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

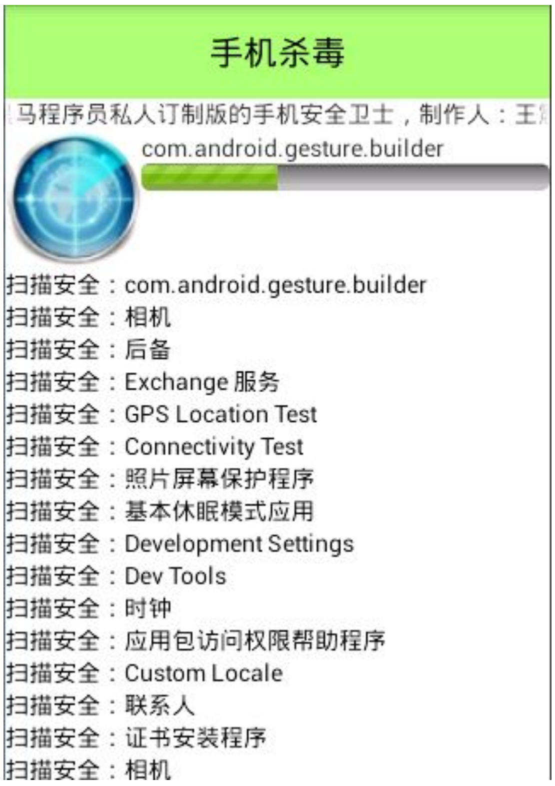
By 阳哥

Android 手机卫士-10 手机杀毒

1. 手机杀毒（★★★★）

1.1 手机杀毒简介

在功能列表中点击 ，进入手机杀毒界面，如下图所示。自动对系统进程扫描。


手机杀毒
马程序员私人订制版的手机安全卫士，制作人：王
com.android.gesture.builder
扫描安全：com.android.gesture.builder
扫描安全：相机
扫描安全：后备
扫描安全：Exchange 服务
扫描安全：GPS Location Test
扫描安全：Connectivity Test
扫描安全：照片屏幕保护程序
扫描安全：基本休眠模式应用
扫描安全：Development Settings
扫描安全：Dev Tools
扫描安全：时钟
扫描安全：应用包访问权限帮助程序
扫描安全：Custom Locale
扫描安全：联系人
扫描安全：证书安装程序
扫描安全：相机

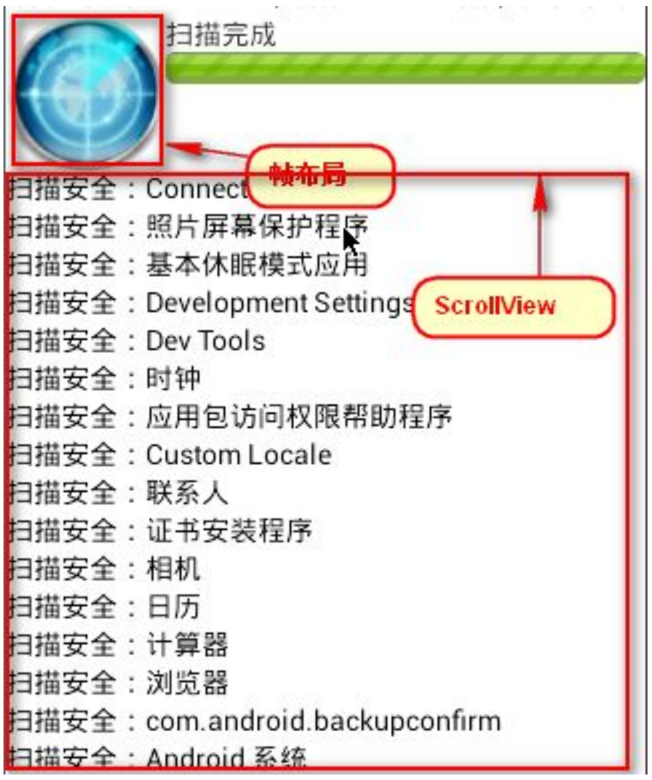
1.2 手机杀毒原理

对于本地查杀，首先我们得有个病毒库，病毒库是事先存在好的，在安装软件的时候安装在相应目录下，如果服务器端病毒库更新了，则我们对应的升级病毒库即可。病毒库存储的是所有病毒软件签名的 md5 码，我们的杀毒软件可以获取手机系统中所有软件签名的 md5 码，将从手机软件中获取的每个 md5 码去数据库中查找，如果发现有，则

代表该软件是病毒软件。

对于联网查杀，我们只需将从手机获取的软件签名 md5 码上传到服务器端，然后由服务器查找病毒库即可。

1.3 手机杀毒布局



扫描完成

扫描安全：Connect
扫描安全：照片屏幕保护程序
扫描安全：基本休眠模式应用
扫描安全：Development Settings
扫描安全：Dev Tools
扫描安全：时钟
扫描安全：应用包访问权限帮助程序
扫描安全：Custom Locale
扫描安全：联系人
扫描安全：证书安装程序
扫描安全：相机
扫描安全：日历
扫描安全：计算器
扫描安全：浏览器
扫描安全：com.android.backupconfirm
扫描安全：Android 系统

