第 7 章 设置中心

第 7 章 设置中心模块

- **了解常用设置模块功能**
- **掌握数据库的创建与使用**
- **掌握自定义控件的使用**

在日常生活中，大家使用手机上网时用的基本都是 3G/4G 网络，虽然速度比 2G 网络要快很多，但也消耗了大量手机流量，并且 Android 中有些 APP 还会在后台偷偷的联网耗费流量，而每个月手机流量都是固定的，如果超过了上限会造成一定的经济损失，因此流量使用也是大家非常关心的问题，本章将针对流量统计模块进行详细讲解。

7.1 自定义标题样式

7.1.1 功能界面的跳转

在本章中，实现在主界面 MainActivity.java 的 GridView 的点击事件中常用设置界面的跳转，代码如下所示：

```
1.      @Override
2.      public void onItemClick(AdapterView<?> parent, View view, int position,
3.              long id) {
4.          switch (position) {
5.          case 0:
6.              AlertDialog.Builder builder = new Builder(this);
7.              builder.setTitle("xxxx");
8.              builder.setMessage("xxxx");
9.              builder.setNegativeButton("xxxx", new OnClickListener() {
10.                 @Override
11.                 public void onClick(DialogInterface dialog, int which) {
12.                     // TODO Auto-generated method stub
13.                 }
14.             });
15.             builder.setPositiveButton("aaa", new OnClickListener() {
16.                 @Override
17.                 public void onClick(DialogInterface dialog, int which) {
18.                     // TODO Auto-generated method stub
19.                 }
```

```
            });
            builder.show();
            break;
        // 常用设置界面
         case 8:
            startActivity(SettingActivity.class);
            break;
         }
    }
    /**
     * 启动页面
     * @param clazz
     */
    private void startActivity(Class<?> clazz) {
        Intent intent = new Intent(this, clazz);
        startActivity(intent);
    }
```

其中，startActivity()方法是抽取出来跳转指定界面的方法，传入参数中？代表泛型即可变，也就是说该方法可以根据传入的 Activity 字节码文件的不同跳转至指定的界面。

7.1.2 传统的标题样式

观察下面完整项目的几大功能界面效果，如图 7-1 所示，上方都存在一个具有同样样式的标题，传统的我们可以使用 TextView 的一些属性直接实现这样的效果，如下所示：

