，还有一个一键清理按钮，点击按钮后可以清除后天进程。

当我们的程序安装好以后，该 widget 并不会自动运行到桌面上，如果想让该 Widget 能够运行必须靠手动将该 widget 拖拽到桌面上。

2.2 Widget 的实现

Widget 的实现总共需要以下步骤：

- 编写布局文件 mywidget.xml（文件名自定义），该布局文件用于 widget 显示界面
- 自定义一个 MyAppWidget(类名自定义)类继承 AppWidgetProvider
- 在 AndroidManifest.xml 中注册 MyAppWidget，因为 AppWidgetProvider 继承了 BroadcastReceiver，

因此本质上说 MyAppWidget 也是一个 BroadcastReceiver，因此需要注册。

Tips：这里注册的广播接收者跟普通的广播稍微不一样，因为我们需要在该广播中设置一个布局文件，如下所示。

```
<receiver android:name="com.itheima.mobileSafe.receiver.MyAppWidget" >
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data
                android:name="android.appwidget.provider"
                android:resource="@xml/example_appwidget_info" />
</receiver>
```

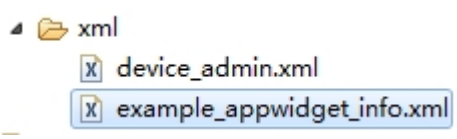
上面的清单文件中`<action android:name="android.appwidget.action.APPWIDGET_UPDATE" />`和

`android:name="android.appwidget.provider"`是由 Android 系统提供的是固定写法。

example_appwidget_info 是关于 widget 配置信息的 xml 文件，文件清单如下所示

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="240dp"
    android:minHeight="72dp"
    android:updatePeriodMillis="1800000"
    android:initialLayout="@layout/mywidget">
</appwidget-provider>
```

该文件在 res->xml 目录中，目录结构如下图所示。



- 如果只有上面的步骤那么我们的 widget 就可以运行了，但是 widget 一般都是需要动态更新的，比如我们的

widget 是需要动态显示当前系统的内存信息的，因此我们还需要在我们的广播中开启一个 service，在 service 中对 widget 进行动态更新。

下面将把上面步骤的详细过程和代码清单展示出来。

2.2.1 widget 布局清单文件

分析该布局，最外层是一个水平的 LinearLayout，里面嵌套了 A、B 两个垂直的 LinearLayout。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:baselineAligned="false"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/widget_bg_portrait"
    android:gravity="center_vertical" >

    <LinearLayout
        android:layout_width="0.0dip"
        android:layout_height="fill_parent"
        android:layout_marginLeft="5.0dip"
        android:layout_weight="1.0"
        android:background="@drawable/widget_bg_portrait_child"
        android:gravity="center_vertical"
        android:orientation="vertical"
        android:paddingBottom="3.0dip"
        android:paddingTop="3.0dip" >

        <TextView
            android:id="@+id/process_count"
            android:text="正在运行的进程12 个"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10.0dip"
            android:textAppearance="@style/widget_text" />
```

```
        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="1.0dip"
            android:layout_marginTop="1.0dip"
            android:background="@drawable/widget_bg_portrait_child_divider" />

        <TextView
            android:text="23M/120M"
            android:id="@+id/process_memory"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10.0dip"
            android:textAppearance="@style/widget_text" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical" >

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center_vertical" >

            <ImageView
                android:layout_width="20.0dip"
                android:layout_height="20.0dip"
                android:src="@drawable/main_icon" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/app_name"
                android:textColor="@color/textColorPrimary" />
        </LinearLayout>

        <Button
            android:id="@+id/btn_clear"
            android:layout_width="90.0dip"
            android:layout_height="wrap_content"
```

```
            android:layout_centerVertical="true"
            android:layout_marginTop="5.0dip"
            android:background="@drawable/function_greenbutton_selector"
            android:text="一键清理"
            android:textColor="@color/function_greenbutton_textcolor_selector" />
    </LinearLayout>
</LinearLayout>
```

上面的 Button 设置了背景色 android:background="@drawable/function_greenbutton_selector"

该北京市一个 xml 文件，文件清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<selector
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
android:drawable="@drawable/function_greenbutton_pressed" />
    <item android:state_focused="true"
android:drawable="@drawable/function_greenbutton_pressed" />
    <item android:drawable="@drawable/function_greenbutton_normal" />
</selector>
```

2.2.2 自定义一个 MyAppWidget 类继承 AppWidgetProvider

MyAppWidget.java 位于 com.itheima.mobileSafe.receiver 包下，代码清单如下所示：

