

## 第 3 章 常用工具

第 3 章 常用工具.....	3
3.1 模块概述.....	3
3.1.1 功能介绍.....	3
3.2 程序锁之界面展示.....	4
3.2.1 常用设置之程序锁 UI.....	4
3.2.2 常用设置界面逻辑.....	6
3.2.3 程序锁主界面 UI.....	6
3.2.4 程序锁主逻辑.....	8
3.2.5 Fragment 之未加锁界面 UI.....	9
3.2.6 Fragment 之加锁界面 UI.....	11
3.3 程序锁之数据库操作.....	13
3.3.1 加锁数据库的创建.....	13
3.3.2 加锁数据库 DAO.....	13
3.3.3 获取所有应用工具类.....	15
3.3.4 应用的实体类.....	17
3.4 程序锁主界面逻辑.....	18
3.4.1 Fragment 之未加锁界面数据初始化.....	18
3.4.2 Fragment 之未加锁界面数据适配器.....	19
3.4.3 Fragment 之加锁界面数据初始化.....	21
3.4.4 Fragment 之加锁界面数据适配器.....	22
3.5 看门狗.....	25
3.5.1 看门狗界面 UI.....	25
3.5.2 看门狗服务之输入密码界面.....	26
3.5.3 看门狗服务的实现.....	30
3.6 数据库的使用.....	35

3.6.1 常用工具之查询归属地.....	35
3.6.2 封装跳转界面工具类.....	38
3.6.3 常用工具之号码查询界面.....	39
3.6.4 常用工具之数据库中号码查询逻辑.....	41
3.6.5 s Shape 的使用.....	44
3.6.6 文本动态监听.....	45
3.7 常用工具之归属地显示.....	47
3.7.1 归属地显示界面.....	47
3.7.2 自定义吐司的简单实用.....	53
3.7.3 封装自定义吐司的实现.....	56
3.7.4 封装自定义吐司的美化.....	59
3.7.5 封装自定义吐司的移动.....	60
3.8 本章小结.....	63

## 第 3 章 常用工具

- ◆ 了解 MD5 加密模块功能
- ◆ 掌握手机号码归属地的获取
- ◆ 掌握手机常用工具中程序锁的实现

本章将针对机号码归属地和 MD 加密进行详细地讲解。

### 3.1 模块概述

#### 3.1.1 功能介绍

常用工具模块包含 4 个独立功能，分别为程序锁、看门狗等。接下来针对各个功能进行介绍，具体如下。

##### 1、程序锁

每个人手机上都有一些隐私信息不想被别人看到，例如短信、照片等，为此可以使用程序锁功能将其保护起来。程序锁可以将某一个应用上锁，当进入该程序时需要输入手机防盗密码，具体如图 8-1 所示。

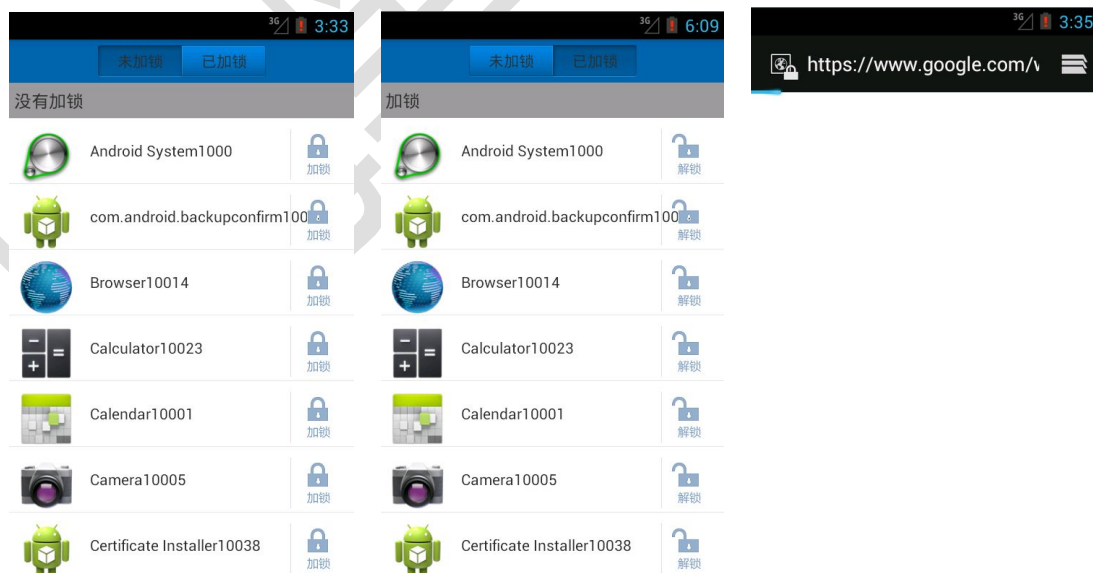


图 3-1 程序锁界面

需要注意的是，在程序锁界面中使用 **Fragment** 控件分别展示未加锁应用和已加锁应用，将一个应用加锁后，此时会将该应用存入到数据库中。当在设置中心打开程序锁服务时，运行在后台的服务监测当前打开的应用，如果打开的应用在数据库中，就说明程序是加锁程序，会弹出输入密码界面，密码输入正确后

才能打开应用（密码为手机防盗密码）。

## 2、看门狗

看门狗服务顾名思义就是替用户监测手机中的一些信息，这里的看门狗主要是为上面的程序锁功能服务的，它可以在用户点击加锁应用时进入一个输入密码的界面，只有输入密码正确才会进入当前应用中，效果图如图 8-2 所示。



图 3-2 看门狗界面

## 3.2 程序锁之界面展示

程序锁功能主要是将加锁的程序信息存入数据库，当程序锁服务打开时，后台会运行一个服务检查当前打开的程序，如果程序在数据库中说明是加锁的，此时会弹出输入密码界面，只有密码输入正确才能进入应用（这个密码就是手机防盗模块中的防盗密码），本节将针对程序锁进行详细讲解。

### 3.2.1 常用设置之程序锁 UI

程序锁的选项与自动更新类似，我们可以直接使用自定义的控件进行设置，布局文件的代码如文件【3-1】所示。

【文件 3-1】 res/layout/activity\_setting.xml

```
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.     <TextView
8.         style="@style/TitleBarTextView"
9.         android:text="设置中心" />
```

```
10.
11.     <!-- android:layout_alignParentRight="true" 在屏幕的右侧 -->
12.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
13.         android:id="@+id/sv_auto_date"
14.         android:layout_width="match_parent"
15.         android:layout_height="wrap_content"
16.         android:layout_marginTop="10dp"
17.         itheima:itbackground="first"
18.         itheima:title="自动更新" />
19.
20.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
21.         android:id="@+id/siv_black_number"
22.         android:layout_width="match_parent"
23.         android:layout_height="wrap_content"
24.         itheima:itbackground="middle"
25.         itheima:title="黑名单" />
26.
27.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
28.         android:id="@+id/siv_app_lock"
29.         android:layout_width="match_parent"
30.         android:layout_height="wrap_content"
31.         itheima:itbackground="middle"
32.         itheima:title="程序锁" />
33.
34.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
35.         android:id="@+id/siv_address"
36.         android:layout_width="match_parent"
37.         android:layout_height="wrap_content"
38.         itheima:itbackground="middle"
39.         itheima:title="归属地显示设置" />
40.
41.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
42.         android:id="@+id/siv_address_style"
43.         android:layout_width="match_parent"
44.         android:layout_height="wrap_content"
45.         itheima:isToggle="true"
46.         itheima:itbackground="last"
47.         itheima:title="归属地设置风格" />
48. </LinearLayout>
```

运行程序，观察常用设置界面的效果展示，如图 3-3 所示。



图 3-3 常用设置之程序锁界面

### 3.2.2 常用设置界面逻辑

布局文件写好之后，下面就需要实现程序锁的点击事件，然后调到显示程序锁内容的界面，在 `SettingActivity` 中需要添加的代码如下所示。

```
1. // 程序锁
2.         siv_app_lock = (SettingItemView) findViewById(R.id.siv_app_lock);
3.         siv_app_lock.setOnClickListener(this);
```

然后，在点击事件 `onClick` 方法中，进行页面的跳转，添加的代码如下所示。

```
4.     case R.id.siv_app_lock:
5.         Intent intent = new Intent(this, AppLockActivity.class);
6.         startActivity(intent);
7.         break;
```

其中，`AppLockActivity` 是用于显示程序锁内容的界面。

### 3.2.3 程序锁主界面 UI

程序锁界面主要包含两个 `Fragment`，分别用于展示未加锁和已加锁程序，当用户点击上面的加锁或者未加锁时，会切换下面帧布局的内容，完整版的效果图如图 3-4 所示。



图 3-4 程序锁主界面

如图 3-4 所示，页面中显示了两个按钮，分别是未加锁和已加锁，点击这两个按钮会切换到不同页面，程序锁界面代码如【文件 3-2】所示。

**【文件 3-2】 activity\_applock.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.     <LinearLayout
7.         android:layout_width="match_parent"
8.         android:layout_height="wrap_content"
9.         android:gravity="center_horizontal"
10.        android:orientation="horizontal"
11.        android:background="#0065ad">
12.        <TextView
13.            android:id="@+id/tv_left"
14.            android:layout_width="wrap_content"
15.            android:layout_height="wrap_content"
16.            android:background="@drawable/tab_left_pressed"
17.            android:gravity="center"
18.            android:text="未加锁" />
19.        <TextView
20.            android:id="@+id/tv_right"
21.            android:layout_width="wrap_content"
22.            android:layout_height="wrap_content"
```

```
23.         android:background="@drawable/tab_right_default"
24.         android:gravity="center"
25.         android:text="已加锁" />
26.     </LinearLayout>
27.     <FrameLayout
28.         android:id="@+id/content"
29.         android:layout_width="match_parent"
30.         android:layout_height="match_parent" >
31.     </FrameLayout>
32. </LinearLayout>
```

在上述布局中，最顶端定义了一个 `LinearLayout` 用于显示两个 `TextView`，分别显示未加锁和已加锁；在页面最下方定义了帧布局用于进行页面的内容填充。当用户点击上面的按钮时，下面帧布局的内容会进行对应的切换。

### 3.2.4 程序锁主逻辑

程序锁的主要功能就是控制两个 `Fragment` 的切换，对于未加锁和已加锁应用的操作都在 `Fragment` 中进行，具体代码如【文件 3-3】所示。

#### 【文件 3-3】AppLockActivity.java

```
1. public class AppLockActivity extends FragmentActivity implements
2.     OnClickListener {
3.     private UnlockFragment unlockFragment;
4.     private LockFragment lockFragment;
5.     private FragmentManager manager;
6.     private TextView tv_left;
7.     private TextView tv_right;
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.         // TODO Auto-generated method stub
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_app_lock);
13.         tv_left = (TextView) findViewById(R.id.tv_left);
14.         tv_right = (TextView) findViewById(R.id.tv_right);
15.         tv_left.setOnClickListener(this);
16.         tv_right.setOnClickListener(this);
17. //         没有加锁
18.         unlockFragment = new UnlockFragment();
19. //         已经加锁
20.         lockFragment = new LockFragment();
21.         // 获取到 fragment 的管理者
22.         manager = getSupportFragmentManager();
23.         // 开启一个事务
24.         FragmentTransaction ft = manager.beginTransaction();
```



```
25.         // 替换当前的页面
26.         // 第一个参数：表示你想替换哪一个页面的 id
27.         // 第二个参数：用哪个 fragment 进行替换
28.         // 提交
29.         ft.replace(R.id.content, unlockFragment).commit();
30.     }
31.     @Override
32.     public void onClick(View v) {
33.         FragmentTransaction ft = manager.beginTransaction();
34.         switch (v.getId()) {
35.             case R.id.tv_left:
36.                 tv_left.setBackgroundResource(R.drawable.tab_left_pressed);
37.                 tv_right.setBackgroundResource(R.drawable.tab_right_default);
38.                 ft.replace(R.id.content, unlockFragment);
39.                 System.out.println("未加锁");
40.                 break;
41.             case R.id.tv_right:
42.                 tv_left.setBackgroundResource(R.drawable.tab_left_default);
43.                 tv_right.setBackgroundResource(R.drawable.tab_right_pressed);
44.                 System.out.println("已加锁");
45.                 ft.replace(R.id.content, lockFragment);
46.                 break;
47.         }
48.         ft.commit();
49.     }
50. }
```

- 第 13~20 行代码用于初始化页面控件，并实例化两个 Fragment。
- 第 21~29 行代码获取 Fragment 管理器，开启事务设置默认显示的 Fragment 界面。
- 第 35~40 行代码用于处理用户点击左边的未加锁按钮的逻辑代码，同时两个按钮的显示背景，并进行对应界面的填充。
- 第 41~46 行代码用于处理用户点击左边的未加锁按钮的逻辑代码，同时两个按钮的显示背景，并进行对应界面的填充。
- 第 48 行代码用于提交事务，这样才能让之前界面的填充替换生效。

### 3.2.5 Fragment 之未加锁界面 UI

观察完整版的效果图，如下图 3-5 所示。

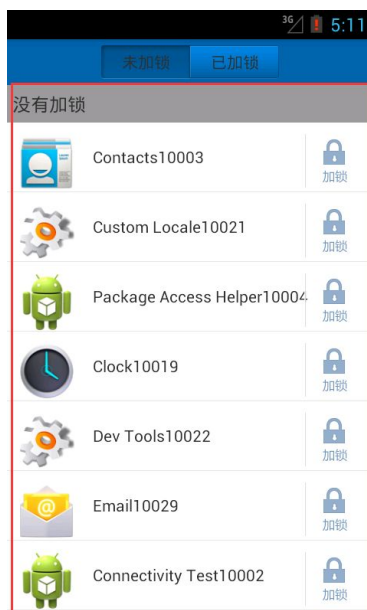


图 3-5 未加锁界面

根据上图 3-5，我们需要定义 Fragment 的布局文件，根据上面的效果图，具体的代码如文件【3-4】所示。

**【文件 3-4】** res/layout/fragment\_left.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.         android:id="@+id/tv_unlock_count"
8.         android:layout_width="match_parent"
9.         android:layout_height="wrap_content"
10.        android:text="没有加锁"
11.        android:textSize="16sp"
12.        android:padding="5dp"
13.        android:background="#66000000"/>
14.     <ListView
15.         android:id="@+id/list_view"
16.         android:layout_width="match_parent"
17.         android:layout_height="match_parent" >
18.     </ListView>
19. </LinearLayout>
```