布局整体采用 LinearLayout，其中的帧布局是两张重叠的图片，一个是底图（背景图片），

另一个是扇形图，

，在进行扫描的时候不停的让扇形图进行 RotateAnimation 即可看到雷达在扫描的效果。

将扫描后的结果添加到 ScrollView 中即可。进度条的最大值是手机中所有软件个数，然后每扫描一个软件，进度就增加 1 步，这样当全部扫描完后进度条正好走到头。

布局文件名 anti_virus_activity.xml，布局清单如下：

```
<?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="手机杀毒"
        android:textColor="#000000"
        android:textSize="20sp" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:orientation="horizontal" >

        <FrameLayout
            android:layout_width="80dp"
            android:layout_height="80dp" >

            <ImageView
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:src="@drawable/ic_scanner_malware" />

            <ImageView
                android:id="@+id/iv_scanning"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:src="@drawable/act_scanning_03" />
        </FrameLayout>

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

            <TextView
                android:id="@+id/tv_status"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="正在快速扫描" />

            <ProgressBar
                android:id="@+id/progressBar1"
                style="?android:attr/progressBarStyleHorizontal"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:progressDrawable="@drawable/progress_horizontal" />
        </LinearLayout>
    </LinearLayout>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <LinearLayout
            android:id="@+id/ll_container"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >
        </LinearLayout>
    </ScrollView>

</LinearLayout>
```

1.4 手机杀毒业务逻辑的实现

1.4.1 手机杀毒知识点清单

- 1）手机杀毒简单原理
- 2）RotateAnimation 动画的使用
- 3）软件卸载（发现病毒软件的时候卸载软件）
- 3）获取软件签名
- 4）md5 加密

- 5）ScrollView 的使用
- 6）SQLiteDatabase 的使用

1.4.2 手机杀毒代码清单

```
public class AntiVirusActivity extends Activity {
    // 扫描状态
    protected static final int SCANNING = 1;
    protected static final int SCANNING_FINISH = 2;
    private ImageView iv_scanning;
    private TextView tv_status;
    private ProgressBar progressBar1;
    private LinearLayout ll_container;
    // 将扫描结果封装在 ScanInfo 中
    private List<ScanInfo> virusList;
    private Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            TextView view = new TextView(AntiVirusActivity.this);
            // 如果是扫描状态发送来的消息，则将扫描的当前结果添加到 ScrollView 中的
LinearLayout 中
            if (msg.what == SCANNING) {
                ScanInfo scanInfo = (ScanInfo) msg.obj;
                tv_status.setText(scanInfo.getName());
                if (scanInfo.isVirus()) {// 如果是病毒
                    view.setTextColor(Color.BLACK);
                    view.setText("发现病毒： " + scanInfo.getName() + ":" +
scanInfo.getDesc());
                    virusList.add(scanInfo);
                } else {
                    view.setTextColor(Color.BLACK);
                    view.setText("扫描安全： " + scanInfo.getName());
                }
                ll_container.addView(view, 0);
                // 如果扫描完发送来的消息，判断是否有病毒，如果有病毒通过 AlertDialog 提示用
户是否删除
            } else if (msg.what == SCANNING_FINISH) {
                iv_scanning.clearAnimation();
                tv_status.setText("扫描完成");
                if (virusList.size() > 0) {
                    AlertDialog.Builder builder = new Builder(AntiVirusActivity.this);
                builder.setTitle("发现病毒");
                builder.setMessage("发现了(" + virusList.size() + ")个病毒！");
                builder.setNegativeButton("", new OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        // <intent-filter>
                        // <action android:name="android.intent.action.VIEW"
                        // />
                        // <action
                        // android:name="android.intent.action.DELETE" />
                        // <category
                        // android:name="android.intent.category.DEFAULT" />
                        // <data android:scheme="package" />
                        // </intent-filter>
                        for (ScanInfo info : virusList) {
                            // 删除病毒软件，是通过系统自带的 Activity 实现的
                            Intent intent = new Intent();
                            intent.setAction("android.intent.action.DELETE");
                            intent.addCategory("android.intent.category.DEFAULT");
                            intent.setData(Uri.parse("package:" +
info.getPackname()));
                            startActivity(intent);
                        }
                    }
                });
                builder.setPositiveButton("不怕病毒", new OnClickListener() {

                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                });
                builder.show();

            } else {
                Toast.makeText(AntiVirusActivity.this, "您的手机十分安全。",
0).show();
            }

        }
    };
};
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.anti_virus_activity);
        virusList = new ArrayList<ScanInfo>();
        iv_scanning = (ImageView) findViewById(R.id.iv_scanning);
        tv_status = (TextView) findViewById(R.id.tv_status);
        progressBar1 = (ProgressBar) findViewById(R.id.progressBar1);
        /*
         * 定义一个动画
         * 第一个参数 fromDegrees Rotation offset to apply at the start of the
         * animation. 开始角度
         * 第二个参数 toDegrees Rotation offset to apply at the end of the
         * animation.目标角度
         * 第三个参数 pivotXType Specifies how pivotXValue should be
         * interpreted. One of Animation.ABSOLUTE, Animation.RELATIVE_TO_SELF,
         * or Animation.RELATIVE_TO_PARENT.相对于 X 坐标类型
         * 第四个参数 pivotXValue The X coordinate of the
         * point about which the object is being rotated, specified as an
         * absolute number where 0 is the left edge. This value can either be an
         * absolute number if pivotXType is ABSOLUTE, or a percentage (where 1.0
         * is 100%) otherwise.相对于 X 坐标值
         * 第五个参数 pivotYType Specifies how pivotYValue should be
         * interpreted. One of Animation.ABSOLUTE, Animation.RELATIVE_TO_SELF,
         * or Animation.RELATIVE_TO_PARENT.相对于 Y 坐标类型
         * 第六个参数 pivotYValue The Y coordinate of the
         * point about which the object is being rotated, specified as an
         * absolute number where 0 is the top edge.This value can either be an
         * absolute number if pivotYType is ABSOLUTE, or a percentage (where 1.0
         * is 100%) otherwise.相对于 Y 坐标值
         */
        RotateAnimation ra = new RotateAnimation(0, 360, Animation.RELATIVE_TO_SELF,
0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
        //2 秒执行完
        ra.setDuration(2000);
        //一直循环播放
        ra.setRepeatCount(Animation.INFINITE);
        iv_scanning.startAnimation(ra);
        progressBar1.setMax(100);
        ll_container = (LinearLayout) findViewById(R.id.ll_container);
        // 扫描病毒显示是耗时操作，因此在子线程中操作
        new Thread(new Runnable() {
          @Override
          public void run() {
             // 获取 PackageManager 对象
             PackageManager pm = getPackageManager();
             // 通过 pm 获取所有的安装包，包括已经安装的和没有安装的，同时获取安装包的签名
信息
             List<PackageInfo> packages =
pm.getInstalledPackages(PackageManager.GET_UNINSTALLED_PACKAGES +
PackageManager.GET_SIGNATURES);
             // 给 ProgressBar 设置最大值为获取到的安装包的个数
             progressBar1.setMax(packages.size());
             int progress = 0;
             // 遍历安装包
             for (PackageInfo info : packages) {
                // 获取应用程序名称
                String label = info.applicationInfo.loadLabel(pm).toString();
                // 获取应用签名信息
                Signature signature = info.signatures[0];
                // 通过 md5 工具将签名信息进行加密
                String md5String = MD5Utils.encode(signature.toCharsString());
                // 通过查询数据库判断是否是病毒，如果是返回病毒信息
                String virus = AntiVirusQueryDao.isVirus(md5String);
                // 将扫描结果封装在 ScanInfo 中
                ScanInfo scanInfo = new ScanInfo();
                scanInfo.setName(label);
                if (TextUtils.isEmpty(virus)) {// 没有
                   scanInfo.setVirus(false);
                } else {
                   scanInfo.setVirus(true);
                   scanInfo.setDesc(virus);
                   scanInfo.setPackname(info.packageName);
                }
                // 从消息队列中获取消息对象，如果没有则创建
                Message msg = Message.obtain();
                msg.what = SCANNING;
                msg.obj = scanInfo;
                handler.sendMessage(msg);
                progress++;
                progressBar1.setProgress(progress);
                // 因为扫描软件过程太快了，为了方便 演示效果而让系统每扫描一个软件暂停
200 毫秒
                SystemClock.sleep(200);
             }
// 循环结束以后发送一个结束消息
                Message msg = Message.obtain();
                msg.what = SCANNING_FINISH;
                handler.sendMessage(msg);

            }
        }).start();
    }
}
```