```
public class MyAppWidget extends AppWidgetProvider {
    @Override
    public void onReceive(Context context, Intent intent) {
        super.onReceive(context, intent);
    }
    /**
     * 当 AppWidgetProvider 接收到 ACTION_APPWIDGET_UPDATE（该广播由系统发出）广播时调用
该方法
     */
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[]
appWidgetIds) {
        super.onUpdate(context, appWidgetManager, appWidgetIds);
```

```
        // 开启 widget 后台服务
        Intent intent = new Intent(context, UpdateAppWidgetService.class);
        context.startService(intent);
    }
    /**
     * 当 AppWidgetProvider 接收到 ACTION_APPWIDGET_OPTIONS_CHANGED（该广播由系统发出）
广播时调用该方法
     */
    @SuppressLint("NewApi")
    @Override
    public void onAppWidgetOptionsChanged(Context context, AppWidgetManager
appWidgetManager, int appWidgetId, Bundle newOptions) {
        super.onAppWidgetOptionsChanged(context, appWidgetManager, appWidgetId,
newOptions);
    }
    /**
     * 当 AppWidgetProvider 接收到 ACTION_APPWIDGET_DELETED（该广播由系统发出）广播时调
用该方法
     */
    @Override
    public void onDeleted(Context context, int[] appWidgetIds) {
        super.onDeleted(context, appWidgetIds);
        // 开启 widget 服务
        Intent intent = new Intent(context, UpdateAppWidgetService.class);
        context.stopService(intent);
    }
    /**
     * 当 AppWidgetProvider 接收到 ACTION_APPWIDGET_ENABLED（该广播由系统发出）广播时调
用该方法
     */
    @Override
    public void onEnabled(Context context) {
        super.onEnabled(context);
        // 开启 widget 服务
        Intent intent = new Intent(context, UpdateAppWidgetService.class);
        context.startService(intent);
    }
    /**
     * 当 AppWidgetProvider 接收到 ACTION_APPWIDGET_DISABLED（该广播由系统发出）广播时调
用该方法
     */
```

```
@Override
    public void onDisabled(Context context) {
        super.onDisabled(context);
        // 关闭 widget 服务
        Intent intent = new Intent(context, UpdateAppWidgetService.class);
        context.stopService(intent);
    }
}
```

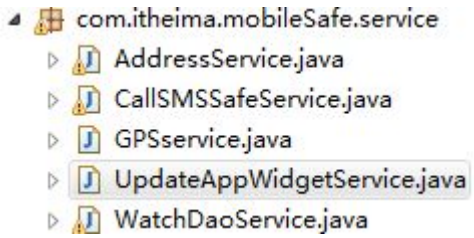
2.2.3 在 AndroidManifest.xml 中注册 MyAppWidget

这一步骤的详细内容在 2.2 节中已经进行了详细的说明，这里就不再重复。之所以引入该节是为了强调该步骤的不可缺失性。

2.2.4 用 Service 动态更新 widget

我们在 MyAppWidget 类的 onUpdate 方法中开启了 UpdateAppWidgetService 在 onDisabled 中关闭了 UpdateAppWidgetService，该服务主要是展示 widget 界面、获取系统内存数据，然后动态更新组件。

UpdateAppWidgetService.java 位于 com.itheima.mobileSafe.service 包写，包结构如下所示。



UpdateAppWidgetService.java 代码清单如下：

```
public class UpdateAppWidgetService extends Service {
    //定义一个定时器对象
    private Timer timer;
    private TimerTask task;
    /**
     * Updates AppWidget state; gets information about installed AppWidget providers
and other
     * AppWidget related state.
```

```
     *
     */
    private AppWidgetManager awm;
    /**
     * 声明锁屏广播接收者，因为在锁屏的时候需要关闭定时任务，这样省电
     */
    private InnerScreenReceiver receiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // 监听锁屏
        receiver = new InnerScreenReceiver();
        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_SCREEN_OFF);
        filter.addAction(Intent.ACTION_SCREEN_ON);
        registerReceiver(receiver, filter);
        //获取 AppWidgetManager 对象
        awm = AppWidgetManager.getInstance(this);
        startUpdateAppWidget();
    }

    private void startUpdateAppWidget() {
        timer = new Timer();
        task = new TimerTask() {