其中，ListView 是用于展示手机中所有应用程序。条目的布局文件代码如文件【3-5】所示。

**【文件 3-5】** res/layout/item\_app\_unlock.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content"
```

```
5.     android:descendantFocusability="blocksDescendants">
6.     <ImageView
7.         android:id="@+id/icon"
8.         android:layout_width="50dp"
9.         android:layout_height="50dp"
10.        android:layout_margin="10dp"
11.        android:src="@drawable/ic_launcher" />
12.     <TextView
13.         android:id="@+id/tv_app_name"
14.         android:layout_width="match_parent"
15.         android:layout_height="wrap_content"
16.         android:layout_centerVertical="true"
17.         android:layout_marginLeft="5dp"
18.         android:layout_toRightOf="@id/icon"
19.         android:singleLine="true"
20.         android:text="黑马手机卫士"
21.         android:layout_marginTop="10dp"
22.         android:textSize="14sp" />
23.     <ImageButton
24.         android:id="@+id/iv_lock"
25.         android:layout_width="wrap_content"
26.         android:layout_height="wrap_content"
27.         android:layout_alignParentRight="true"
28.         android:layout_margin="10dp"
29.         android:background="@android:color/transparent"
30.         android:src="@drawable/list_button_lock_selected" />
31. </RelativeLayout>
```

其中，list\_button\_lock\_selected.xml 是 ListView 条目上加锁按钮的选择器，具体代码如文件【3-6】所示。

**【文件 3-6】** res/layout/list\_button\_lock\_selected.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <selector xmlns:android="http://schemas.android.com/apk/res/android">
3.     <!-- android:state_focused 获取到焦点 用到遥控器-->
4.     <item android:drawable="@drawable/list_button_lock_pressed" android:state_focused=
5.         "true"></item>
6.     <item android:drawable="@drawable/list_button_lock_pressed" android:state_pressed=
7.         "true"></item>
8.     <item android:drawable="@drawable/list_button_lock_default"></item>
9. </selector>
```

## 3.2.6 Fragment 之加锁界面 UI

观察完整版的加锁界面效果图，如下图 3-6 所示。

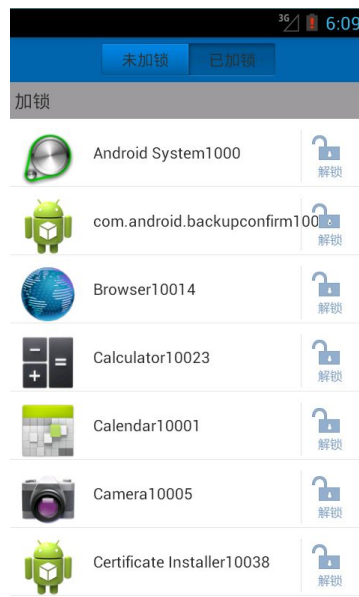


图 3-6 加锁界面

根据上图 3-6，我们需要定义加锁 Fragment 的布局文件 `fragment_right.xml`，根据上面的效果图，具体的代码如文件【3-7】所示。

【文件 3-7】 `res/layout/fragment_right.xml`

```
20. <?xml version="1.0" encoding="utf-8"?>
21. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
22.     android:layout_width="match_parent"
23.     android:layout_height="match_parent"
24.     android:orientation="vertical" >
25.
26.     <TextView
27.         android:id="@+id/tv_lock_count"
28.         android:layout_width="match_parent"
29.         android:layout_height="wrap_content"
30.         android:text="加锁"
31.         android:textSize="16sp"
32.         android:padding="5dp"
33.         android:background="#66000000"/>
34.     <ListView
35.         android:id="@+id/list_view"
36.         android:layout_width="match_parent"
37.         android:layout_height="match_parent" >
38.     </ListView>
39. </LinearLayout>
```

## 3.3 程序锁之数据库操作

### 3.3.1 加锁数据库的创建

在编写程序锁逻辑之前，先创建存储已加锁应用的数据库以及数据库操作类。创建数据库 applock.db 用于存储加锁后的应用信息，具体文件如【3-8】所示。

【文件 3-8】 com.itheima.mobilesafe\_sh2.db/AppLockSQLiteOpenHelper.java

```
1. public class AppLockSQLiteOpenHelper extends SQLiteOpenHelper {
2.     public AppLockSQLiteOpenHelper(Context context) {
3.         //第二个参数：数据库的名字
4.         //第三个参数：游标工厂
5.         //第四个参数：数据库版本号 > 1
6.         super(context, "applock.db", null, 1);
7.         // TODO Auto-generated constructor stub
8.     }
9.     /**
10.      * 创建数据库的表
11.      */
12.     @Override
13.     public void onCreate(SQLiteDatabase db) {
14.         db.execSQL("create table lock (_id integer primary key autoincrement, packagename
15.             varchar(20))");
16.     }
17.     @Override
18.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
19.         // TODO Auto-generated method stub
20.     }
21. }
```

在上述代码中，创建了一个 applock.db 数据库，在该数据库中定义了一个 applock 表，表中有两个属性，id 和 packagename。

### 3.3.2 加锁数据库 DAO

数据库创建完成之后，需要对数据库记性增删改查的操作，接下来就创建该数据库的操作类，具体代码如【文件 3-9】所示。

【文件 3-9】 AppLockDao.java

```
1. public class AppLockDao {
2.     private AppLockSQLiteOpenHelper helper;
3.     private Context mContext;
4.     public AppLockDao(Context context) {
5.         helper = new AppLockSQLiteOpenHelper(context);
```

```
6.         this.mContext = context;
7.     }
8.     /**
9.      * 往程序锁的数据库里面添加数据
10.     * @param packageName
11.     *         程序的包名
12.     * @return
13.     */
14.     public void add(String packageName) {
15.         // 获取到写的数据库
16.         SQLiteDatabase db = helper.getWritableDatabase();
17.         ContentValues values = new ContentValues();
18.         values.put("packagename", packageName);
19.         db.insert("lock", null, values);
20.         mContext.getContentResolver().notifyChange(Uri.parse("content://com.itheima.mobilesaf
21.         e_sh2"), null);
22.     }
23.     /**
24.      * 根据包名进行删除数据
25.     * @param packageName
26.     *         电话号码
27.     * @return
28.     */
29.     public void delete(String packageName) {
30.         SQLiteDatabase db = helper.getWritableDatabase();
31.         int rowNumber = db.delete("lock", "packagename = ?",
32.             new String[] { packageName });
33.         mContext.getContentResolver().notifyChange(Uri.parse("content://com.itheima.mobil
34.         esafe_sh2"), null);
35.     }
36.     /**
37.      * 根据包名进行查找是否在程序锁的数据库里面
38.     * @param number
39.     * @return
40.     */
41.     public String findLock(String packageName) {
42.         String name = "";
43.         SQLiteDatabase db = helper.getReadableDatabase();
44.         Cursor cursor = db.query("lock", null,
45.             "packagename = ?", new String[] { packageName }, null, null, null);
46.         if (cursor.moveToNext()) {
47.             name = cursor.getString(0);
48.         }
49.         return name;
```

```

50.     }
51.     /**
52.      * 查询所有在程序锁数据库里面的数据
53.      * @return
54.      */
55.     public List<String> findAll() {
56.         SQLiteDatabase db = helper.getReadableDatabase();
57.         Cursor cursor = db
58.             .query("lock", new String[] { "packageName" }, null,
59.                 null, null, null, null);
60.         List<String> lists = new ArrayList<String>();
61.         while (cursor.moveToNext()) {
62.             String packageName = cursor.getString(0);
63.             lists.add(packageName);
64.         }
65.         cursor.close();
66.         db.close();
67.         return lists;
68.     }
69.
70. }

```

- 第 8~22 行代码为添加数据，参数接收应用程序的包名，如果数据添加成功则通知内容观察者内容发生变化。
- 第 23~35 行代码为删除数据，同样地如果数据删除成功通知内容观察者。
- 第 63~72 行代码查询数据库中所有的数据，返回 List 集合，集合中包含应用数据的包名。

需要注意的是，在添加和删除方法中都使用了 ContentResolver 暴露数据，而注册 ContentResolver 的 uri 是在第 4 行代码所自定义的 content://com.itheima.mobilesafe\_sh2。然后会在已加锁界面、未加锁界面、程序锁服务中使用该 uri 进行注册内容观察者，如果使用数据库进行增加和删除功能导致数据变化时会通知注册了该 uri 的内容观察者以便做相应地操作。

### 3.3.3 获取所有应用工具类

要实现程序的加锁功能，首先要先获取到手机上的所有应用程序，获取方法在前面几章已经讲解过，这里不再过多解释，具体代码如【文件 3-10】所示。

**【文件 3-10】** AppInfoParser.java

```

1. public class AppInfoParser {
2.     / * 获取到所有的 app 的基本信息
3.     * @return
4.     */
5.     public static List<AppInfo> getAppInfos(Context context) {
6.         // 获取到安装包的管理者
7.         PackageManager packageManager = context.getPackageManager();
8.         // 获取到所有的安装包

```

```
9.          // 参数：获取到所有的标记
10.         List<PackageInfo> installedPackages = packageManager
11.             .getInstalledPackages(0);
12.         List<AppInfo> lists = new ArrayList<AppInfo>();
13.         // 迭代所有的安装包
14.         for (PackageInfo packageInfo : installedPackages) {
15.             AppInfo info = new AppInfo();
16.             // 获取到应用的包名
17.             String appPackageName = packageInfo.packageName;
18.             info.appPackageName = appPackageName;
19.             // 获取到手机的图片
20.             Drawable icon = packageInfo.applicationInfo
21.                 .loadIcon(packageManager);
22.             info.icon = icon;
23.             // 获取到所有的安装包的名字
24.             //packageInfo.applicationInfo.uid 获取到用户 id
25.             String appName = packageInfo.applicationInfo.loadLabel(
26.                 packageManager).toString() + packageInfo.applicationInfo.uid;
27.             info.appName = appName;
28.             // 获取到当前 apk 安装的位置
29.             String sourceDir = packageInfo.applicationInfo.sourceDir;
30.             File file = new File(sourceDir);
31.             // 获取到当前 apk 的大小
32.             long appSize = file.length();
33.             info.appSize = appSize;
34.             // 如果是系统 app /system/app
35.             // 如果是用户 app /data/data/app
36.             if (sourceDir.startsWith("/system")) {
37.                 // 安装的系统的 app
38.                 info.isUserApp = false;
39.                 System.out.println("系统 app");
40.             } else {
41.                 // 用户 app
42.                 info.isUserApp = true;
43.                 System.out.println("用户 app");
44.             }
45.             // 获取到当前 app 的标记
46.             int flags = packageInfo.applicationInfo.flags;
47.             // 判断当前是否是系统应用
48.             //ApplicationInfo.FLAG_SYSTEM 表示系统标记 (当前是系统应用程序)
49.             if ((flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
50.                 // 系统应用
51.                 System.out.println("系统 app");
52.             } else {
```



```
53.         // 用户应用
54.         System.out.println("用户 app");
55.     }
56.     //判断当前的 apk 是否是在外部存储(sd 卡)
57.     if((flags & ApplicationInfo.FLAG_EXTERNAL_STORAGE) != 0){
58.         //sd 卡
59.         info.isRom = false;
60.     }else{
61.         //机身内存
62.         info.isRom = true;
63.     }
64.     lists.add(info);
65. }
66. return lists;
67. }
68. }
```

### 3.3.4 应用的实体类

为了方便对应用进行操作，需要定义一个应用程序的实体类，该类中包含包名、图标、应用名、应用路径、是否加锁，具体代码如【文件 3-11】所示。

**【文件 3-11】** AppInfo.java

```
1. public class AppInfo {
2.     // Drawable 不仅仅表示图片 表示资源文件 表示的范围更广泛
3.     // bitmap 只能单纯表示图片
4.     /**
5.      * 表示应用的图标
6.      */
7.     public Drawable icon;
8.     /**
9.      * 应用的名字
10.     */
11.     public String appName;
12.     /**
13.      * 存放 app 的位置 手机内存 true 或者是 sd 卡 false
14.      */
15.     public boolean isRom;
16.     /**
17.      * app 的大小
18.      */
19.     public long appSize;
20.     /**
21.      * 是否是用户程序 true 表示用户程序 false 表示系统程序
```

```
22.     */
23.     public boolean isUserApp;
24.     /**
25.      * 应用程序的包名
26.      */
27.     public String appPackageName;
28. }
```

## 3.4 程序锁主界面逻辑

### 3.4.1 Fragment 之未加锁界面数据初始化

当进入程序锁模块时默认显示的是未解锁界面，当第一次打开该模块时会将手机中安装的所有应用信息都显示在界面中，当点击某一个应用时会将该应用从未加锁界面删除并添加到已加锁界面中，并且被删除的条目会有一个滑动删除的动画。

根据上图 3-5 的效果图和前面定义好的布局文件，我们需要在 `ListView` 适配器上显示所有的应用信息，这个我们需要使用之前定义好的工具类 `AppInfoParser` 来获取。具体的代码如文件【3-12】所示。

**【文件 3-12】** com.itheima.mobilesafe\_sh2.fragment/UnLockFragment.java

```
1. public class UnLockFragment extends Fragment {
2.     private ListView mListView;
3.     private TextView tv_unlock_count;
4.     private AppLockDao dao;
5.     private List<AppInfo> unLockInfos;
6.     private UnLockAdapter adapter;
7.     @Override
8.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
9.         Bundle savedInstanceState) {
10.         View contentView = inflater.inflate(R.layout.fragment_left, null);
11.         mListView = (ListView) contentView.findViewById(R.id.list_view);
12.         // 没有加锁的总数
13.         tv_unlock_count = (TextView) contentView
14.             .findViewById(R.id.tv_unlock_count);
15.         return contentView;
16.     }
17.     @Override
18.     public void onStart() {
19.         // TODO Auto-generated method stub
20.         super.onStart();
21.         // 获取到所有安装到手机上面的 app
22.         List<AppInfo> mInfos = AppInfoParser.getAppInfos(getActivity());
23.         // 程序锁数据库
24.         dao = new AppLockDao(getActivity());
```

```
25.         // 初始化没有加锁的集合
26.         unlockInfos = new ArrayList<AppInfo>();
27.         for (AppInfo appInfo : mInfos) {
28.             // 如果查询出来是空。说明没有在程序锁的数据库里面
29.             if (TextUtils.isEmpty(dao.findLock(appInfo.appPackageName))) {
30.                 //判断如果是自己的应用程序。那么就不往数据库里面进行添加
31.                 if(appInfo.appPackageName.equals(getActivity().getPackageName())){
32.                     continue ;
33.                 }
34.                 // 没有加锁
35.                 unlockInfos.add(appInfo);
36.             }
37.         }
38.         adapter = new UnLockAdapter();
39.         mListView.setAdapter(adapter);
40.     }
41. }
```