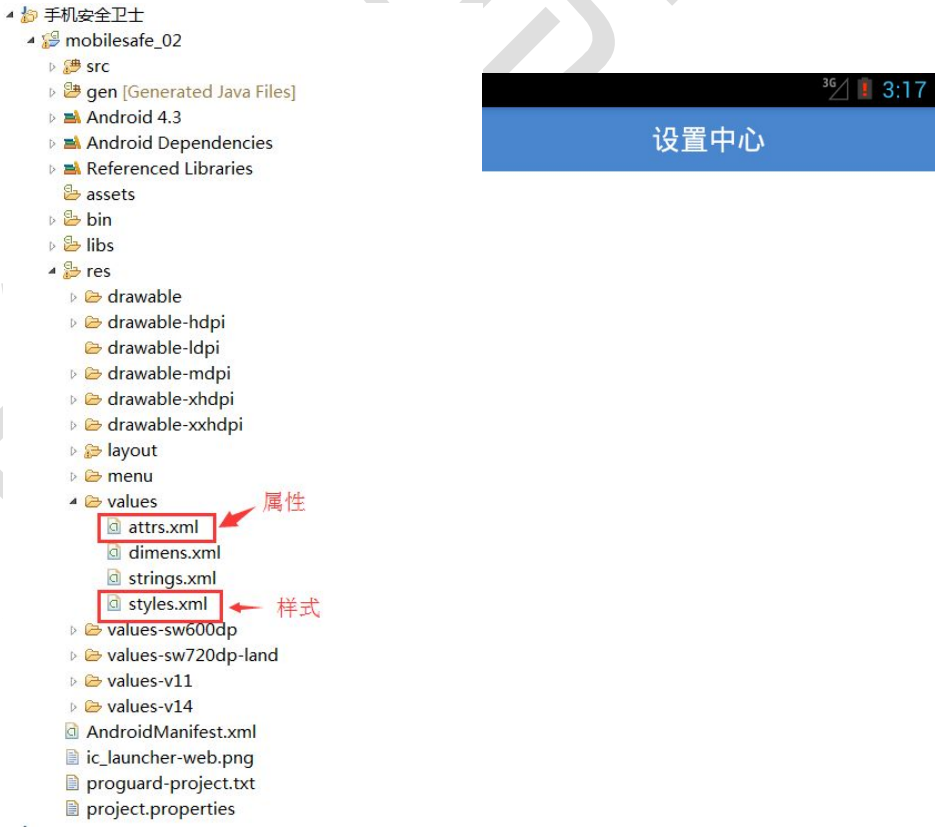
图 7-1 功能界面展示

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

```
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.      android:orientation="vertical"
7.       >
8.   <TextView
9.      android:layout_width="match_parent" -->
10.     android:layout_height="45dp" -->
11.     android:text="设置中心" -->
12.     android:gravity="center" -->
13.     android:textColor="#fff" -->
14.     android:textSize="20sp" -->
15.     android:background="#4A86CE" -->
16.     android:text="设置中心"
17.    />
18.  </LinearLayout>
```

7.1.3 自定义的标题样式

考虑到代码复用性问题，这里我们采用自定义的样式来实现各个功能界面的标题样式效果。样式的内容代码放在项目的 values 目录的 syles 文件，如图 7-2（a）所示。在 styles.xml 文件中我们自定义一个名为 TitleBarTextView 的样式。



（a）　　（b）

图 7-2　syles 样式和标题

```
<!-- android:layout_width="match_parent" -->
<!-- android:layout_height="45dp" -->
<!-- android:text="设置中心" -->
<!-- android:gravity="center" -->
<!-- android:textColor="#fff" -->
<!-- android:textSize="20sp" -->
<!-- android:background="#4A86CE" -->
<!-- 自定义样式名称 -->
<style name="TitleBarTextView">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">45dp</item>
    <item name="android:gravity">center</item>
    <item name="android:textColor">#fff</item>
    <item name="android:textSize">20sp</item>
    <item name="android:background">#4A86CE</item>
</style>
```

自定义好标题样式之后，改写之前布局文件中标题样式的代码，代码下所示，效果如图 7-2（b），这样定义好之后，在书写其他功能界面的布局时可以直接引用自定义好的标题样式即可。

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

7.2 设置中心界面

7.2.1 自动更新

观察如图 7-3 的完整项目中的设置中心界面，第一个功能是自动更新，即打开时应用会自动检测是否需要升级，并弹出升级对话框，关闭时就不会有升级提示。

图 7-3　设置界面的自动更新

可以看到位于最上方的背景图片与中间和最下面的图片都是不同的，这就需要我们设置不同的背景图片，这时我们可以这样书写代码实现简单的布局界面，如下所示：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2"
4.      android:layout_width="match_parent"
5.      android:layout_height="match_parent"
6.      android:orientation="vertical"
7.      >
8.      <TextView
9.          style="@style/TitleBarTextView"
10.         android:text="设置中心"
11.        />
12.     <RelativeLayout
13.         android:layout_width="match_parent"
14.         android:layout_height="wrap_content"
15.        <!-- first_selected 是最上面功能的背景选择器 -->
16.         android:background="@drawable/first_selected"
17.         android:layout_marginTop="10dp"
18.         android:paddingLeft="15dp"
19.         android:paddingRight="15dp"
20.         android:paddingTop="8dp"
21.         android:onClick="autoDate"
22.         android:clickable="true"
23.         android:paddingBottom="8dp"
24.         >
25.         <TextView
26.             android:layout_width="match_parent"
```