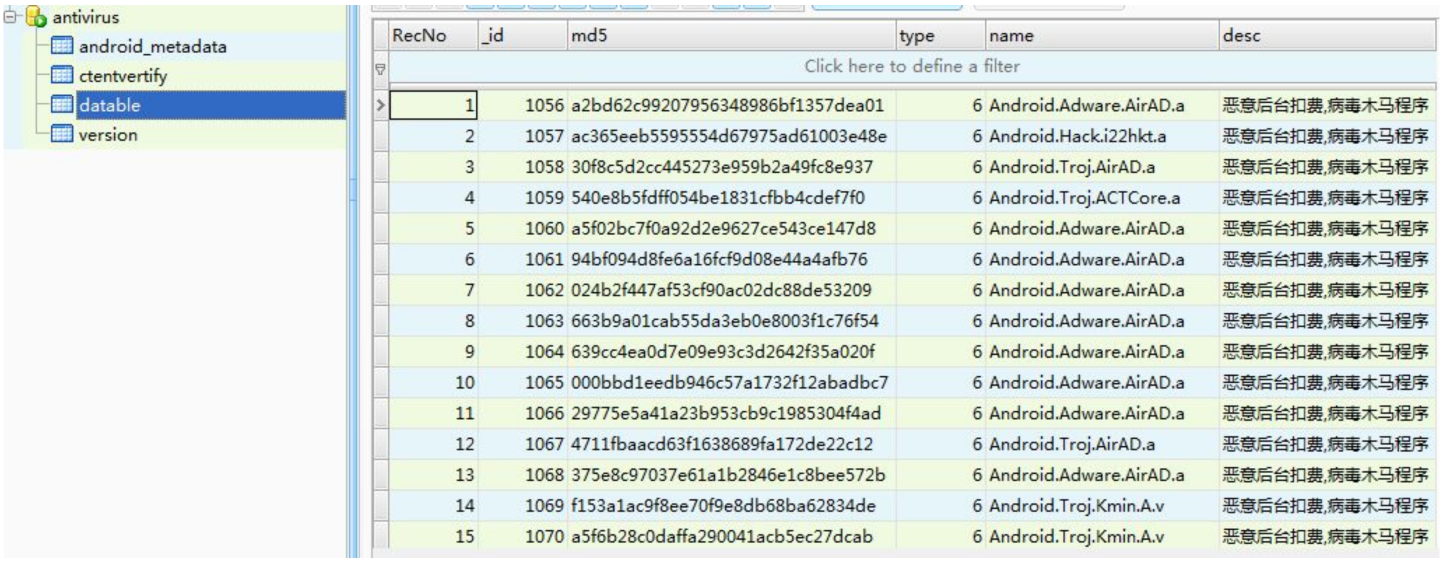
1.4.3 MD5Utils 工具类的实现

```
public class MD5Utils {
    //将字符串进行 md5 加密
    public static String encode(String password) {
        MessageDigest digest;
        try {
            //通过加密工具，并创建一个 md5 算法加密对象
            digest = MessageDigest.getInstance("md5");
            //将密码的字节码加密成字节数组
            byte[] result = digest.digest(password.getBytes());
            StringBuffer buffer = new StringBuffer();
            for (byte b : result) {
                //对每一个字节跟 255 进行和操作，转换为 int 类型
                int number = b & 0xff;
                //将 int 类型转换为 16 进制字符串类型
                String numberStr = Integer.toHexString(number);
                //如果 16 进制的长度只有一位（0~9）只有一位，在前面补 0
                if (numberStr.length() == 1) {
                    buffer.append("0");
                }
                //添加 16 进制字符串
                buffer.append(numberStr);
            }
            return buffer.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return "";
        }
    }}
```