### 3.4.2 Fragment 之未加锁界面数据适配器

未加锁界面和已加锁界面使用的数据适配器都是 `AppLockAdapter.java`，它接收一个 `List` 集合，不同界面调用 `Adapter` 时，传输的数据集合不同，例如未加锁界面调用时传递过来的集合数据是未加锁应用信息，已加锁界面调用时传递过来的集合数据是已加锁应用信息。关于适配器的代码如下所示。

```
1.     private class UnLockAdapter extends BaseAdapter {
2.         @Override
3.         public int getCount() {
4.             // TODO Auto-generated method stub
5.             return unlockInfos.size();
6.         }
7.         @Override
8.         public Object getItem(int position) {
9.             // TODO Auto-generated method stub
10.            return unlockInfos.get(position);
11.        }
12.        @Override
13.        public long getItemId(int position) {
14.            // TODO Auto-generated method stub
15.            return position;
16.        }
17.        @Override
18.        public View getView(final int position, View convertView,
19.                             ViewGroup parent) {
20.            final ViewHolder holder;
```

```
21.         final View view;
22.         System.out.println("position-----" + position);
23.         if (convertView == null) {
24.             view = View
25.                 .inflate(getActivity(), R.layout.item_app_lock, null);
26.             holder = new ViewHolder();
27.             holder.iv_icon = (ImageView) view.findViewById(R.id.icon);
28.             holder.tv_app_name = (TextView) view
29.                 .findViewById(R.id.tv_app_name);
30.             holder.iv_lock = (ImageButton) view.findViewById(R.id.iv_lock);
31.             view.setTag(holder);
32.         } else {
33.             view = convertView;
34.             holder = (ViewHolder) view.getTag();
35.         }
36.         final AppInfo appInfo = unlockInfos.get(position);
37.         // setImage 表示 src
38.         // setBackgroundDrawable 设置背景
39.         // setBackgroundDrawable 需要注意。设置这个过时了的方法
40.         holder.iv_icon.setImageDrawable(appInfo.icon);
41.         holder.tv_app_name.setText(appInfo.appName);
42.         // 把当前的对象添加到数据库里面
43.         holder.iv_lock.setOnClickListener(new OnClickListener() {
44.             @Override
45.             public void onClick(View v) {
46.                 // 当前控件不可用
47.                 holder.iv_lock.setEnabled(false);
48.                 // 第一个参数：相对自己
49.                 TranslateAnimation animation = new TranslateAnimation(
50.                     Animation.RELATIVE_TO_SELF, 0,
51.                     Animation.RELATIVE_TO_SELF, 1.0f,
52.                     Animation.RELATIVE_TO_SELF, 0,
53.                     Animation.RELATIVE_TO_SELF, 0);
54.                 animation.setDuration(300);
55.                 view.startAnimation(animation);
56.                 new Thread() {
57.                     public void run() {
58.                         SystemClock.sleep(300);
59.                         getActivity().runOnUiThread(new Runnable() {
60.                             @Override
61.                             public void run() {
62.                                 // TODO Auto-generated method stub
63.                                 // 把包名添加到程序锁的数据库里面
64.                                 dao.add(appInfo.appPackageName);
```

```
65.                                     // 从未枷锁的集合里面移除当前对象
66.                                     unlockInfos.remove(position);
67.                                     // 刷新界面
68.                                     adapter.notifyDataSetChanged();
69.                                     //当前控件不可用
70.                                     //当动画做完之后重新设置回来
71.                                     holder.iv_lock.setEnabled(true);
72.                                     }
73.                                     });
74.                                     };
75.                                     }.start();
76.                                     }
77.                                     });
78.                                     return view;
79.                                     }
80.                                     }
81.                                     static class ViewHolder {
82.                                         ImageView iv_icon;
83.                                         TextView tv_app_name;
84.                                         ImageButton iv_lock;
85.                                     }
```

运行程序，点击加锁几个应用程序，观察效果图如图 3-7 所示，发现加锁的应用未加锁界面消失了。

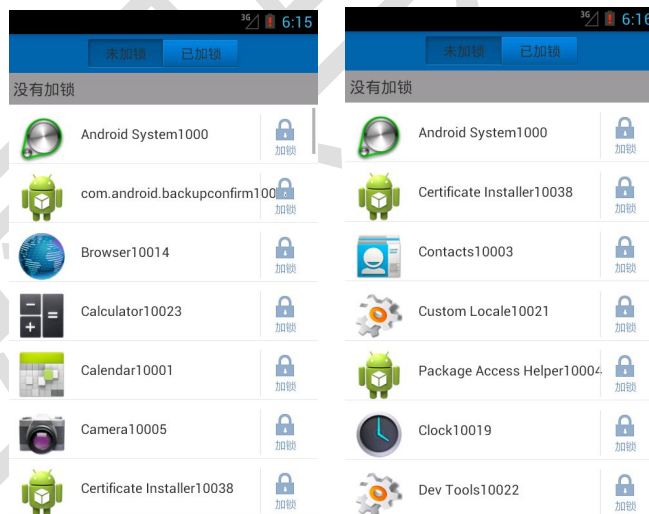


图 3-7 未加锁界面

### 3.4.3 Fragment 之加锁界面数据初始化

与未加锁界面相同，只不过需要判断当前应用是否在加锁数据库中，如果在就放到数据集合中，加锁界面的初始化操作如下所示。

```
1. public class LockFragment extends Fragment {
2.     private ListView mListView;
```

```
3.     private TextView tv_lock_count;
4.     private AppLockDao dao;
5.     private List<AppInfo> lockLists;
6.     private LockAdapter adapter;
7.     @Override
8.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
9.         Bundle savedInstanceState) {
10.         View view = View.inflate(getActivity(), R.layout.fragment_right, null);
11.         mListview = (ListView) view.findViewById(R.id.list_view);
12.         tv_lock_count = (TextView) view.findViewById(R.id.tv_lock_count);
13.         return view;
14.     }
15.     @Override
16.     public void onStart() {
17.         // TODO Auto-generated method stub
18.         super.onStart();
19.         dao = new AppLockDao(getActivity());
20.         List<AppInfo> appInfos = AppInfoParser.getAppInfos(getActivity());
21.         lockLists = new ArrayList<AppInfo>();
22.         for (AppInfo appInfo : appInfos) {
23.             if (TextUtils.isEmpty(dao.findLock(appInfo.appPackageName))) {
24.                 // 没有加锁
25.             } else {
26.                 // 已经加锁
27.                 lockLists.add(appInfo);
28.             }
29.         }
30.         adapter = new LockAdapter();
31.         mListview.setAdapter(adapter);
32.     }
```

### 3.4.4 Fragment 之加锁界面数据适配器

数据我们已经获取，下面就是进行数据适配器中的填充，关于适配器的代码如下所示。

```
1. private class LockAdapter extends BaseAdapter {
2.     @Override
3.     public int getCount() {
4.         // TODO Auto-generated method stub
5.         return lockLists.size();
6.     }
7.     @Override
8.     public Object getItem(int position) {
9.         // TODO Auto-generated method stub
```

```
10.         return lockLists.get(position);
11.     }
12.     @Override
13.     public long getItemId(int position) {
14.         // TODO Auto-generated method stub
15.         return position;
16.     }
17.
18.     @Override
19.     public View getView(final int position, final View convertView,
20.         ViewGroup parent) {
21.         final ViewHolder holder;
22.         final View view;
23.         if (convertView == null) {
24.             view = View.inflate(getActivity(), R.layout.item_app_unlock,
25.                 null);
26.             holder = new ViewHolder();
27.             holder.icon = (ImageView) view.findViewById(R.id.icon);
28.             holder.tv_app_name = (TextView) view
29.                 .findViewById(R.id.tv_app_name);
30.             holder.iv_lock = (ImageButton) view.findViewById(R.id.iv_lock);
31.             view.setTag(holder);
32.         } else {
33.             view = convertView;
34.             holder = (ViewHolder) convertView.getTag();
35.         }
36.         final AppInfo appInfo = lockLists.get(position);
37.         holder.icon.setImageDrawable(appInfo.icon);
38.         holder.tv_app_name.setText(appInfo.appName);
39.
40.         holder.iv_lock.setOnClickListener(new OnClickListener() {
41.             @Override
42.             public void onClick(View v) {
43.                 holder.iv_lock.setEnabled(false);
44.                 // 第一个参数：相对自己
45.                 TranslateAnimation animation = new TranslateAnimation(
46.                     Animation.RELATIVE_TO_SELF, 0f,
47.                     Animation.RELATIVE_TO_SELF, -1.0f,
48.                     Animation.RELATIVE_TO_SELF, 0,
49.                     Animation.RELATIVE_TO_SELF, 0);
50.                 animation.setDuration(300);
51.                 view.startAnimation(animation);
52.                 new Thread() {
```

```

53.         public void run() {
54.             SystemClock.sleep(300);
55.             getActivity().runOnUiThread(new Runnable() {
56.                 @Override
57.                 public void run() {
58.                     // TODO Auto-generated method stub
59.                     // 从程序锁的数据库里面删除数据
60.                     dao.delete(appInfo.appPackageName);
61.                     lockLists.remove(position);
62.                     adapter.notifyDataSetChanged();
63.                     holder.iv_lock.setEnabled(true);
64.                 }
65.             });
66.         };
67.         }.start();
68.     }
69. });
70.     return view;
71. }
72. }
73.
74. static class ViewHolder {
75.     ImageView icon;
76.     TextView tv_app_name;
77.     ImageButton iv_lock;
78. }

```

- 第 27~33 行的 `onCreateView()` 方法，用于初始化未加锁布局以及控件，Fragment 打开时先执行该方法。
- 第 36~37 行代码首先得到数据库对象和手机中安装的所有应用列表。
- 第 41~47 行代码，在创建数据库的添加和删除方法时做了一个操作，通知内容观察者某个 uri 的数据发生了改变，在这里我们使用该 uri 注册一个内容观察者，如果数据库内容发生了改变时，会执行 `onChange()` 方法，我们在该方法中则重新执行 `fillData()` 方法刷新界面。
- 第 49~66 行的 `fillData()` 方法中遍历应用是否在数据库中，如果不在则添加到 `ArrayList` 集合中，设置标志为 `false`，代表该应用未加锁，并发送一条 `Message`。
- 第 67~102 行代码为 `ListView` 条目点击监听，首先判断点击的条目信息是否是本应用，如果是则直接返回不再执行。否则执行一个动画效果，点击其他条目时，条目会从右向左滑动消失，并将该应用添加到数据库中（表示该应用已加锁）重新刷新界面。

要注意的是，适配器中 `getView` 方法中 `view` 对象必须是局部变量，如果变为适配器的成员变量，当用户点击解锁或者加锁时，实际上是点击的最下面的那个条目对象，这是由于缓存的存在造成的。但是，如果使用局部变量，用户就能准确点击到当前的 `view` 对象，并进行相应的动画效果。

运行程序，效果图如图 3-8 所示，说明我们加锁应用成功。



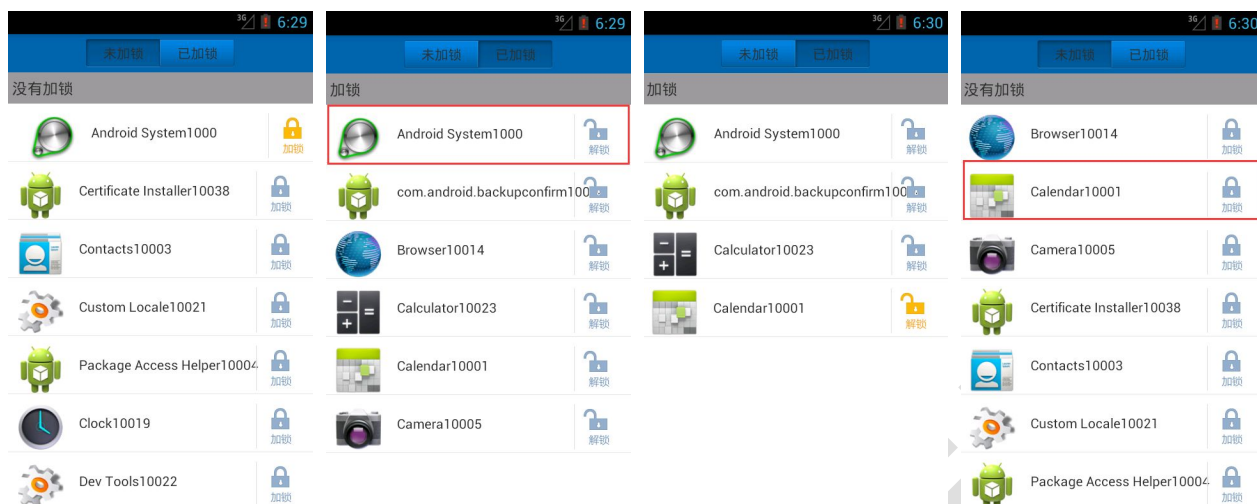


图 3-8 加锁界面

## 3.5 看门狗

看门狗，顾名思义就是一个帮助用户时刻关注某些应用的工具，这里我们使用服务来实现看门狗的功能，即帮用户检测那些被加入程序锁的应用，如果被点击的话将会弹出一个输入密码的界面，只有输入正确密码才可以进入当前应用中。

### 3.5.1 看门狗界面 UI

在设置中心的布局文件中添加以下代码，如下所示。

```
1. <com.itheima.mobilesafe_sh2.act.view.SettingItemView
2.     android:id="@+id/siv_app_dog"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content"
5.     itheima:itbackground="middle"
6.     itheima:itTitle="看门狗" />
```

设置中心界面中需要一些初始化，代码在 onCreate() 中，如下所示。

```
1. //看门狗
2.     siv_app_dog = (SettingItemView) findViewById(R.id.siv_app_dog);
3.     siv_app_dog.setOnClickListener(this);
```

点击事件的处理在 onclick() 方法中，如下所示。

```
1. case R.id.siv_app_dog:
2.     Intent dogIntent = new Intent(this,
3.         WatchDogService.class);
4.     // 设置滑动开关。如果是开着的点击关闭。如果是关闭的。点击打开
5.     if (ServiceStateUtils.serviceRunning(SettingActivity.this,
6.         WatchDogService.class)) {
7.         siv_app_dog.setToggle(false);
8.         stopService(dogIntent);
```

```
9.         } else {
10.             siv_app_dog.setToggle(true);
11.             startService(dogIntent);
12.         }
13.         break;
```

回显操作在 `onstart()`方法中，具体如下所示。

```
1. // 判断看门狗是否已经开启
2.     if (ServiceStateUtils.serviceRunning(SettingActivity.this,
3.         WatchDogService.class)) {
4.         siv_app_dog.setToggle(true);
5.     } else {
6.         siv_app_dog.setToggle(false);
7.     }
```