```
27.           android:layout_height="wrap_content"
28.           android:text="自动更新"
29.           />
30.       <ImageView
31.            android:layout_width="wrap_content"
32.            android:layout_height="wrap_content"
33.            android:src="@drawable/off"
34.           <!-- android:layout_alignParentRight="true"  在屏幕的右侧 -->
35.            android:layout_alignParentRight="true"
36.           />
37.    </RelativeLayout>
38.     <RelativeLayout
39.       android:layout_width="match_parent"
40.       android:layout_height="wrap_content"
41.       <!-- middle_selected 是中间功能的背景选择器 -->
42.       android:background="@drawable/middle_selected"
43.       android:paddingLeft="15dp"
44.       android:paddingRight="15dp"
45.       android:paddingTop="8dp"
46.       android:onClick="autoDate"
47.       android:clickable="true"
48.       android:paddingBottom="8dp"
49.       >
50.       <TextView
51.          android:layout_width="match_parent"
52.          android:layout_height="wrap_content"
53.          android:text="自动更新"
54.          />
55.       <ImageView
56.            android:layout_width="wrap_content"
57.            android:layout_height="wrap_content"
58.            android:src="@drawable/off"
59.            android:layout_alignParentRight="true"
60.          />
61.    </RelativeLayout>
62.     <RelativeLayout
63.       android:layout_width="match_parent"
64.       android:layout_height="wrap_content"
65.      <!-- last_selected 是最下面功能的背景选择器 -->
66.       android:background="@drawable/last_selected"
67.       android:paddingLeft="15dp"
68.       android:paddingRight="15dp"
69.       android:paddingTop="8dp"
```

```
70.         android:onClick="autoDate"
71.         android:clickable="true"
72.         android:paddingBottom="8dp"  >
73.         <TextView
74.            android:layout_width="match_parent"
75.            android:layout_height="wrap_content"
76.            android:text="自动更新"
77.            />
78.         <ImageView
79.             android:layout_width="wrap_content"
80.             android:layout_height="wrap_content"
81.             android:src="@drawable/off"
82.             android:layout_alignParentRight="true"
83.            />
84.     </RelativeLayout>
85.     </LinearLayout>
```

上面代码中的三个选择器是创建在 res 下的 drawable 目录（需要手动创建该文件夹）中的，注意选择器在创建应选择 selector 标签属性。具体各自的代码如文件 7-1，文件 7-2，文件 7-3 所示。

【文件 7-1】 res/drawable/first_selected.xml

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

【文件 7-2】 res/drawable/middle_selected.xml

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

【文件 7-3】 res/drawable/last_selected.xml

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

同时创建常用设置界面的 SettingActivity.java，并实现我们在布局文件中定义的 autoDate 点击事件，即：

```
1.  //仅仅这样书写即可
2.  public void autoDate(View view){
3.  }
```

这时，运行程序并在主界面跳转至常用设置界面，效果如图 7-4 所示。

图 7-5 设置界面

7.2.2 定义组合控件

为增加代码的复用性，这里考虑使用组合控件来定义设置中心的条目，这里将上述代码组合在一个条目布局文件中即 item_setting_view.xml。如文件 7-4 所示：

【文件 7-4】 res/layout/item_setting_view.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:background="@drawable/first_selected"
6.      android:layout_marginTop="10dp"
7.      android:paddingBottom="8dp"
8.      android:paddingLeft="15dp"
9.      android:paddingRight="15dp"
10.     android:paddingTop="8dp" >
11.     <TextView
12.        android:id="@+id/tv_title"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:text="自动更新" />
16.     <ImageView
17.        android:id="@+id/iv_toggle"
18.        android:layout_width="wrap_content"
19.        android:layout_height="wrap_content"
```

```
20.         android:layout_alignParentRight="true"
21.         android:src="@drawable/off" />
22. </RelativeLayout>
```

其次，在 com.itheima.mobilesafe_sh2.act.view 包下创建自定义 SettingItemView 继承 RelativeLayout，并复写其构造函数，初始化组合控件的 View，如下所示：

```
1.  public class SettingItemView extends RelativeLayout {
2.      public SettingItemView(Context context, AttributeSet attrs, int defStyle) {
3.          super(context, attrs, defStyle);
4.          // TODO Auto-generated constructor stub
5.      }
6.      public SettingItemView(Context context, AttributeSet attrs) {
7.          super(context, attrs);
8.          init(context, attrs);
9.      }
10.     /**
11.      * 初始化 view
12.      * @param context
13.      * @param attrs
14.      */
15.     private void init(Context context, AttributeSet attrs) {
16.         // this 表示把 xml 布局放置到 SettingItemView 身上
17.         View view = View.inflate(context, R.layout.item_setting_view, this);
18.     }
19.     public SettingItemView(Context context) {
20.         super(context);
21.         // TODO Auto-generated constructor stub
22.     }
```