1.4.4 AntiVirusQueryDao 类的实现

```java
private static String path = "/data/data/com.itheima.mobileSafe/files/antivirus.db";
    public static String isVirus(String signatures) {
        SQLiteDatabase database = SQLiteDatabase.openDatabase(path, null,
SQLiteDatabase.OPEN_READONLY);
        String sql = "select desc from datable where md5=?";
        Cursor cursor = database.rawQuery(sql, new String[]{signatures});
        String desc  = null;
        while(cursor.moveToNext()){
            desc = cursor.getString(0);
            break;
        }
        cursor.close();
        return desc;
    }
}
```

Antivirus.db 是一个病毒库，打开其中的 datable 表，观察其中的数据如下图所示。


antivirus
android_metadata
ctentvertify
datable
version

Click here to define a filter

RecNo	_id	md5	type	name	desc
1	1056	a2bd62c99207956348986bf1357dea01	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
2	1057	ac365eeb5595554d67975ad61003e48e	6	Android.Hack.i22hkt.a	恶意后台扣费,病毒木马程序
3	1058	30f8c5d2cc445273e959b2a49fc8e937	6	Android.Troj.AirAD.a	恶意后台扣费,病毒木马程序
4	1059	540e8b5fdff054be1831cfbb4cdef7f0	6	Android.Troj.ACTCore.a	恶意后台扣费,病毒木马程序
5	1060	a5f02bc7f0a92d2e9627ce543ce147d8	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
6	1061	94bf094d8fe6a16fcf9d08e44a4afb76	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
7	1062	024b2f447af53cf90ac02dc88de53209	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
8	1063	663b9a01cab55da3eb0e8003f1c76f54	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
9	1064	639cc4ea0d7e09e93c3d2642f35a020f	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
10	1065	000bbd1eedb946c57a1732f12abadbc7	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
11	1066	29775e5a41a23b953cb9c1985304f4ad	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
12	1067	4711fbaacd63f1638689fa172de22c12	6	Android.Troj.AirAD.a	恶意后台扣费,病毒木马程序
13	1068	375e8c97037e61a1b2846e1c8bee572b	6	Android.Adware.AirAD.a	恶意后台扣费,病毒木马程序
14	1069	f153a1ac9f8ee70f9e8db68ba62834de	6	Android.Troj.Kmin.A.v	恶意后台扣费,病毒木马程序
15	1070	a5f6b28c0daffa290041acb5ec27dcab	6	Android.Troj.Kmin.A.v	恶意后台扣费,病毒木马程序