运行程序，效果图如图 3-9 所示。



图 3-9 设置中心之看门狗界面

### 3.5.2 看门狗服务之输入密码界面

看门狗服务 `WatchDogService` 主要用于在手机后台监测手机信息的，在这里是用于监视加锁后的应用，如果被打开则先进入输入密码界面，其中输入密码界面 `EnterPwdActivity.java` 的界面效果如图 3-10 所示。



图 3-10 输入密码界面

布局文件的代码如文件【3-13】所示。

**【文件 3-13】** res/layout/activity\_enter\_pwd.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="输入隐私保护密码" />
9.     <ImageView
10.        android:id="@+id/iv_icon"
11.        android:layout_width="100dp"
12.        android:layout_height="100dp"
13.        android:layout_gravity="center"
14.        android:src="@drawable/ic_launcher" />
15.
16.     <RelativeLayout
17.        android:layout_width="match_parent"
18.        android:layout_height="wrap_content" >
19.         <EditText
20.            android:id="@+id/et_pwd"
21.            android:layout_width="match_parent"
22.            android:layout_height="wrap_content"
23.            android:background="@drawable/et_input_normal"
24.            android:inputType="phone"/>
25.         <Button
26.            android:onClick="buttonOk"
27.            android:layout_width="wrap_content"
28.            android:layout_height="wrap_content"
29.            android:layout_alignParentRight="true"
30.            android:layout_centerVertical="true"
31.            android:background="@drawable/dg_btn_confirm_selected"
32.            android:text="确定" />
33.     </RelativeLayout>
34. </LinearLayout>
```

其中，dg\_btn\_confirm\_selected.xml 是确定按钮的选择器，具体代码如文件【3-14】所示。

**【文件 3-14】** res/layout/activity\_enter\_pwd.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_btn_confirm_select" android:state_pressed="true">
4.         </item>
5.     <item android:drawable="@drawable/dg_btn_confirm_normal"></item>
```

6. </selector>

输入密码的输入框背景 et\_input\_normal.xml 也是使用 shape 来定义的，具体代码如文件【3-15】所示。

**【文件 3-15】** res/drawable/activity\_enter\_pwd.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <shape xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:shape="rectangle" >
4.      <!--
5.          rectangle 矩形
6.          oval 圆角椭圆
7.          line 线
8.          ring 环行
9.      -->
10.
11.     <!--
12.         gradient 梯度渐变色
13.         设置开始中间和结束的颜色
14.     -->
15.     <!-- <gradient -->
16.     <!-- android:centerColor="#0000ff" -->
17.     <!-- android:endColor="#ff00ff00" -->
18.     <!-- android:startColor="#ff000000" /> -->
19.     <!-- 圆角边角 -->
20.     <corners android:radius="5dp" />
21.     <!-- 固定色 不能和渐变色一起用。不然会有冲突 -->
22.     <solid android:color="#ffffd6" />
23.     <!-- 内容距离控件的位置 -->
24.
25.     <padding
26.         android:bottom="20dp"
27.         android:left="20dp"
28.         android:right="20dp"
29.         android:top="20dp" />
30.
31.     <!--
32.         描边 android:width="1dp"
33.         android:color="#000"实体边框
34.         dashWidth 虚线边框
35.     -->
36.     <stroke
37.         android:width="1dp"
38.         android:color="#0000ff" />
39.     <size android:height="48dp" />
40. </shape>
```

布局文件写好之后，我们需要实现输入密码界面的一些逻辑，具体如文件【3-16】所示。

**【文件 3-16】** com.itheima.mobilesafe\_sh2.act/EnterPwdActivity.java

```
1.  public class EnterPwdActivity extends Activity {
2.      private EditText et_pwd;
3.      private String packageName;
4.      @Override
5.      protected void onCreate(Bundle savedInstanceState) {
6.          // TODO Auto-generated method stub
7.          super.onCreate(savedInstanceState);
8.          setContentView(R.layout.activity_enter_pwd);
9.          et_pwd = (EditText) findViewById(R.id.et_pwd);
10.         ImageView iv_icon = (ImageView) findViewById(R.id.iv_icon);
11.         Intent intent = getIntent();
12.         if (intent != null) {
13.             packageName = intent.getStringExtra("packageName");
14.         }
15.         PackageManager packageManager = getPackageManager();
16.         try {
17.             PackageInfo packageInfo = packageManager.getPackageInfo(packageName, 0);
18.             // 获取到被锁程序的图标
19.             Drawable icon = packageInfo.applicationInfo.loadIcon(packageManager);
20.             iv_icon.setImageDrawable(icon);
21.         } catch (Exception e) {
22.             e.printStackTrace();
23.         }
24.     }
25.     // <intent-filter>
26.     // <action android:name="android.intent.action.MAIN" />
27.     // <category android:name="android.intent.category.HOME" />
28.     // <category android:name="android.intent.category.DEFAULT" />
29.     // <category android:name="android.intent.category.MONKEY"/>
30.     // </intent-filter>
31.     /** 监听后退键，回到桌面
32.     */
33.     @Override
34.     public void onBackPressed() {
35.         Intent intent = new Intent();
36.         intent.setAction("android.intent.action.MAIN");
37.         intent.addCategory("android.intent.category.HOME");
38.         intent.addCategory("android.intent.category.DEFAULT");
39.         intent.addCategory("android.intent.category.MONKEY");
40.         startActivity(intent);
41.     }
42.     /**确定
```

```
43.      * @param view
44.      */
45.      public void buttonOk(View view) {
46.          // 获取到用户输入的密码
47.          String str_pwd = et_pwd.getText().toString().trim();
48.          // 判断当前用户输入的密码是否正确
49.          // 123
50.          if (TextUtils.isEmpty(str_pwd)) {
51.              ToastUtils.showSafeToast(this, "请输入密码");
52.          } else {
53.              // 如果当前的密码等于 123 表示密码正确
54.              if (str_pwd.equals("123")) {
55.                  // 让狗停下来
56.                  Intent intent = new Intent();
57.                  // 设置一个动作
58.                  intent.setAction("www.heima.com");
59.                  // 传递包名
60.                  intent.putExtra("packageName", packageName);
61.                  // 发送一个让狗停下来保护的广播
62.                  sendBroadcast(intent);
63.                  // 停止保护
64.                  finish();
65.              } else {
66.                  ToastUtils.showSafeToast(this, "密码错误");
67.              }
68.          }
69.      }
70. }
```

### 3.5.3 看门狗服务的实现

下面我们就要实现看门狗服务 WatchDogService 的逻辑，具体的代码如文件【3-17】所示。

**【文件 3-17】** com.itheima.mobilesafe\_sh2.service/WatchDogService.java

```
1. public class WatchDogService extends Service {
2.     // 看门狗的标记
3.     private boolean flag;
4.     private ActivityManager am;
5.     private AppLockDao dao;
6.     private Intent intent;
7.     private List<String> mLockLists;
8.     @Override
9.     public IBinder onBind(Intent intent) {
10.         return null;
11.     }
```

```
12.     private String stopProtectingPackageName;
13.     /**
14.      * 让狗停下来的广播
15.      */
16.     private class WatchDogReceiver extends BroadcastReceiver {
17.         @Override
18.         public void onReceive(Context context, Intent intent) {
19.             // 判断当前是否需要停止保护
20.             // 说明用户输入的密码正确
21.             if ("www.heima.com".equals(intent.getAction())) {
22.                 // 获取到停止保护的包名
23.                 stopProtectingPackageName = intent.getStringExtra("packageName");
24.                 // 获取到当前的包名。那么就需要停止保护
25.             } else if (Intent.ACTION_SCREEN_OFF.equals(intent.getAction())) {
26.                 // 让狗停下来
27.                 flag = false;
28.                 stopProtectingPackageName = null;
29.             } else if (Intent.ACTION_SCREEN_ON.equals(intent.getAction())) {
30.                 // 如果解锁之后。那么让狗继续看门
31.                 if (flag == false) {
32.                     startWatchDog();
33.                 }
34.             }
35.         }
36.     }
37.     private class MyContentObserver extends ContentObserver{
38.         public MyContentObserver(Handler handler) {
39.             super(handler);
40.         }
41.         /**当数据库发生改变的时候调用当前的方法
42.         */
43.         @Override
44.         public void onChange(boolean selfChange) {
45.             // TODO Auto-generated method stub
46.             super.onChange(selfChange);
47.             //如果数据库发生改变重新查询数据库
48.             mLockLists = dao.findAll();
49.         }
50.     }
51.     @Override
52.     public void onCreate() {
53.         // TODO Auto-generated method stub
54.         super.onCreate();
```

```
55.         am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
56.         intent = new Intent(getApplicationContext(), EnterPwdActivity.class);
57.         ContentResolver contentResolver = getContentResolver();
58.         Uri uri = Uri.parse("content://com.itheima.mobilesafe_sh2");
59.         MyContentObserver observer = new MyContentObserver(new Handler());
60.         //自定义一个内容观察者
61.         contentResolver.registerContentObserver(uri, true, observer);
62.         WatchDogReceiver receiver = new WatchDogReceiver();
63.         // 自定义一个广播。action 可以随便定义
64.         IntentFilter filter = new IntentFilter("www.heima.com");
65.         // 锁屏
66.         filter.addAction(Intent.ACTION_SCREEN_OFF);
67.         // 解锁
68.         filter.addAction(Intent.ACTION_SCREEN_ON);
69.         // 如果用户锁屏。那么就让狗休息
70.         // 注册一个让狗停下来的广播
71.         registerReceiver(receiver, filter);
72.         dao = new AppLockDao(this);
73.         // 查询出来所有在程序锁数据库里面的包
74.         mLockLists = dao.findAll();
75.         startWatchDog();
76.     }
77.     /**启动看门狗 一直在后台看着我的手机
78.     */
79.     private void startWatchDog() {
80.         new Thread() {
81.             public void run() {
82.                 flag = true;
83.                 // 永帧循环
84.                 // 死循环表示逻辑错误
85.                 while (flag) {
86.                     // 获取到正在运行的任务栈
87.                     List<RunningTaskInfo> tasks = am.getRunningTasks(1);
88.                     // 获取到最顶层应用的包名
89.                     String packageName = tasks.get(0).topActivity
90.                         .getPackageName();
91.                     // 查询数据库。看看当前的 app 有没有在程序锁的数据库里面
92.                     // 判断当前的应用程序是否需要停止保护
93.                     // if (packageName.equals(stopProtectingPackageName)) {
94.                     // // 停止保护
95.                     // System.out.println("停止保护");
96.                     // } else {
97.                     // 每次进来一个应用程序。都需要开启数据库和关闭数据库
98.                     // if (TextUtils.isEmpty(dao.findLock(packageName))) {
```



```

99.          // mLockLists 表示所有的集合
100.          // 只需要查询一次 .从内存当中查询
101.          if (mLockLists.contains(packageName)) {
102.              // 说明没有在程序锁的数据库里面
103.              if (packageName.equals(stopProtectingPackageName)) {
104.                  // 停止保护
105.              } else {
106.                  // 说明在程序锁的数据库里面
107.                  // 开启保护 跳转到程序锁密码的界面
108.                  // 服务往 activity 里面跳转的时候。必须加一个 flag
109.                  // FLAG_ACTIVITY_NEW_TASK
110.                  intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
111.                  intent.putExtra("packageName", packageName);
112.                  startActivity(intent);
113.              }
114.          } else {
115.              // 查询数据库。看看当前的 app 有没有在程序锁的数据库里面
116.              // 判断当前的应用程序是否需要停止保护
117.          }
118.          // }
119.          // 让狗休息下
120.          SystemClock.sleep(50);
121.      }
122.  };
123.  }.start();
124.  }
125.  @Override
126.  public void onDestroy() {
127.      // TODO Auto-generated method stub
128.      super.onDestroy();
129.      flag = false;
130.  }
131.  }

```

上面的代码中定义了一个观察程序锁的观察者，我们需要在程序锁 DAO 中的增加和删除方法中，当数据库中数据发生改变时进行刷新，对应的代码如下所示。

```

1.      /**
2.       * 往程序锁的数据库里面添加数据
3.       * @param packageName
4.       *         程序的包名
5.       * @return
6.       */
7.      public void add(String packageName) {
8.          // 获取到写的数据库

```