7.2.3 自定义属性

自定好组合控件之后，之前的 activity_setting.xml 中的代码就可以进行简化，具体如下所示：

```
<?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
      style="@style/TitleBarTextView"
      android:text="设置中心"
     />
   <!-- android:layout_alignParentRight="true"  在屏幕的右侧 -->
```

```
   <com.itheima.mobilesafe_sh2.act.view.SettingItemView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       />
   <com.itheima.mobilesafe_sh2.act.view.SettingItemView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
     />
   <com.itheima.mobilesafe_sh2.act.view.SettingItemView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
      />
</LinearLayout>
```

但是，考虑到不同条目上的信息是不同的，例如该条目的标题和该条目处于哪个位置。这里考虑使用自定义属性来定义，在定义自定义属性之前，我们需要定义属于自己的属性空间，在上面的线性布局中，增加这样一条语句：**xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2",**语句的最后一句是该应用的包名。

接下来，在布局文件中书写属于我们自己定义的一些属性，即每个条目的标题和条目的位置，完整的 activity_setting.xml 如文件 7-5 所示：

【文件 7-5】 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.     <!-- 注意：该语句的后面是包名 -->
4.      xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2"
5.      android:layout_width="match_parent"
6.      android:layout_height="match_parent"
7.      android:orientation="vertical"
8.      >
9.      <TextView
10.         style="@style/TitleBarTextView"
11.         android:text="设置中心"
12.        />
13.     <!-- android:layout_alignParentRight="true"  在屏幕的右侧 -->
14.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
15.         android:layout_width="match_parent"
16.         android:layout_height="wrap_content"
17.         itheima:title="自动更新"
18.        />
19.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
20.         android:layout_width="match_parent"
21.         android:layout_height="wrap_content"
22.         itheima:title="黑马程序员"
23.        />
```

```
    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        itheima:title="哈哈嘎嘎"
        />
</LinearLayout>
```

这时，需要在 values 目录下创建一个属性文件即 attrs.xml，存放我们之前定义的属性，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--
     SettingItemView 属性的名字
      attr 属性的字段
      format 属性的类型
    -->
    <declare-styleable name="SettingItemView">
     <!-- 此处 string 前面的字母 s 不能大写，否则会报错 -->
        <attr name="title" format="string" />
    </declare-styleable>
</resources>
```

定义好属性之后必须要手动读出来再设置，这时需要在自定义的 SettingItemView 中的 init()方法进行改写，如下所示：

```
    /**
     * 初始化 view
     * @param context
     * @param attrs
     */
    private void init(Context context, AttributeSet attrs) {
        // this 表示把 xml 布局放置到 SettingItemView 身上
        View view = View.inflate(context, R.layout.item_setting_view, this);
        // 标题
        mTvTitle = (TextView) findViewById(R.id.tv_title);
        // 滑动开关
        mIvToggle = (ImageView) findViewById(R.id.iv_toggle);
        // 读取刚刚我们定义的属性
        // 第一个参数：属性
        // 第二个参数：属性的数组
        TypedArray ta = context.obtainStyledAttributes(attrs,
                R.styleable.SettingItemView);
        String title = ta.getString(R.styleable.SettingItemView_title);
        ta.recycle();
        mTvTitle.setText(title);
    }
```

当这样设置之后，运行程序如图 7-6 所示，之所以背景都是这样，是因为 item_setting_view 中的背景图片我们使用的都是第一个选择器。

图 7-6 自定义属性后的设置界面

为了修正不同条目对应的背景图片，我们应该使用一个属性来标识当前条目所处的位置，这样我们在读取当前所设属性时就可以根据位置进行对应背景图片的更换。因此，修改 activity_setting.xml 文件，增加一条 itbackground 属性标识当前条目所处的位置，这时完整的 activity_setting.xml 如文件 7-6 所示：

【文件 7-6】 res/layout/setup_password_dialog.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.      >
8.      <TextView
9.          style="@style/TitleBarTextView"
10.         android:text="设置中心"
11.        />
12.     <!-- android:layout_alignParentRight="true"  在屏幕的右侧 -->
13.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
14.         android:id="@+id/sv_auto_date"
15.         android:layout_width="match_parent"
16.         android:layout_height="wrap_content"
17.         itheima:title="自动更新"
18.         android:layout_marginTop="10dp"
19.         itheima:itbackground="first" />
20.
21.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
22.         android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        itheima:title="黑马程序员"
        itheima:itbackground="middle"/>