在开发的时候我们需要将该数据库在 assets 目录中，然后在 SplashActivity 的 onCreate 方法中添加

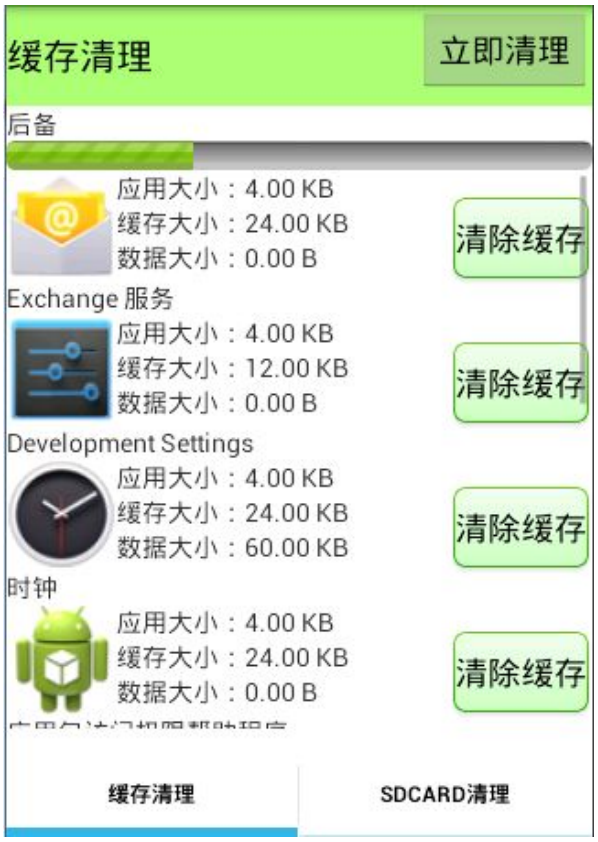
`copyDB("antivirus.db");`将该数据库文件拷贝到

`/data/data/com.itheima.mobileSafe/files/antivirus.db` 目录下。

2. 缓存清理（★★★★）

2.1 缓存清理简介

在功能列表界面点击缓存清理，进入缓存清理主界面，如下图所示。



后备
应用大小：4.00 KB
缓存大小：24.00 KB
数据大小：0.00 B

Exchange 服务
应用大小：4.00 KB
缓存大小：12.00 KB
数据大小：0.00 B

Development Settings
应用大小：4.00 KB
缓存大小：24.00 KB
数据大小：60.00 KB

时钟
应用大小：4.00 KB
缓存大小：24.00 KB
数据大小：0.00 B

点击清除缓存按钮，可以将对应程序的缓存清除掉。

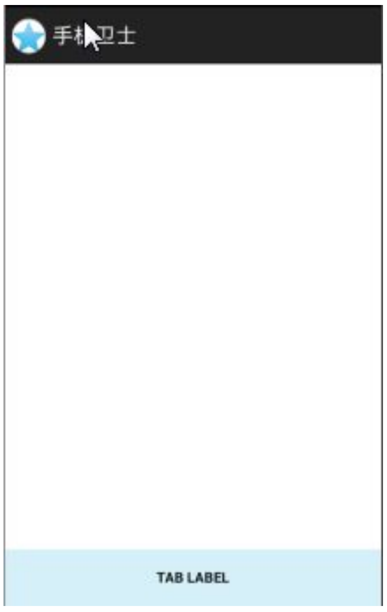
2.2 缓存清理布局

该布局整体是一个 TabHost 布局，TabHost 是一款比较老的布局方式，其用法比较固定。在 TabHost 中整体式 LinearLayout。

clean_activity.xml 是清理缓存的主布局文件，该文件演示了 TabHost 节点的使用方式。布局清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <TabWidget
            android:layout_alignParentBottom="true"
            android:id="@android:id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >
        </TabWidget>
    </RelativeLayout>
    <FrameLayout
        android:id="@android:id/tabcontent"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="60dp" >
    </FrameLayout>
</TabHost>
```

上面布局预览图如下所示：



上面布局只是整体布局的一个大的框架，具体布局文件名为 clean_cache_activity.xml，其布局清单如下：

```
<?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" >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#8866ff00" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="55dp"
            android:gravity="center"
            android:text="缓存清理"
            android:textColor="#000000"
            android:textSize="20sp" />
        <Button
            android:id="@+id/bt_clear"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:onClick="clear"
            android:text="立即清理" />
    </RelativeLayout>

    <TextView
        android:id="@+id/tv_status"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="扫描状态" />
    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:progressDrawable="@drawable/progress_horizontal" />
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <LinearLayout
            android:id="@+id/lv_container"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

2.3 缓存清理业务逻辑的实现

2.3.1 缓存清理知识点清单

- 1）TabHost 的使用
- 2）获取软件缓存
- 3）使用系统清理软件缓存
- 4）aidl 的使用

2.3.2 TabHost 实现代码

清理缓存对应的 Activity 为 CleanActivity.java，其代码清单如下所示：

```
//定义一个 Activity 继承 TabActivity 类
public class CleanActivity extends ~~TabActivity~~ {
   @SuppressWarnings("deprecation")
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.clean_activity);
      //获取 TabHost 对象
      TabHost tabHost = getTabHost();
      //用 tabHost 创建一个 TabSpec,其实就是一个类似选项卡的东西 这里的名字仅仅是作为一个
TabSpec 的唯一标识，不会显示在界面
      TabSpec tabSpec = tabHost.newTabSpec("缓存清理");
      //用 tabHost 创建 sdcard 清理选项卡
      TabSpec tabSpec2 = tabHost.newTabSpec("sdcard 清理");
      //给 tabSpec 设置显示名称
      tabSpec.setIndicator("缓存清理");
      //给 tabSpect 设置一个 Intent 对象，Intent 指向一个 Activity，我们可以理解为 TabHost
```

```
        里面嵌套 Activity 对象
        tabSpec.setContent(new Intent(this, CleanCacheActivity.class));
        //这里仅仅作为演示 TabHost 的用法，因此 sdcard 清理用的 Activity 和缓存清理是一致的
        tabSpec2.setContent(new Intent(this, CleanCacheActivity.class));
        tabSpec2.setIndicator("sdcard 清理");
        //将 tabSpec 添加到 tabHost 中
        tabHost.addTab(tabSpec);
        tabHost.addTab(tabSpec2);
    }
}
```