```
9.         SQLiteDatabase db = helper.getWritableDatabase();
10.         ContentValues values = new ContentValues();
11.         values.put("packagename", packageName);
12.         db.insert("lock", null, values);
13.         mContext.getContentResolver().notifyChange(Uri.parse("content://com.itheima.
14.             mobilesafe_sh2"), null);
15.     }
16.     /**
17.      * 根据包名进行删除数据
18.      * @param packageName
19.      *      电话号码
20.      * @return
21.      */
22.     public void delete(String packageName) {
23.         SQLiteDatabase db = helper.getWritableDatabase();
24.         int rowNumber = db.delete("lock", "packagename = ?", new String[] { packageName });
25.         mContext.getContentResolver().notifyChange(Uri.parse("content://com.itheima.
26.             mobilesafe_sh2"), null);
27.     }
```

运行程序，演示效果，如图 3-11 所示。

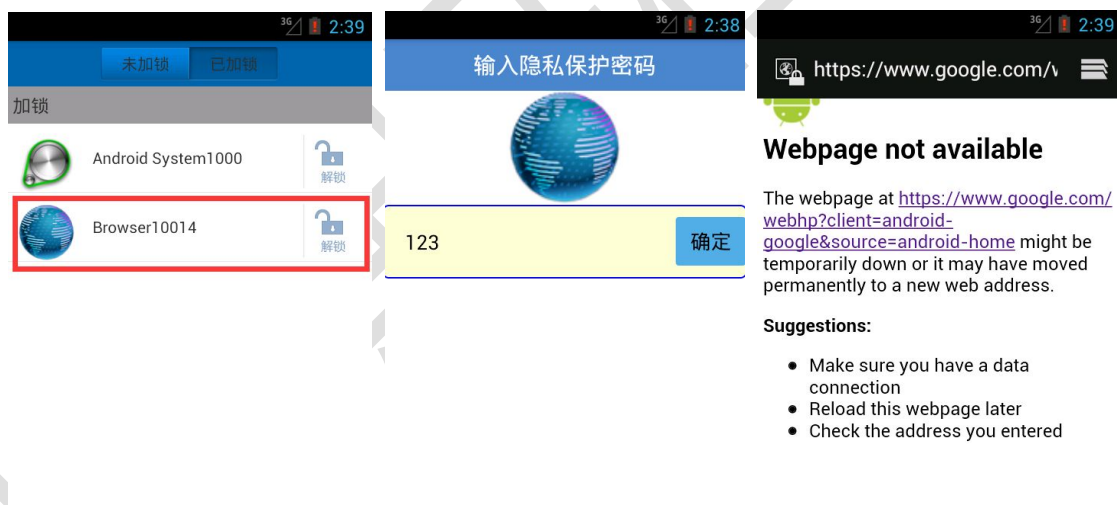


图 3-11 看门狗功能演示界面

## 3.6 数据库的使用

### 3.6.1 常用工具之查询归属地

常用工具有这样的一个查询归属地的功能，它是用于查询手机号码的归属地，功能的展示界面如图 3-12 所示。

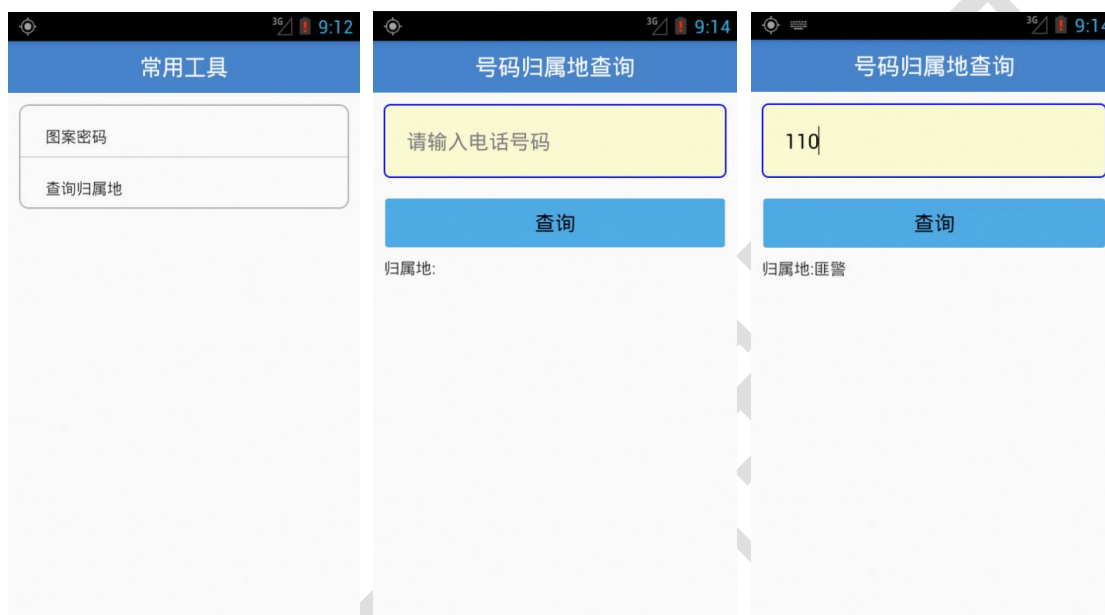
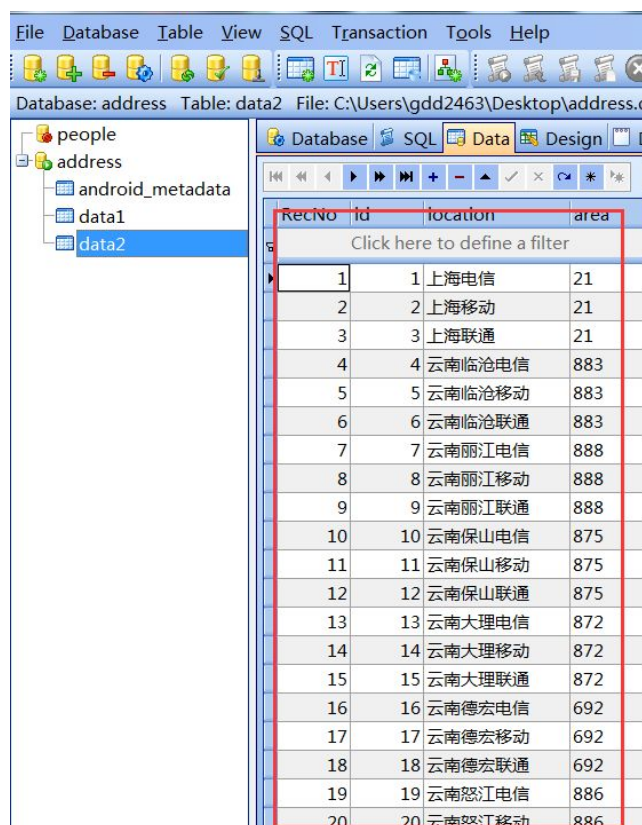


图 3-12 号码归属地查询功能

实际上，号码所查询的是一个数据中的数据，数据库的内容如图 3-13 所示，数据库 address.db 文件我们是放在 assets 目录下的。



RecNo	id	location	area
1	1	上海电信	21
2	2	上海移动	21
3	3	上海联通	21
4	4	云南临沧电信	883
5	5	云南临沧移动	883
6	6	云南临沧联通	883
7	7	云南丽江电信	888
8	8	云南丽江移动	888
9	9	云南丽江联通	888
10	10	云南保山电信	875
11	11	云南保山移动	875
12	12	云南保山联通	875
13	13	云南大理电信	872
14	14	云南大理移动	872
15	15	云南大理联通	872
16	16	云南德宏电信	692
17	17	云南德宏移动	692
18	18	云南德宏联通	692
19	19	云南怒江电信	886
20	20	云南怒江移动	886

图 3-13 号码归属地查询数据库中的内容

应用程序在欢迎界面就已将数据库文件拷贝到应用自身的数据库文件夹中，然后进行之后的一系列查询操作，这里在欢迎界面中用于拷贝数据库的代码如下所示。

```

1.  /**
2.   * 拷贝数据库
3.   * @param dbName
4.   *      数据库的名字
5.   */
6.  private void copyDB(final String dbName) {
7.      //UI(主线程) 不能做耗时间的操作, anr 无响应
8.      new Thread(){
9.          public void run() {
10.             try {
11.                 // getFilesDir()---/data/data/com.itheima.mobilesafe_sh2/files
12.                 System.out.println("getFilesDir()---" + getFilesDir());
13.                 File file = new File(getFilesDir(), dbName);
14.                 // 判断当前的文件是否已经拷贝过了
15.                 if (file.exists() && file.length() > 0) {
16.                     return;
17.                 }
18.                 // getAssets() 获取到资产目录的管理者, 打开一个流
19.                 InputStream is = getAssets().open(dbName);
20.                 FileOutputStream fos = new FileOutputStream(file);
21.                 // 初始化一个缓冲区

```

```
22.         byte[] buffer = new byte[1024];
23.         int len = 0;
24.         while ((len = is.read(buffer)) != -1) {
25.             fos.write(buffer, 0, len);
26.         }
27.         fos.close();
28.         is.close();
29.         System.out.println("拷贝完毕");
30.     } catch (Exception e) {
31.         // TODO Auto-generated catch block
32.         e.printStackTrace();
33.     }
34. };
35. }.start();
36. }
```

拷贝文件成功以后，我们需要书写常用工具界面 `AToolsActivity.java` 中的查询归属地功能，对应的布局文件如【文件 3-18】所示。

**【文件 3-18】** res/layout/activity\_atools.xml

```
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.     <TextView
8.         style="@style/TitleBarTextView"
9.         android:text="常用工具" />
10.    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
11.        android:id="@id/rl_lock_pattern_view"
12.        android:layout_width="match_parent"
13.        android:layout_height="wrap_content"
14.        android:layout_marginTop="10dp"
15.        itheima:isToggle="true"
16.        itheima:itbackground="first"
17.        itheima:title="图案密码" />
18.    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
19.        android:id="@+id/rl_number_query"
20.        android:layout_width="match_parent"
21.        android:layout_height="wrap_content"
22.        itheima:isToggle="true"
23.        itheima:itbackground="last"
24.        itheima:title="查询归属地" />
25. </LinearLayout>
```

上面定义的 `itheima:isToggle="true"` 属性，是用于控制隐藏滑动开关，它是在 `attrs.xml` 文件中增加一条属性，如下所示，正如前面所讲，`title` 用于显示控件的名称，`itbackground` 用于标示当前控件所处的位置，`isToggle` 用于标示是否显示滑动开关。

```

1. <declare-styleable name="SettingItemView">
2.     <attr name="isToggle" format="boolean" />
3.     <attr name="title" format="string" />
4.     <attr name="itbackground">
5.         <enum name="first" value="0" />
6.         <enum name="middle" value="1" />
7.         <enum name="last" value="2" />
8.     </attr>
9. </declare-styleable>
10. <declare-styleable name="LockPatternView">

```

在定义好自定义属性之后，我们需要在构造方法中将对应的属性读取出来，这里我们是根据属性 `isToggle` 的值来决定当前滑动开关的显示或者隐藏，当 `isToggle` 为 `true` 时就隐藏，反之显示。具体代码如下所示。

```

1. //获取到滑动开关
2. boolean seting_isToggle = ta.getBoolean(R.styleable.SettingItemView_isToggle,
3.     false);
4. //设置滑动开关是否隐藏
5. mIvToggle.setVisibility(seting_isToggle ? View.INVISIBLE : View.VISIBLE);

```

运行程序，效果图如图 3-14 所示。



图 3-14 常用工具中号码归属地界面

### 3.6.2 封装跳转界面工具类

由于界面的跳转经常用到，这里考虑使用一个工具类将整个方法封装起来，在 `PackageUtils.java` 中增加下面一个 `startActivity()` 方法用于跳转界面。如下所示。

```
1.      /**
2.       * 跳转
3.       */
4.      public static void startActivity(Context context ,Class<?> clazz) {
5.          Intent intent = new Intent(context,
6.              clazz);
7.          //添加一个标记
8.          intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
9.          context.startActivity(intent);
10.     }
```

### 3.6.3 常用工具之号码查询界面

这里我们需要新建一个号码归属地查询的界面 QueryAddressActivity.java, 用于显示号码归属地查询的结果, 效果图如图 3-15 所示。



图 3-15 号码归属地查询界面

对应的布局文件如文件【3-19】所示。

**【文件 3-19】** res/layout/activity\_query\_address.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="号码归属地查询" />
9.     <!-- android:inputType="phone" 输入类型只能是电话号码 -->
10.    <EditText
11.        android:id="@+id/et_phone"
12.        android:layout_width="match_parent"
13.        android:layout_height="wrap_content"
14.        android:layout_margin="10dp"
15.        android:background="@drawable/et_input_normal"
```

```
16.         android:inputType="phone"
17.         android:hint="请输入电话号码"/>
18.     <Button
19.         android:layout_width="match_parent"
20.         android:layout_height="wrap_content"
21.         android:layout_marginLeft="10dp"
22.         android:layout_marginRight="10dp"
23.         android:layout_marginTop="5dp"
24.         android:background="@drawable/dg_btn_confirm_selected"
25.         android:onClick="query"
26.         android:text="查询" />
27.     <TextView
28.         android:id="@+id/tv_location"
29.         android:layout_width="match_parent"
30.         android:layout_height="wrap_content"
31.         android:layout_marginLeft="10dp"
32.         android:layout_marginRight="10dp"
33.         android:layout_marginTop="5dp"
34.         android:text="归属地:" />
35. </LinearLayout>
```

号码归属地查询的逻辑代码如文件【3-20】所示。

**【文件 3-20】** com.itheima.mobilesafe\_sh2.act/QueryAddressActivity.java

```
1. public class QueryAddressActivity extends Activity {
2.     private EditText et_phone;
3.     private TextView tv_location;
4.     @Override
5.     protected void onCreate(Bundle savedInstanceState) {
6.         // TODO Auto-generated method stub
7.         super.onCreate(savedInstanceState);
8.         setContentView(R.layout.activity_query_address);
9.         et_phone = (EditText) findViewById(R.id.et_phone);
10.        tv_location = (TextView) findViewById(R.id.tv_location);
11.    }
12.    /**
13.     * 查询
14.     *
15.     * @param view
16.     */
17.    public void query(View view) {
18.        // 获取到电话号码
19.        String phone = et_phone.getText().toString().trim();
20.        // 判断用户是否输入电话号码
21.        if (TextUtils.isEmpty(phone)) {
22.            Animation shake = AnimationUtils.loadAnimation(this, R.anim.shake);
```



```

23.         et_phone.startAnimation(shake);
24.         ToastUtils.showSafeToast(QueryAddressActivity.this, "号码不能为空");
25.     } else {
26.         // 查询归属地
27.         String location = QueryAddressDao.getLocation(this, phone);
28.         //设置归属地
29.         tv_location.setText("归属地:" + location);
30.     }
31. }
32. }

```

其中，在点击查询号码的时候，如果号码编辑框为空时会有一个抖动的动画效果，该动画 shake.xml 放在 anim 文件下，如文件【3-21】所示。

**【文件 3-21】** res/anim/shake.xml

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <translate xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:fromXDelta="0"
4.     android:toXDelta="10"
5.     android:duration="1000"
6.     android:interpolator="@anim/cycle_7" />

```

shake.xml 中的 interpolator 是一个函数，它计算动画如何播放，它的代码如文件【3-22】所示。

**【文件 3-22】** anim/cycle\_7.xml

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <cycleInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:cycles="7" />

```

另外，当输入的号码为空时，该编辑框还会有一个震动的效果，如果是真机的话手机就会震动，实现这一个效果仅仅几行代码和一个权限即可搞定，如下所示。

```

1.         //获取到震动的管理者
2.         mVibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
3.         mVibrator.vibrate(2000); //振动两秒钟

```

实现震动效果需要一个权限，如下所示。

```

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

```

### 3.6.4 常用工具之数据库中号码查询逻辑

界面写好之后，下面就是进行具体号码查询的逻辑，这里所要查询的是数据库中的信息，创建一个应用包名为 com.itheima.mobilesafe\_sh2.dao，该包所存放的都是与数据库进行操作的逻辑代码。这里，QueryAddressDao.java 即为用于查询号码归属地的工具类，如【文件 3-23】所示。

**【文件 3-23】** com.itheima.mobilesafe\_sh2.dao/QueryAddressDao

```

1. public class QueryAddressDao {
2.     /**
3.      * 根据电话号码查询归属地
4.      * @param context
5.      * @param phone