            @Override
            public void run() {
                /*
                 * Create a new component identifier from a Context and Class object.
                 */
                ComponentName provider = new ComponentName(UpdateAppWidgetService.this,
MyAppWidget.class);
                /*
                 * A class that describes a view hierarchy that can be displayed in another
process.
                 * The hierarchy is inflated from a layout resource file, and this class
provides
```

```
             * some basic operations for modifying the content of the inflated
hierarchy.
             * 该类描述了一个视图树用于显示在其他进程中（我的 widget 是运行在其他进程中
的，而不是当前应用的进程中）。这个视图
             * 是从一个 xml 布局文件中填充出来的，并且这个类提供了一些基本操作用来修改我
们视图的视图
             */
            RemoteViews views = new RemoteViews(getPackageName(),
R.layout.mywidget);
            /*
             * 给 widget 界面的 TextView 设置文本内容，第一个参数是该 TextView 的 id，第
二个是要显示的文本内容
             *
             */
            views.setTextViewText(R.id.process_count, "正在运行的软件：" +
SystemInfoUtil.getRunProgressCount(UpdateAppWidgetService.this));
            views.setTextViewText(R.id.process_memory, "可用的内存：" +
Formatter.formatFileSize(UpdateAppWidgetService.this,
SystemInfoUtil.getAvailRam(UpdateAppWidgetService.this)));
            /*
             * 创建一个 Intent,该意图用于发送一个广播，发送的广播被一个广播接收者接收，
接收以后进行了一键清理操作
             */
            Intent intent = new Intent();
            intent.setAction("com.itheima.mobileSafe.killProcess");
            intent.addCategory("android.intent.category.DEFAULT");
            /**
             * A description of an Intent and target action to perform with it.
Instances
             * of this class are created with {@link #getActivity}, {@link
#getActivities},
             * {@link #getBroadcast}, and {@link #getService}; the returned object
can be
             * handed to other applications so that they can perform the action you
             * described on your behalf at a later time.
             * 等待意图或者叫延时意图，当某个事件触发时才使该意图中的 Intent 去执行，这里
是发送一个广播
             */
            PendingIntent pendingIntent =
PendingIntent.getBroadcast(UpdateAppWidgetService.this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);
            //通过 RemoteViews 给 widget 中的 Button 设置点击事件，点击后发出
pendingIntent 意图
```

```
            views.setOnClickPendingIntent(R.id.btn_clear, pendingIntent);
            //更新 widget
            awm.updateAppWidget(provider, views);
        }
    };
    //开始定时任务，每 10 秒更新一次
    timer.schedule(task, 0, 10000);
}

private class InnerScreenReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
            if (timer != null) {
                timer.cancel();
                timer = null;
            }
            if (task != null) {
                task.cancel();
                task = null;
            }
        } else if (intent.getAction().equals(Intent.ACTION_SCREEN_ON)) {
            if (timer == null && task == null) {
                startUpdateAppWidget();
            }
        }
    }

}

@Override
public void onDestroy() {
    super.onDestroy();
    if (timer != null) {
        timer.cancel();
        timer = null;
    }
    if (task != null) {
        task.cancel();
        task = null;
    }
    unregisterReceiver(receiver);
```

```
        receiver = null;
    }
}
```

Tips：上面代码中我们使用了一个意图

```
Intent intent = new Intent();
intent.setAction("com.itheima.mobileSafe.killProcess");
intent.addCategory("android.intent.category.DEFAULT");
```

该意图被 PendingIntent 使用，当我们的 widget 被点击的时候才真正发送了一个广播。该广播接收者在 AndroidManifest.xml 中进行了注册，注册清单如下：

```
<receiver
android:name="com.itheima.mobileSafe.receiver.AutoKillProcessReceiver" >
            <intent-filter>
                <action android:name="com.itheima.mobileSafe.killProcess" />
                <category android:name="android.intent.category.DEFAULT" >
                </category>
            </intent-filter>
</receiver>
```

AutoKillProcessReceiver 代码清单如下：

```
public class AutoKillProcessReceiver extends BroadcastReceiver {

    private ActivityManager am;
    @Override
    public void onReceive(Context context, Intent intent) {
        am = (ActivityManager) context.getSystemService(context.ACTIVITY_SERVICE);
        List<RunningAppProcessInfo> processes = am.getRunningAppProcesses();
        for(RunningAppProcessInfo info : processes){
            am.killBackgroundProcesses(info.processName);
            System.out.println("杀死了进程: "+info.processName);
        }
    }
}
```

至此，本文档完！

2015 年 2 月 10 日 星期二 14:09:43
北京市海淀区中关村软件园国际软件大厦