2.3.3 清理缓存代码清单

清理缓存用到了类为 CleanCacheActivity,其代码清单如下所示：

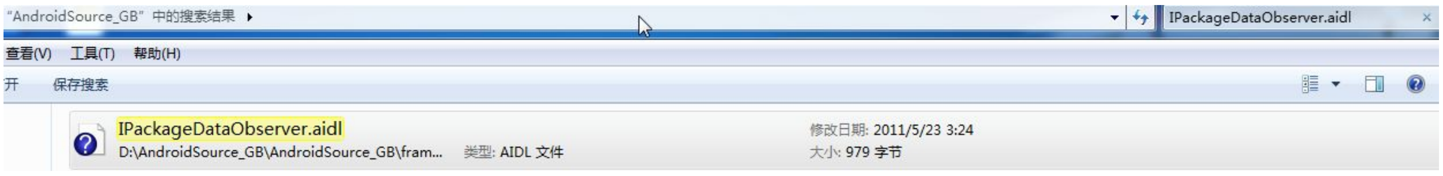
```
public class CleanCacheActivity extends Activity {
    protected static final int SCANNING = 0;
    protected static final int SCANNING_FINISH = 1;
    private static final int FIND_CACHE = 2;
    private TextView tv_status;
    private ProgressBar progressBar1;
    private Button bt_clear;
    private PackageManager pm;
    private List<PackageInfoMy> infos;
    private LinearLayout lv_container;
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
            case SCANNING:
                // 扫描状态
                PackageInfoMy packageInfoMy = (PackageInfoMy) msg.obj;
                tv_status.setText(packageInfoMy.getLabel());
                break;
            case SCANNING_FINISH:
                // 扫描结束
                tv_status.setText("扫描完毕");
                break;
            case FIND_CACHE:
                // 发现缓存
                PackageInfoMy packageInfoMy2 = (PackageInfoMy) msg.obj;
                findCache(packageInfoMy2);
                break;
            default:
                break;
            }
        }

        //发现缓存
        private void findCache(final PackageInfoMy packageInfoMy) {
            //填充一个布局
            View view = View.inflate(CleanCacheActivity.this,
R.layout.clean_cache_list_item, null);
            //初始化布局中的控件
            ImageView iv = (ImageView) view.findViewById(R.id.iv);
            TextView tv_cacheSize = (TextView) view.findViewById(R.id.tv_cacheSize);
            TextView tv_codeSize = (TextView) view.findViewById(R.id.tv_codeSize);
            TextView tv_dataSize = (TextView) view.findViewById(R.id.tv_dataSize);
            TextView tv_name = (TextView) view.findViewById(R.id.tv_name);
            Button bt_clear = (Button) view.findViewById(R.id.bt_clear);
            //给清除缓存按钮添加点击事件
            bt_clear.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {
                    //这里的清除缓存是通过隐式意图打开系统软件设置界面进行清除
                    Intent intent = new Intent();

    intent.setAction("android.settings.APPLICATION_DETAILS_SETTINGS");
                    intent.setData(Uri.parse("package:" +
packageInfoMy.getPackName()));
                    startActivity(intent);
                }
            });
            //给控件设置属性值
            iv.setImageDrawable(packageInfoMy.getIcon());
            tv_cacheSize.setText(packageInfoMy.getCacheSize());
            tv_codeSize.setText(packageInfoMy.getCodeSize());
            tv_dataSize.setText(packageInfoMy.getDataSize());
            //将视图添加到 ListView 中
            lv_container.addView(view, 0);
            tv_name.setText(packageInfoMy.getLabel());
        };
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.clean_cache_activity);
        //初始化控件
        tv_status = (TextView) findViewById(R.id.tv_status);
        progressBar1 = (ProgressBar) findViewById(R.id.progressBar1);
        bt_clear = (Button) findViewById(R.id.bt_clear);
        /*
         * 给立即清理添加点击事件
         * 这里是清除所有应用的缓存，使用 aidl+反射的方式调用 PackageManager 的
freeStorageAndNotify 方法
         */
        bt_clear.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {

                try {
                    //通过反射获取 freeStorageAndNotify Method
                    Method freeStorageAndNotify =
pm.getClass().getMethod("freeStorageAndNotify", new Class[] { long.class,
IPackageDataObserver.class });
                    //遍历应用信息
                    for (PackageInfoMy my : infos) {
                        //获取包名
                        final String packName = my.getPackName();
                        //调用 freeStorageAndNotify 方法
                        freeStorageAndNotify.invoke(pm, Integer.MAX_VALUE, new
IPackageDataObserver.Stub() {
                            //回调函数，当缓存清理成功会被调用
                            @Override
                            public void onRemoveCompleted(String packageName, boolean
succeeded) throws RemoteException {
                                System.out.println(packName + "的缓存已经清除完" +
succeeded);
                            }
                        });

                    }
                } catch (Exception e) {
                    Toast.makeText(CleanCacheActivity.this, e.toString(), 1).show();
                    System.out.println(e);
                    e.printStackTrace();
                }
            }
        });
        pm = getPackageManager();
        infos = new ArrayList<PackageInfoMy>();
        lv_container = (LinearLayout) findViewById(R.id.lv_container);
        //开始扫描应用缓存
        scan();
    }

    private void scan() {
        new Thread(new Runnable() {

            @Override
            public void run() {
                //获取所有的安装包（安装过的和未安装过的，参数中是获取未安装包，是因为在获取
为安装的软件时也会安装已经安装的）
                List<PackageInfo> packages =
pm.getInstalledPackages(PackageManager.GET_UNINSTALLED_PACKAGES);
                progressBar1.setMax(packages.size());
                int progress = 0;
                //遍历包信息
                for (PackageInfo info : packages) {
                    SystemClock.sleep(50);
                    progress++;
                    progressBar1.setProgress(progress);
                    //获取包信息并将这些信息封装到自定义的 PackageInfoMy bean 中
                    final PackageInfoMy packageInfo = new PackageInfoMy();
                    String label = info.applicationInfo.loadLabel(pm).toString();
                    packageInfo.setLabel(label);
                    Drawable icon = info.applicationInfo.loadIcon(pm);
                    packageInfo.setIcon(icon);
                    packageInfo.setPackName(info.packageName);

                    try {
                        //通过反射获取 PackageManager 的 getPackageSizeInfo 方法
                        Method getPackageSizeInfoMethod =
pm.getClass().getMethod("getPackageSizeInfo", new Class[] { String.class,
IPackageStatsObserver.class });
                        getPackageSizeInfoMethod.invoke(pm, packageInfo.getPackName(),
new IPackageStatsObserver.Stub() {
                            //当获取完应用信息后回调该方法
                            @Override
                            public void onGetStatsCompleted(PackageStats pStats, boolean
succeeded) throws RemoteException {
                                long cacheSize = pStats.cacheSize;
                                System.out.println("cacheSize=" + cacheSize);
                                long codeSize = pStats.codeSize;
                                long dataSize = pStats.dataSize;
                                //格式化字节
                                packageInfo.setCacheSize("缓存大小：" +
Formatter.formatFileSize(CleanCacheActivity.this, cacheSize));
                                packageInfo.setCodeSize("应用大小：" +
Formatter.formatFileSize(CleanCacheActivity.this, codeSize));
                                packageInfo.setDataSize("数据大小：" +
Formatter.formatFileSize(CleanCacheActivity.this, dataSize));
                                //如果缓存存在则发送消息
                                if (cacheSize > 0) {
                                    infos.add(packageInfo);
                                    Message msg = Message.obtain();
                                    msg.what = FIND_CACHE;
                                    msg.obj = packageInfo;
                                    handler.sendMessage(msg);
                                    System.out.println("发现缓存：" + packageInfo);
                                }
                            }

                        });
                    } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println(e);
                    }
                    //扫描完成后发送消息
                    Message message = Message.obtain();
                    message.obj = packageInfo;
                    message.what = SCANNING;
                    handler.sendMessage(message);
                }
                Message message = Message.obtain();
                message.what = SCANNING_FINISH;
                handler.sendMessage(message);
            }
        }).start();
    }}
```