```

```
6.      *          电话号码
7.      * @return
8.      */
9.  public static String getLocation(Context context, String phone) {
10.     String address = null;
11.     // getFilesDir()---/data/data/com.itheima.mobilesafe_sh2/files
12.     // 第一个参数：数据库的位置
13.     // 第二个参数：cursor 的工厂
14.     // 第三个参数：数据库的标记 数据库只读
15.     SQLiteDatabase db = SQLiteDatabase.openDatabase(context.getFilesDir()
16.         + "/address.db", null, SQLiteDatabase.OPEN_READONLY);
17.     // 判断当前的电话号码是否是手机号码
18.     // \\表示转义
19.     boolean matches = phone.matches("^1[34578]\\d{9}$");
20.     if (matches) {
21.         // 先查询表一：根据电话号码进行查询 outkey select outkey from data1 where id =
22.         // 1300001
23.         // 查询表二：把 outkey 做为 id 查询 location
24.         String sub_phone = phone.substring(0, 7);
25.         Cursor cursor = db.rawQuery("select location from data2 where id =(select
26. outkey from data1 where id = ?)", new String[] { sub_phone });
27.         /**
28.          * 判断手机和座机区别
29.          手机：
30.          一般是 1[34578]+9 个正整数 手机号码是 11 位
31.          因此，可以使用正则表达式来匹配手机号码，即 ^1[34578]\\d{9}$
32.          座机：
33.          1 区号有 3 位。有 4 位 也是 11 位
34.          2 110 120 119
35.          3 模拟器
36.          5 10086 10000 10010
37.          */
38.         if (cursor.moveToNext()) {
39.             // 查询地址
40.             address = cursor.getString(0);
41.         }
42.         cursor.close();
43.     } else {
44.         // 电话号码
45.         switch (phone.length()) {
46.             // 110 , 120, 119
47.             case 3:
48.                 if ("110".equals(phone)) {
49.                     address = "匪警";
```

```
50.         }
51.         break;
52.     // 模拟器
53.     case 4:
54.         if ("5556".equals(phone)) {
55.             address = "模拟器";
56.         }
57.         break;
58.     // 10000 10086 95555
59.     case 5:
60.         if ("10000".equals(phone)) {
61.             address = "中国电信";
62.         }
63.         break;
64.     default:
65.         // 电话号码
66.         address = null;
67.         String substring = phone.substring(1, 3);
68.         Cursor cursor = db.rawQuery(
69.             "select location from data2 where area = ?",
70.             new String[] { substring });
71.         if (cursor.moveToNext()) {
72.             address = cursor.getString(0);
73.         }
74.         substring = phone.substring(1, 4);
75.         cursor = db.rawQuery("select location from data2 where area = ?",
76.             new String[] { substring });
77.         if (cursor.moveToNext()) {
78.             address = cursor.getString(0);
79.         }
80.         cursor.close();
81.         break;
82.     }
83. }
84. return address;
85. }
86. }
```

创建好查询号码的工具类之后，这时可以运行程序，测试效果，如图 3-16 所示。



图 3-16 测试号码归属地查询

### 3.6.5 s Shape 的使用

在输入电话号码的编辑框，如果是没有经过绘制的效果图如下图 3-17（a）所示，这里为了让编辑框更加美观，使用 shape 属性绘制编辑框的形状，效果图如图 3-17（b）所示。

为了实现这样的效果我们需要在布局文件中设置编辑框的背景，改写原有的布局文件，修改的代码如下所示。



图 3-17 号码归属地查询的编辑框效果

```
1. <EditText
2.     android:id="@+id/et_phone"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content"
5.     android:layout_margin="10dp"
6.     <!-- 设置编辑框的背景效果 -->
7.     android:background="@drawable/et_input_normal"
8.     android:inputType="phone"
9.     android:hint="请输入电话号码"/>
```

其中，编辑框的背景文件 et\_input\_normal.xml，具体的代码如文件【3-24】所示

【文件 3-24】 drawable/et\_input\_normal.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:shape="rectangle" >
4.     <!--
5.         rectangle 矩形
```

```
6.      oval 圆角椭圆
7.      line 线
8.      ring 环行
9.      -->
10.     <!--
11.      gradient 梯度渐变色
12.      设置开始中间和结束的颜色
13.      -->
14.     <!-- <gradient -->
15.     <!-- android:centerColor="#0000ff" -->
16.     <!-- android:endColor="#ff00ff00" -->
17.     <!-- android:startColor="#ff000000" /> -->
18.     <!-- 圆角边角 -->
19.     <corners android:radius="5dp" />
20.     <!-- 固定色 不能和渐变色一起用。不然会有冲突 -->
21.     <solid android:color="#ffffd6" />
22.
23.     <!-- 内容距离控件的位置 -->
24.     <padding
25.         android:bottom="20dp"
26.         android:left="20dp"
27.         android:right="20dp"
28.         android:top="20dp" />
29.     <!--
30.         描边 android:width="1dp"
31.         android:color="#000"实体边框
32.         dashWidth 虚线边框
33.     -->
34.     <stroke
35.         android:width="1dp"
36.         android:color="#0000ff" />
37.     <size android:height="48dp" />
38. </shape>
```

### 3.6.6 文本动态监听

为实现文本动态改变的监听，这里需要实现一个文本改变的监听接口，具体的代码如下所示。

```
1.      //添加文本改变的监听
2.      et_phone.addTextChangedListener(new TextWatcher() {
3.          //文本改变之前调用的方法
4.          @Override
5.          public void beforeTextChanged(CharSequence s, int start, int count,
6.              int after) {
```

```

7.          // TODO Auto-generated method stub
8.          System.out.println("beforeTextChanged"+s.toString());
9.      }
10.     //文本正在改变的时候调用
11.     @Override
12.     public void onTextChanged(CharSequence s, int start, int before,
13.         int count) {
14.         // TODO Auto-generated method stub
15.         System.out.println("onTextChanged");
16.     }
17.     //文本结束之后调用的方法
18.     @Override
19.     public void afterTextChanged(Editable s) {
20.         // s 表示当前输入的文本
21.         System.out.println("afterTextChanged"+s.toString());
22.         String location = QueryAddressDao.getLocation(QueryAddressActivity.this,
23.             s.toString());
24.         //设置归属地
25.         tv_location.setText("归属地:" + location);
26.     }
27. }

```

上面的代码存在一个小 Bug，当输入的号码为 1 位时，会报出下面的异常，如图 3-18 所示。



图 3-18 查询号码为 1 位时爆出的异常

上面的代码是由于 String 截取异常造成的，当号码为 1 位时，截取位数越界，因此需要将之前的代码进行一定的修改，修改 QueryAddressDao.java 中 getLocation() 方法对电话号码逻辑的判断，如下所示。

```

1.     default:
2.         // 电话号码
3.         address = null;
4.         //判断当前的电话号码必须要大于 7 位 电话号码必须 0 开头
5.         if(phone.length() > 7 && phone.startsWith("0")){

```

```
6.         String substring = phone.substring(1, 3);
7.         Cursor cursor = db.rawQuery(
8.             "select location from data2 where area = ?",
9.             new String[] { substring });
10.        if (cursor.moveToNext()) {
11.            address = cursor.getString(0);
12.        }
13.        substring = phone.substring(1, 4);
14.        cursor = db.rawQuery("select location from data2 where area = ?",
15.            new String[] { substring });
16.        if (cursor.moveToNext()) {
17.            address = cursor.getString(0);
18.        }
19.        cursor.close();
20.        break;
21.    }
22. }
```

修改好代码之后，运行程序，输入一些号码观察是否能在不点击查询按钮的情况下动态查出结果，如图 3-19 所示。



图 3-19 号码动态变化时查询归属地

## 3.7 常用工具之归属地显示

### 3.7.1 归属地显示界面

前面我们实现了号码的归属地查询的功能，为了能够在不同页面实现来电号码的归属地显示功能，这里在常用设置中增加了归属地显示设置的按钮，点击此按钮后会开启来电归属地显示，同时可以选择归属地显示的样式风格，点击该按钮会从屏幕下方弹出一个选择归属地样式的对话框，效果图如图 3-20 所示。



图 3-20 归属地显示样式的选择

无论当前处于哪个界面，当来电时都会在来电界面上显示归属地，能够解决这个问题的方法就是在后台运行一个服务进程，该服务进程能够在来电时及时地在来电界面上显示归属地。

这时常用设置界面 `SettingActivity` 所对应的布局文件如文件【3-25】所示。

【文件 3-25】 `res/layout/activity_setting.xml`

```
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.     <TextView
8.         style="@style/TitleBarTextView"
9.         android:text="设置中心" />
10.    <!-- android:layout_alignParentRight="true" 在屏幕的右侧 -->
11.    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
12.        android:id="@+id/sv_auto_date"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:layout_marginTop="10dp"
16.        itheima:itbackground="first"
17.        itheima:title="自动更新" />
18.    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
19.        android:id="@+id/siv_address"
20.        android:layout_width="match_parent"
21.        android:layout_height="wrap_content"
22.        itheima:itbackground="middle"
23.        itheima:title="归属地显示设置" />
24.    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
25.        android:id="@+id/siv_address_style"
26.        android:layout_width="match_parent"
```



```

27.         android:layout_height="wrap_content"
28.         itheima:isToggle="true"
29.         itheima:itbackground="last"
30.         itheima:title="归属地设置风格" />
31. </LinearLayout>

```

布局文件写好之后，在 `SettingActivity` 的 `onCreate()` 方法中拿到对应的 `id`，实现点击事件，具体修改增加的代码如下所示，注意：当前的 `SettingActivity` 已实现 `OnClickListener` 接口。

```

1.         //来点归属地
2.         siv_address = (SettingItemView) findViewById(R.id.siv_address);
3.         //设置来点归属地的监听
4.         siv_address.setOnClickListener(this);
5.         //来点归属地样式
6.         siv_address_style = (SettingItemView) findViewById(R.id.siv_address_style);
7.         siv_address_style.setOnClickListener(this);

```

接着就需要在 `onClick()` 方法中根据对应的 `id` 进行对应的逻辑代码设计，具体如下所示。

```

1. @Override
2.     public void onClick(View v) {
3.         switch (v.getId()) {
4.             case R.id.siv_address:
5.                 Intent addressIntent = new Intent(this,AddressServices.class);
6.                 // 设置滑动开关。如果是开着的点击关闭。如果是关闭的。点击打开
7.                 if(ServiceStateUtils.serviceRunning(SettingActivity.this,
8. AddressServices.class)){
9.                     siv_address.setToggle(false);
10.                    stopService(addressIntent);
11.                }else{
12.                    siv_address.setToggle(true);
13.                    startService(addressIntent);
14.                }
15.                break;
16.             case R.id.siv_address_style:
17.                 break;
18.         }
19.     }

```

此时，运行程序发现，如图 3-21 所示。当点击开启归属地显示按钮后如图 3-21 (a)，退出常用设置界面再次进入时，滑动开关的状态仍然处于关闭状态如图 3-21 (b)。显然，这涉及到滑动开关状态的回显设置，但是在这里我们不能使用前面所用的 `SharedPreferences` 进行状态信息的存储，因为该按钮的开关状态应于后台是否有查询号码的服务进程有关，如果使用固定数值存储，当手机的其他服务进程将安全卫士的归属地查询的服务杀死后，由于之前 `sp` 中固定数值的存储，回显时仍然显示该服务进程开启，这就造成该功能未能准确实现。



图 3-21 归属地显示设置的错误回显

为解决上述问题，这里我们需要判断手机后台中是否有我们开启的归属地查询的服务，以此来决定回显时的状态信息，我们在工具类中创建一个 `ServiceStateUtils.java` 文件，它用于判断当前的某个服务是否开启，对应的文件【3-26】如下所示，其中 `? extends Service` 限制传入的参数必须是继承 `Service` 的服务类。

**【文件 3-26】** `com.itheima.mobilesafe_sh2.utils/ServiceStateUtils.java`

```

1. public class ServiceStateUtils {
2.     /**
3.      * 判断服务
4.      * @param context
5.      * @param clazz
6.      *      服务的名字 ? extends Service
7.      * @return
8.      */
9.     public static boolean serviceRunning(Context context,
10.         Class<? extends Service> clazz) {
11.         // 进程管理器(任务管理器)
12.         ActivityManager am = (ActivityManager) context
13.             .getSystemService(Context.ACTIVITY_SERVICE);
14.         // 返回正在运行的进程
15.         // 获取正在运行的服务
16.         List<RunningServiceInfo> runningServices = am.getRunningServices(50);
17.
18.         // 迭代所有的任务管理器
19.         for (RunningServiceInfo info : runningServices) {
20.             // 获取到当前的服务
21.             ComponentName service = info.service;
22.             // 获取到服务的名字
23.             String className = service.getClassName();
24.             // System.out.println("className---" + className);
25.             // System.out.println("clazz----" + clazz);
26.             // 判断当前的服务名字是否一致
27.             if (className.equals(clazz.getName())) {
28.                 return true;
29.             }

```

```
30.     }
31.     return false;
32. }
33. }
```

为解决归属地显示设置的回显状态，我们复写 `SettingActivity` 的 `onStart()` 方法，在该方法中判断当前手机的服务进程中是否有归属地查询的服务进程，从而决定当前滑动开关的状态，如下所示。

```
1. @Override
2.     protected void onStart() {
3.         // TODO Auto-generated method stub
4.         super.onStart();
5.         //判断当前的服务是否已经启动
6.         if(ServiceStateUtils.serviceRunning(SettingActivity.this, AddressServices.
7. class)){
8.             siv_address.setToggle(true);
9.         }else{
10.             siv_address.setToggle(false);
11.         }
12.     }
```

此时，运行程序，发现可以正常实现归属地查询的回显状态，如图 3-22 所示。

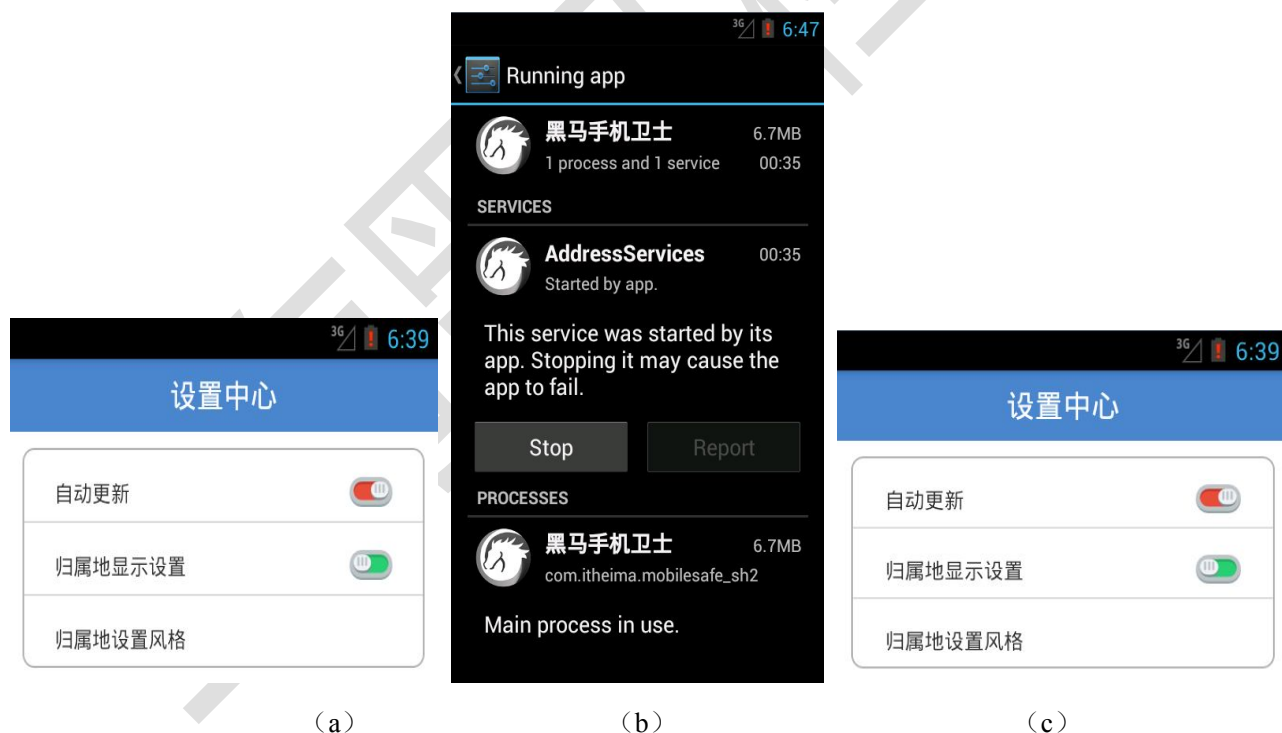


图 3-22 归属地显示设置的正确回显

其中，`AddressServices` 是创建的一个用于查询归属地位置的服务，它在手机后台进行归属地查询的逻辑，具体的代码如文件【3-27】所示。

【文件 3-27】 com.itheima.mobilesafe\_sh2.service/AddressServices.java