    <com.itheima.mobilesafe_sh2.act.view.SettingItemView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        itheima:title="哈哈嘎嘎"
        itheima:itbackground="last" />
</LinearLayout>
```

同时，属性文件 attrs.xml 中的 SettingItemView 属性下需要增加 itbackground 属性，并且该属性只能取 first、middle 和 last 值中的一个，这时可以使用枚举来定义，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--
     SettingItemView 属性的名字
      attr 属性的字段
      format 属性的类型
    -->
    <declare-styleable name="SettingItemView">
        <attr name="title" format="string" />
        <attr name="itbackground">
            <!-- 使用枚举-->
            <enum name="first" value="0" />
            <enum name="middle" value="1" />
            <enum name="last" value="2" />
        </attr>
    </declare-styleable>
</resources>
```

定义好属性之后，我们需要在代码中读取属性才能进行设置，这时需要修改 SettingItemView.java 中的代码，如下所示：

```
    private final static int FIRST = 0;
     private final static int MIDDLE = 1;
     private final static int LAST = 2;
    /**
      * 初始化 view
      * @param context
      * @param attrs
      */
     private void init(Context context, AttributeSet attrs) {
         // this 表示把 xml 布局放置到 SettingItemView 身上
         View view = View.inflate(context, R.layout.item_setting_view, this);
         // 标题
         mTvTitle = (TextView) findViewById(R.id.tv_title);
```

```
14.         // 滑动开关
15.         mIvToggle = (ImageView) findViewById(R.id.iv_toggle);
16.         // 读取刚刚我们定义的属性
17.         // 第一个参数：属性
18.         // 第二个参数：属性的数组
19.         TypedArray ta = context.obtainStyledAttributes(attrs,
20.                 R.styleable.SettingItemView);
21.         String title = ta.getString(R.styleable.SettingItemView_title);
22.         //获取 itbackground 属性的值
23.         int itbackground = ta.getInt(R.styleable.SettingItemView_itbackground,
24.                 0);
25.         ta.recycle();
26.         mTvTitle.setText(title);
27.         //根据 itbackground 的值设置不同的背景图片
28.         switch (itbackground) {
29.         case FIRST:
30.             view.setBackgroundResource(R.drawable.first_selected);
31.             break;
32.         case MIDDLE:
33.             view.setBackgroundResource(R.drawable.middle_selected);
34.             break;
35.         case LAST:
36.             view.setBackgroundResource(R.drawable.last_selected);
37.             break;
38.         }
39.     }
```

设置好 title 和 itbackground 属性之后，运行程序如图 7-7 所示：

图 7-7 设置中心界面

7.2.4 封装土司工具类

土司弹窗在项目会经常用到，为促进代码的简洁性，这里考虑封装一个吐司工具类。为了能够提供一个安全的吐司，这里考虑了主线程和子线程的情况，如文件【7-7】所示。

【文件 7-7】 com.itheima.mobilesafe_sh2.utils/ToastUtils.java

```
1.  public class ToastUtils {
2.      /**
3.       * 展示一个安全的土司
4.       * @param activity
5.       * @param msg
6.       */
7.      public static void showSafeToast(final Activity activity,final String msg){
8.          //当在主线程时，土司直接弹出
9.          if(Thread.currentThread().getName().equals("main")){
10.             Toast.makeText(activity, msg, 0).show();
11.         }else{
12.         //在子线程时，让土司在主线程中弹出
13.             activity.runOnUiThread(new Runnable() {
14.                 @Override
15.                 public void run() {
16.                     // TODO Auto-generated method stub
17.                     Toast.makeText(activity, msg, 0).show();
18.                 }
19.             });
```

```
        }
    }
}
```

7.2.5 滑动开关

此处需要实现设置界面中滑动开关状态的切换，即在 SettingActivity.java 中实现 SettingItemView 的点击事件，如果是打开的，那么点击就变成关闭；如果是关闭的，点击之后就会变成打开。代码如下所示：

```
//实现自动更新的点击事件
protected void onCreate(Bundle savedInstanceState) {
      // TODO Auto-generated method stub
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_setting);
      //自动更新
      mAutoUpDate = (SettingItemView) findViewById(R.id.sv_auto_date);
      mAutoUpDate.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
              //如果是打开的。那么点击就变成关闭。如果是关闭点击之后就变成打开
              //获取到滑动开关返回的值
              if(mAutoUpDate.isToggle()){
                  mAutoUpDate.setToggle(false);
              }else{
                  mAutoUpDate.setToggle(true);
              }
          }
      });
  }
```