Tips：在上面的代码中我们有两个地方使用了 aidl。

1）freeStorageAndNotify

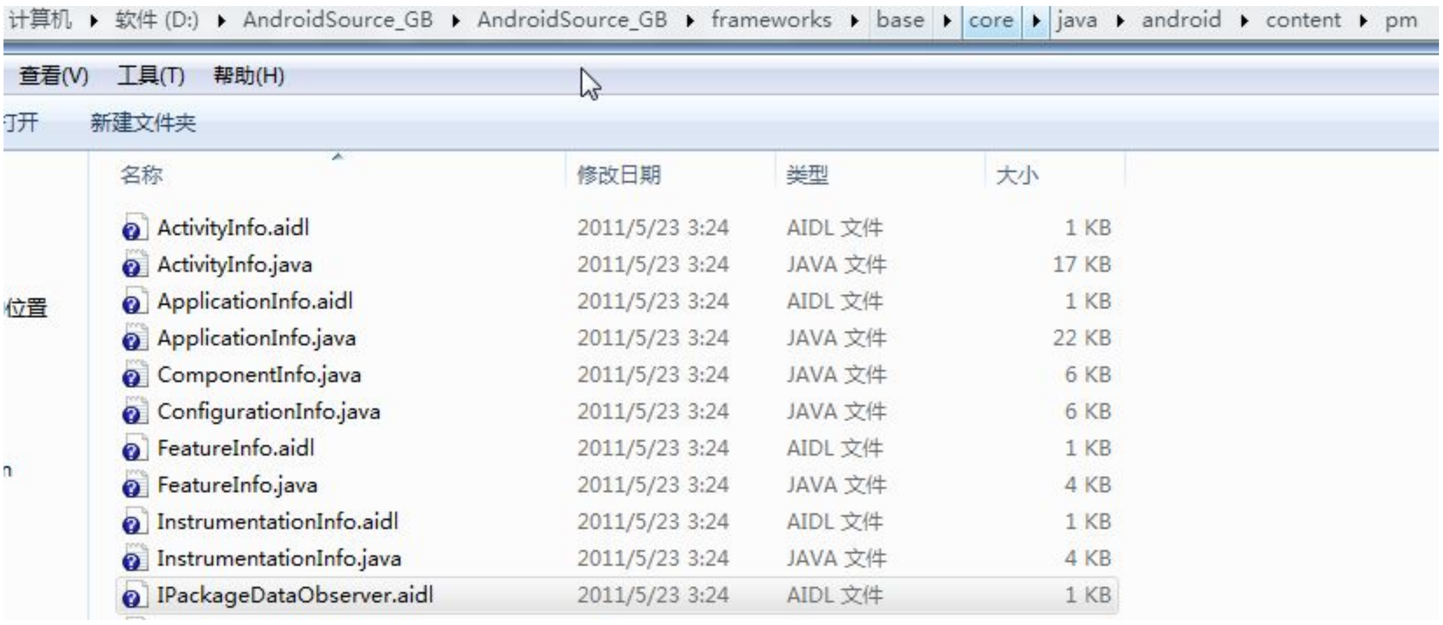
```
 /**
     * Free storage by deleting LRU sorted list of cache files across
     * all applications. If the currently available free storage
     * on the device is greater than or equal to the requested
     * free storage, no cache files are cleared. If the currently
     * available storage on the device is less than the requested
     * free storage, some or all of the cache files across
     * all applications are deleted (based on last accessed time)
     * to increase the free storage space on the device to
     * the requested value. There is no guarantee that clearing all
     * the cache files from all applications will clear up
     * enough storage to achieve the desired value.
     * @param freeStorageSize The number of bytes of storage to be
     * freed by the system. Say if freeStorageSize is XX,
     * and the current free storage is YY,
     * if XX is less than YY, just return. if not free XX-YY number
     * of bytes if possible.
     * @param observer call back used to notify when
     * the operation is completed
     *
     * @hide
     */
    public abstract void freeStorageAndNotify(long freeStorageSize,
IPackageDataObserver observer);
```