```
1 public class AddressServices extends Service {
2     private TelephonyManager tm;
```

```
3     private MyPhoneStateListener listener;
4     private AddressTost toast;
5     @Override
6     public IBinder onBind(Intent intent) {
7         // TODO Auto-generated method stub
8         return null;
9     }
10    private class MyPhoneStateListener extends PhoneStateListener {
11        private WindowManager mWM;
12        private TextView mView;
13
14        // 状态改变的方法
15        // 第一个参数：电话状态
16        // 第二个参数：电话号码
17        @Override
18        public void onCallStateChanged(int state, String incomingNumber) {
19            switch (state) {
20                case TelephonyManager.CALL_STATE_IDLE:
21                    // 电话闲置状态
22                    System.out.println("TelephonyManager.CALL_STATE_IDLE");
23                    break;
24
25                case TelephonyManager.CALL_STATE_OFFHOOK:
26                    // 电话接通的状态
27                    System.out.println("TelephonyManager.CALL_STATE_OFFHOOK");
28                    break;
29
30                case TelephonyManager.CALL_STATE_RINGING:
31                    // 电话铃响状态
32                    System.out.println("TelephonyManager.CALL_STATE_RINGING");
33                    // 第二个参数表示电话号码
34                    String location = QueryAddressDao.getLocation(
35                        getApplicationContext(), incomingNumber);
36                    //使用吐司将归属地显示出来
37                    Toast.makeText(getApplicationContext(), "归属地"+location, 1).show();
38                    break;
39            }
40        }
41    }
42    @Override
43    public void onCreate() {
44        // TODO Auto-generated method stub
45        super.onCreate();
46        // 获取电话管理者
```

```
47     tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
48     //初始化状态的监听器
49     listener = new MyPhoneStateListener();
50     /**
51      * 第一个参数电话状态的监听
52      */
53     tm.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
54 }
55 @Override
56 public void onDestroy() {
57     // TODO Auto-generated method stub
58     super.onDestroy();
59     // 关闭当前电话的状态
60     tm.listen(listener, PhoneStateListener.LISTEN_NONE);
61     listener = null;
62 }
63 }
```

为读取手机的通话状态，需要加上一个权限，如下所示。

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

运行程序，打开归属地显示设置按钮，在 DDMS 下给模拟器模拟打电话，下面是一组测试，如图 3-23 可知，来电的页面能够显示出来电号码的正确归属地。但是提示框会在显示一段时间之后会消失，但是我们希望这个吐司能够一直不消失。因此，我们需要自定义一个吐司。

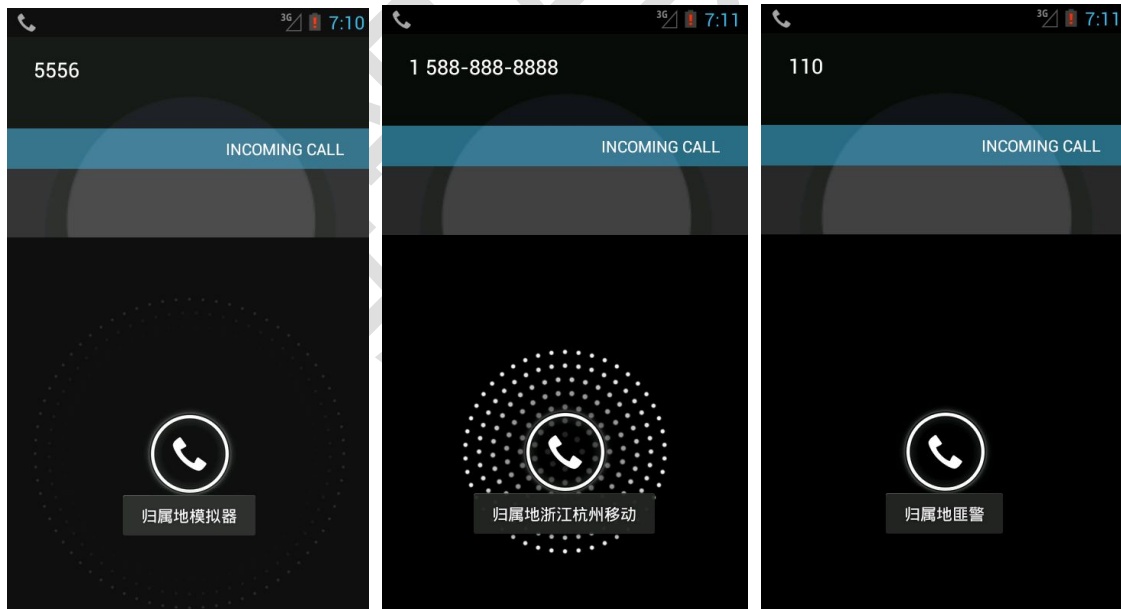


图 3-23 归属地显示

### 3.7.2 自定义吐司的简单实用

由上述可知，我们需要自定义一个吐司让归属地显示能够一直不消失，并在此基础上进行一定样式的

修改，下面我们先开始从系统的吐司源码中抽取我们所需要的代码，进行改写后的代码如下所示。

```
1. private class MyPhoneStateListener extends PhoneStateListener {
2.     private WindowManager mWM;
3.     private WindowManager.LayoutParams mParams;
4.     private TextView mLocation;
5.
6.     // 状态改变的方法
7.     // 第一个参数：电话状态
8.     // 第二个参数：电话号码
9.     @Override
10.    public void onCallStateChanged(int state, String incomingNumber) {
11.        switch (state) {
12.            case TelephonyManager.CALL_STATE_IDLE:
13.                // 电话闲置状态
14.                System.out.println("TelephonyManager.CALL_STATE_IDLE");
15.                break;
16.
17.            case TelephonyManager.CALL_STATE_OFFHOOK:
18.                // 电话接通的状态
19.                System.out.println("TelephonyManager.CALL_STATE_OFFHOOK");
20.                break;
21.
22.            case TelephonyManager.CALL_STATE_RINGING:
23.                // 电话铃响状态
24.                System.out.println("TelephonyManager.CALL_STATE_RINGING");
25.                // 第二个参数表示电话号码
26.                String location = QueryAddressDao.getLocation(
27.                    getApplicationContext(), incomingNumber);
28.                // Toast.makeText(getApplicationContext(), "归属地"+location, 1).show();
29.                //下面都是从系统源码中抽取出的显示吐司的源码
30.
31.                mWM = (WindowManager) getSystemService(Context.WINDOW_SERVICE);
32.                mParams = new WindowManager.LayoutParams();
33.                WindowManager.LayoutParams params = mParams;
34.
35.                params.height = WindowManager.LayoutParams.WRAP_CONTENT;
36.                params.width = WindowManager.LayoutParams.WRAP_CONTENT;
37.                params.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE
38.                    | WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE
39.                    | WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON;
40.                params.format = PixelFormat.TRANSLUCENT;
41.                // params.windowAnimations =
42.                // com.android.internal.R.style.Animation_Toast;
43.                params.type = WindowManager.LayoutParams.TYPE_TOAST;
```

```

44.
45.         mLocation=new TextView(getApplicationContext());
46.         // 设置归属地的颜色
47.         mLocation.setBackgroundColor(Color.RED);
48.         // 设置归属地的位置
49.         mLocation.setText(location);
50.         mWM.addView(mLocation, mParams);
51.         break;
52.     }
53. }
54. }

```

运行程序，向模拟器打电话，结果如下图 3-24 所示，当有来电时手机会弹出一个我们定义好样式的归属地，如这里我们定义的红色文本。但是当我们挂断电话后，这个吐司一直都存在，没有消失。

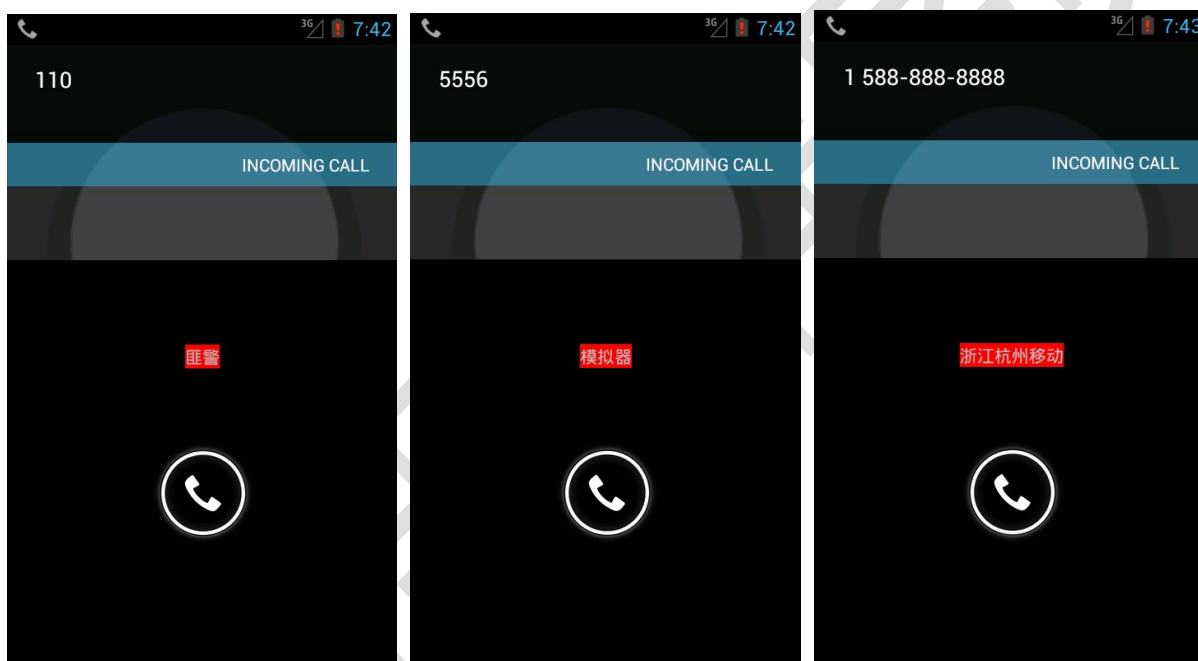


图 3-24 来电吐司的显示

为解决上面挂断电话时需让吐司消失的效果，我们需要在电话闲置状态下将吐司隐藏，修改代码，如下所示。

```

1.     case TelephonyManager.CALL_STATE_IDLE:
2.         // 电话闲置状态
3.         System.out.println("TelephonyManager.CALL_STATE_IDLE");
4.
5.         // 隐藏吐司
6.         if (mLocation != null) {
7.             // note: checking parent() just to make sure the view has
8.             // been added... i have seen cases where we get here when
9.             // the view isn't yet added, so let's try not to crash.
10.            if (mLocation.getParent() != null) {
11.                // if (localLOGV) Log.v(TAG, "REMOVE! " + mView + " in " +

```

```
12.         // this);
13.         mWM.removeView(mLocation);
14.     }
15.     mLocation = null;
16. }
17. break;
```

运行程序，模拟打电话，结果如图 3-25 所示，来电和通话中显示吐司，而闲置状态时隐藏吐司。

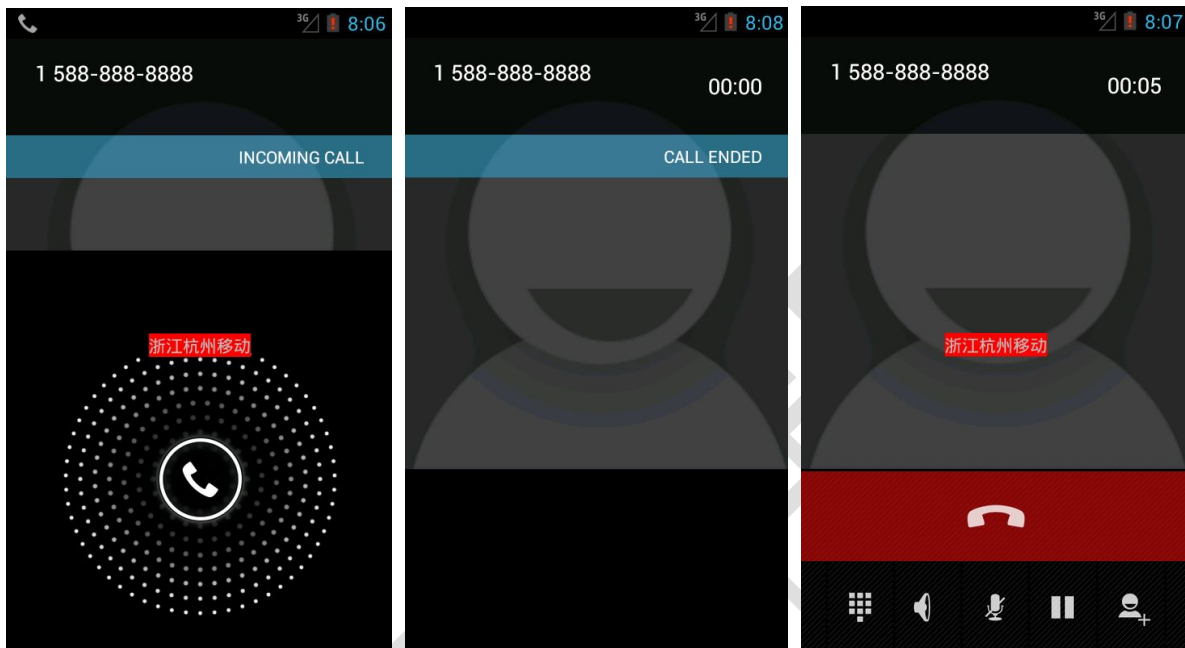


图 3-25 来电吐司的显示和隐藏

### 3.7.3 封装自定义吐司的实现

上面我们已经将自定义吐司的效果做出来了，但是代码比较混乱，这里我们考虑将自定义的吐司封装起来，以后随时可以拿来使用。在自定控件的包 `com.itheima.mobilesafe_sh2.act.view` 下创建文件 `AddressTost.java`，封装好的代码如文件【3-28】所示。

**【文件 3-28】** `com.itheima.mobilesafe_sh2.act.view/AddressTost.java`