有了源码以后我们还需要 aidl 文件，在本地源码中搜索 IPackageDataObserver.aidl，搜索如下图所示：



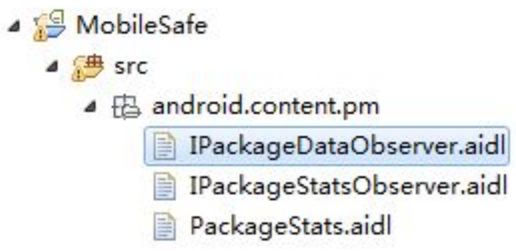
IPackageDataObserver.aidl
D:\AndroidSource_GB\AndroidSource_GB\fram...　类型: AIDL 文件
修改日期: 2011/5/23 3:24
大小: 979 字节

然后右击文件，打开文件位置：


计算机 ▸ 软件 (D:) ▸ AndroidSource_GB ▸ AndroidSource_GB ▸ frameworks ▸ base ▸ core ▸ java ▸ android ▸ content ▸ pm
查看(V) 工具(T) 帮助(H)
打开 新建文件夹
位置

名称	修改日期	类型	大小
ActivityInfo.aidl	2011/5/23 3:24	AIDL 文件	1 KB
ActivityInfo.java	2011/5/23 3:24	JAVA 文件	17 KB
ApplicationInfo.aidl	2011/5/23 3:24	AIDL 文件	1 KB
ApplicationInfo.java	2011/5/23 3:24	JAVA 文件	22 KB
ComponentInfo.java	2011/5/23 3:24	JAVA 文件	6 KB
ConfigurationInfo.java	2011/5/23 3:24	JAVA 文件	6 KB
FeatureInfo.aidl	2011/5/23 3:24	AIDL 文件	1 KB
FeatureInfo.java	2011/5/23 3:24	JAVA 文件	4 KB
InstrumentationInfo.aidl	2011/5/23 3:24	AIDL 文件	1 KB
InstrumentationInfo.java	2011/5/23 3:24	JAVA 文件	4 KB
IPackageDataObserver.aidl	2011/5/23 3:24	AIDL 文件	1 KB

将 IPackageDataObserver.aidl 拷贝到我们工程中，工程目录(android.content.pm 包名必须这样写，也就是必须保证 aidl 的包名跟提供者严格一致)如下所示：



```
MobileSafe
  src
    android.content.pm
      IPackageDataObserver.aidl
      IPackageStatsObserver.aidl
      PackageStats.aidl
```

2）freeStorageAndNotify

```
/**
     * Retrieve the size information for a package.
     * Since this may take a little while, the result will
     * be posted back to the given observer.  The calling context
     * should have the {@link android.Manifest.permission#GET_PACKAGE_SIZE}
permission.
     *
     * @param packageName The name of the package whose size information is to be
retrieved
     * @param userHandle The user whose size information should be retrieved.
     * @param observer An observer callback to get notified when the operation
     * is complete.
     * {@link
android.content.pm.IPackageStatsObserver#onGetStatsCompleted(PackageStats,
boolean)}
     * The observer's callback is invoked with a PackageStats object(containing the
     * code, data and cache sizes of the package) and a boolean value representing
     * the status of the operation. observer may be null to indicate that
     * no callback is desired.
     *
     * @hide
     */
    public abstract void getPackageSizeInfo(String packageName, int userHandle,
            IPackageStatsObserver observer);
/**
     * Like {@link #getPackageSizeInfo(String, int, IPackageStatsObserver)}, but
     * returns the size for the calling user.
     *
     * @hide
     */
    public void getPackageSizeInfo(String packageName, IPackageStatsObserver
observer) {
        getPackageSizeInfo(packageName, UserHandle.myUserId(), observer);
    }
```

同样的步骤我们将 `IPackageStatsObserver.aidl` 也拷贝到工程中。

至此，本文档完！

2015 年 2 月 19 日　星期四　11:15:28

河南省济源市梨林镇