```
1. public class AddressTost implements onTouchListener {
2.     private WindowManager mWM;
3.     private Context mContext;
4.     private WindowManager.LayoutParams mParams;
5.     private TextView mLocation;
6.     private TextView mView;
7.
8.     public AddressTost(Context context) {
9.         this.mContext = context;
10.        mWM = (WindowManager) context.getSystemService(Context.WINDOW_SERVICE);
11.        mParams = new WindowManager.LayoutParams();
12.        WindowManager.LayoutParams params = mParams;
```



```
13.         params.height = WindowManager.LayoutParams.WRAP_CONTENT;
14.         params.width = WindowManager.LayoutParams.WRAP_CONTENT;
15.         params.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE
16. //         | WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE //需要 toast 有触摸事件
17.         | WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON;
18.         params.format = PixelFormat.TRANSLUCENT;
19.         // params.windowAnimations =
20.         // com.android.internal.R.style.Animation_Toast;
21. //         params.type = WindowManager.LayoutParams.TYPE_TOAST;//土司天生没有触摸事件
22.         params.type = WindowManager.LayoutParams.TYPE_PRIORITY_PHONE;//换成电话的类型
23.     }
24.
25.     /**
26.      * 展示号码归属地显示
27.      */
28.     public void show(String location) {
29.         mLocation = (TextView) view.findViewById(R.id.tv_location);
30.         mView = new TextView(mContext);
31.         // 设置归属地的颜色
32.         mView.setBackgroundColor(Color.RED);
33.         // 设置归属地的位置
34.         mLocation.setText(location);
35.         mWM.addView(mView, mParams);
36.     }
37.     /**
38.      * 隐藏号码归属地
39.      */
40.     public void hide() {
41.         // 隐藏土司
42.         if (mView!= null) {
43.             // note: checking parent() just to make sure the view has
44.             // been added... i have seen cases where we get here when
45.             // the view isn't yet added, so let's try not to crash.
46.             if (mView.getParent() != null) {
47.                 // if (localLOGV) Log.v(TAG, "REMOVE! " + mView + " in " +
48.                 // this);
49.                 mWM.removeView(mView);
50.             }
51.             view = null;
52.         }
53.     }
54. }
```

另外，在 AddressServices 中也要进行一些修改，如下所示，改动部分见粗字体标注。

```
1. public class AddressServices extends Service {
2.     private TelephonyManager tm;
3.     private MyPhoneStateListener listener;
4.     private AddressTost toast;
5.     @Override
6.     public IBinder onBind(Intent intent) {
7.         // TODO Auto-generated method stub
8.         return null;
9.     }
10.    private class MyPhoneStateListener extends PhoneStateListener {
11.        // 状态改变的方法
12.        // 第一个参数：电话状态
13.        // 第二个参数：电话号码
14.        @Override
15.        public void onCallStateChanged(int state, String incomingNumber) {
16.            switch (state) {
17.                case TelephonyManager.CALL_STATE_IDLE:
18.                    // 电话闲置状态
19.                    System.out.println("TelephonyManager.CALL_STATE_IDLE");
20.                    //隐藏归属地地址
21.                    toast.hide();
22.                    break;
23.                case TelephonyManager.CALL_STATE_OFFHOOK:
24.                    // 电话接通的状态
25.                    System.out.println("TelephonyManager.CALL_STATE_OFFHOOK");
26.                    break;
27.                case TelephonyManager.CALL_STATE_RINGING:
28.                    // 电话铃响状态
29.                    System.out.println("TelephonyManager.CALL_STATE_RINGING");
30.                    // 第二个参数表示电话号码
31.                    String location = QueryAddressDao.getLocation(
32.                        getApplicationContext(), incomingNumber);
33.                    //展示归属地地址
34.                    toast.show(location);
35.                    break;
36.            }
37.        }
38.    }
39.    @Override
40.    public void onCreate() {
41.        // TODO Auto-generated method stub
42.        super.onCreate();
43.        // 获取电话管理者
```

```
44.         tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
45.         //初始化状态的监听器
46.         listener = new MyPhoneStateListener();
47.         //初始化归属地的土司
48.         toast = new AddressTost(getApplicationContext());
49.         /**
50.          * 第一个参数电话状态的监听
51.          */
52.         tm.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
53.     }
54.
55.     @Override
56.     public void onDestroy() {
57.         // TODO Auto-generated method stub
58.         super.onDestroy();
59.         // 关闭当前电话的状态
60.         tm.listen(listener, PhoneStateListener.LISTEN_NONE);
61.         listener = null;
62.     }
63. }
```

### 3.7.4 封装自定义吐司的美化

上面的代码跟上面的效果一样的，但是可以看到上面的吐司看着不是十分好看，这里我们打算美化一下吐司的效果。可知，我们可以通过布局填充器将一个布局文件填充到吐司中，这样也就可以在这个布局文件做一些美化的操作，修改的代码如下所示。

```
1.     /**
2.      * 展示号码归属地显示
3.      */
4.     public void show(String location) {
5.         view = View.inflate(mContext, R.layout.toast_address, null);
6.         // 设置触摸事件
7.         view.setOnTouchListener(this);
8.         mLocation = (TextView) view.findViewById(R.id.tv_location);
9.         // mView = new TextView(mContext);
10.        // // 设置归属地的颜色
11.        // mView.setBackgroundColor(Color.RED);
12.        // // 设置归属地的位置
13.        mLocation.setText(location);
14.        mWM.addView(view, mParams);
15.    }
```

其中，toast\_address.xml 是用于填充吐司的布局文件，内容如文件【3-29】所示。

【文件 3-29】 res/layout/toast\_address.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. >
7.
8.     <TextView
9.         android:id="@+id/tv_location"
10.        android:layout_width="wrap_content"
11.        android:layout_height="wrap_content"
12.        android:text="北京电信"
13.        android:background="@drawable/call_locate_blue"
14.        android:textColor="#fff"
15.        android:gravity="center"
16.        android:drawableLeft="@drawable/main_icon_36"/>
17. </LinearLayout>
```

其中，call\_locate\_blue 和 main\_icon\_36 是吐司的背景和左边的图片，运行程序，效果如图 3-26 所示。

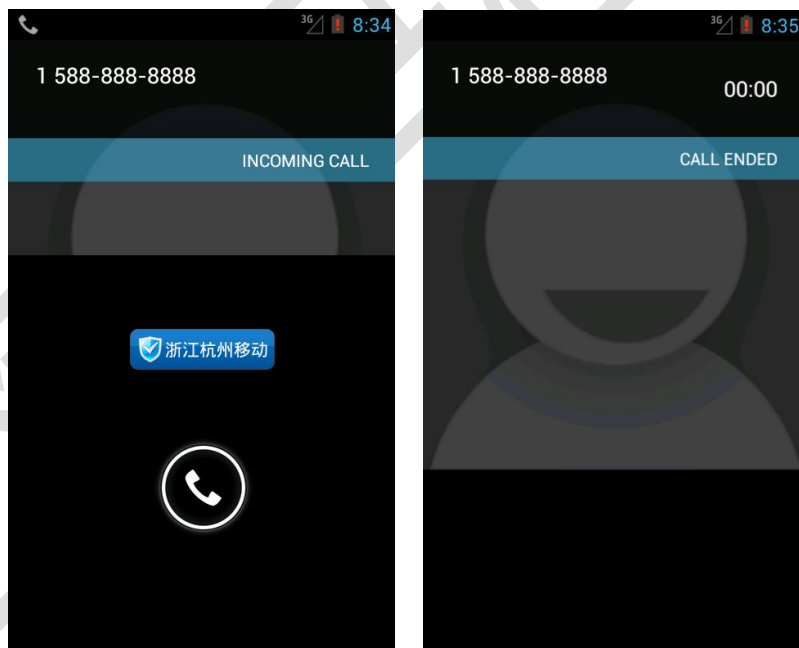


图 3-26 吐司美化之后的效果

### 3.7.5 封装自定义吐司的移动

这里，我们将实现吐司的移动，为实现这样的效果就需要实现吐司的 onTouch() 方法，即在吐司的触摸事件中实现这一效果，实现 onTouch() 之前，前面的自定义吐司的代码我们需要进行一定的更改，如下所示，粗体标注的部分。

```
1. public AddressTost(Context context) {
2.     this.mContext = context;
3.     mWM = (WindowManager) context.getSystemService(Context.WINDOW_SERVICE);
4.     mParams = new WindowManager.LayoutParams();
5.     WindowManager.LayoutParams params = mParams;
6.     params.height = WindowManager.LayoutParams.WRAP_CONTENT;
7.     params.width = WindowManager.LayoutParams.WRAP_CONTENT;
8.
9.     params.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE
10.    // | WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE //需要 toast 有触摸事件
11.    | WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON;
12.     params.format = PixelFormat.TRANSLUCENT;
13.     // params.windowAnimations =
14.     // com.android.internal.R.style.Animation_Toast;
15.
16.    // params.type = WindowManager.LayoutParams.TYPE_TOAST;//吐司天生没有触摸事件
17.    params.type = WindowManager.LayoutParams.TYPE_PRIORITY_PHONE;//换成电话的类型
18. }
```

关于控件的触摸移动的情况，可以观察图 3-27 的详细分析。



图 3-27 控件的触摸移动分析

接下来，在实现的 onTouch() 方法中，我们按照上面的分析步骤进行书写即可，代码如下所示。

```
1. /**
2.  * 归属地的触摸事件
3.  */
4. @Override
```

```
5.     public boolean onTouch(View v, MotionEvent event) {
6.         switch (event.getAction()) {
7.             // 手指按下去调用
8.             case MotionEvent.ACTION_DOWN:
9.                 //获取到开始的 x 轴和 y 轴的值
10.                startX = (int) (event.getRawX() + 0.5f);
11.                startY = (int) (event.getRawY() + 0.5f);
12.
13.                System.out.println("ACTION_DOWN");
14.                break;
15.            // 手指抬起
16.            case MotionEvent.ACTION_UP:
17.                System.out.println("ACTION_UP");
18.                break;
19.            // 手指移动
20.            case MotionEvent.ACTION_MOVE:
21.                System.out.println("ACTION_MOVE");
22.                //获取到新的 x 轴和 y 轴的值
23.                int newX = (int) (event.getRawX() + 0.5f);
24.                int newY = (int) (event.getRawY() + 0.5f);
25.
26.                //求他们之间的差值
27.                int diffX = newX - startX;
28.                int diffY = newY - startY;
29.                //重新给归属地赋值
30.                mParams.x += diffX;
31.                mParams.y += diffY;
32.
33.                //更新当前的参数
34.                mWM.updateViewLayout(view, mParams);
35.                startX = (int) (event.getRawX() + 0.5f);
36.                startY = (int) (event.getRawY() + 0.5f);
37.                break;
38.        }
39.        return true;
40.    }
```

#### 注意：

`getX()` 是表示 `view` 相对于自身左上角的 `x` 坐标, 而 `getRawX()` 是表示相对于屏幕左上角的 `x` 坐标值(这个屏幕左上角是手机屏幕左上角, 不管 `activity` 是否有 `titleBar` 或是否全屏幕)。

运行程序, 效果如图 3-28 所示, 可以拖动吐司进行移动, 而且拖动后再次来电时, 显示的位置是上次拖动结束时的位置。

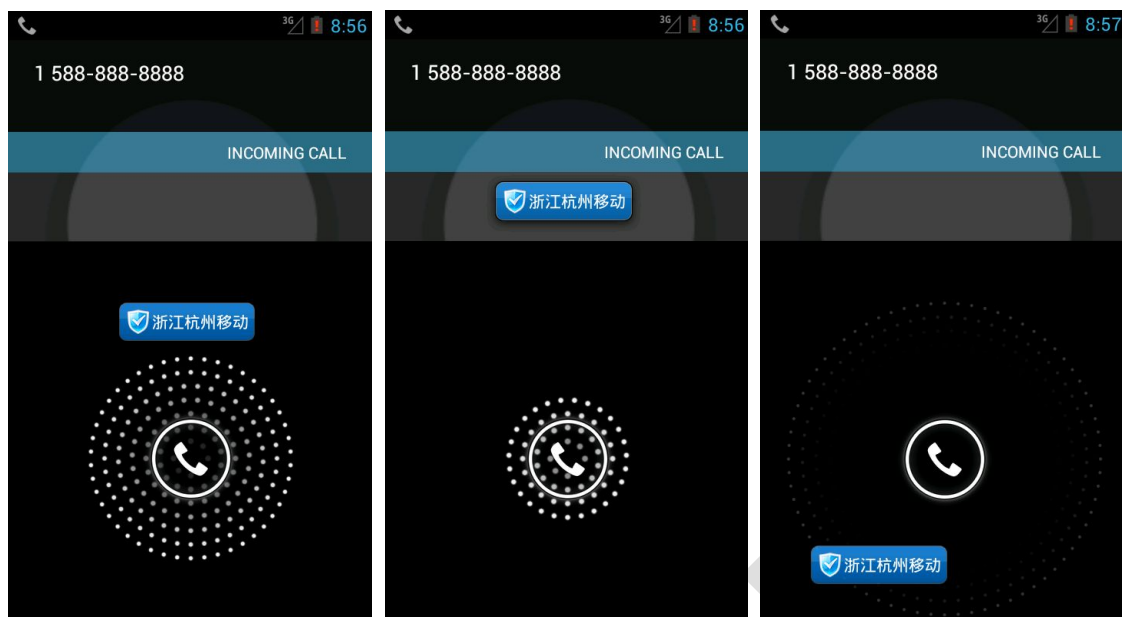


图 3-28 吐司的移动效果

### 3.8 本章小结

本章主要是针对手机卫士中的常用工具中的号码归属地查询功能、归属地显示样式、程序锁和看门狗进行讲解。