在 SettingItemView 中实现 isToggle（）和 setToggle（）这两个方法，前者用于判断当前滑动开关的状态，后者用于设置当前的滑动开关的状态来改变滑动开关的背景图片，并且需要在初始化 init()中设置默认的开关状态，具体的代码如下所示。

```
//初始化滑动开关
 private boolean isToggle = false;
//初始化时设置默认的开关状态
private void init(Context context, AttributeSet attrs) {
    ......
    //设置默认为 false
    setToggle(isToggle);
 }
/**
  * 返回滑动开关是打开还是关闭
  * @return
  */
```

```
13.     public boolean isToggle() {
14.         return isToggle;
15.     }
16.     /**
17.      * 设置滑动开关的值，同时改变滑动开关的背景图片
18.      * @param b
19.      */
20.     public void setToggle(boolean b) {
21.         this.isToggle = b;
22.         if(isToggle){
23.             mIvToggle.setImageResource(R.drawable.on);
24.         }else{
25.             mIvToggle.setImageResource(R.drawable.off);
26.         }
27.     }
```

这样设置之后，就实现了点击自动更新条目时改变滑动开关的状态，如图 7-8 所示。

图 7-8　自动更新开关状态的切换

7.2.6 自动更新的回显

为了实现自动更新状态的保存，这里我们使用 SharedPreferences 来保存用户的设置状态，即保存自动更新的滑动开关的状态。代码如下所示。

```
1.    @Override
2.      protected void onCreate(Bundle savedInstanceState) {
3.          // TODO Auto-generated method stub
4.          super.onCreate(savedInstanceState);
```

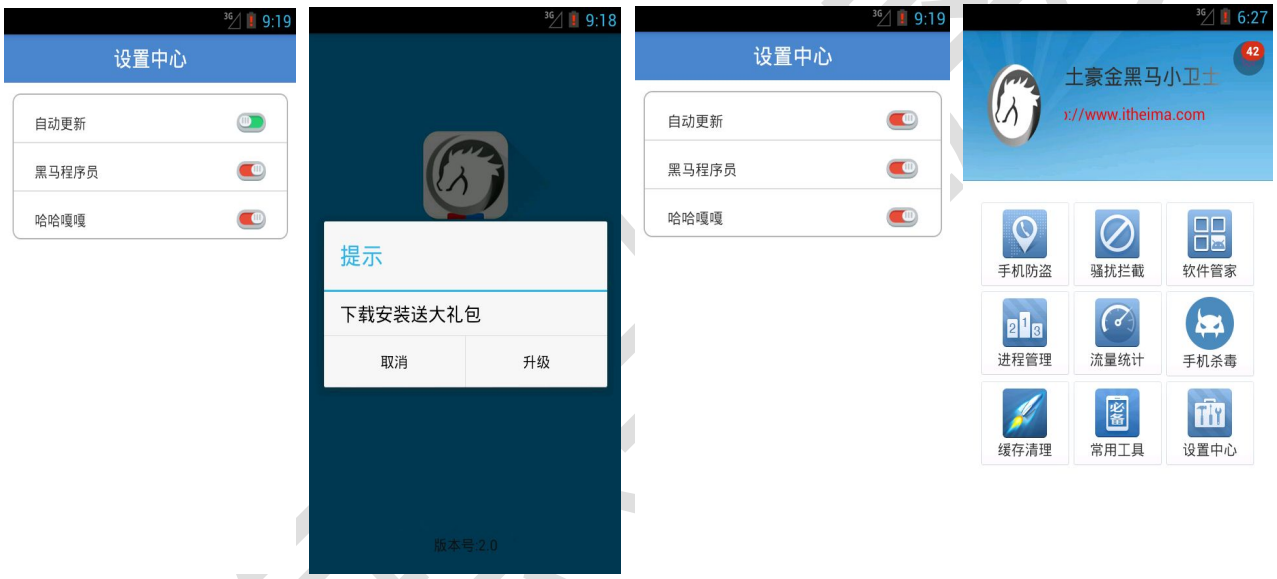
```
        setContentView(R.layout.activity_setting);
        // 第一个参数名字
        // 第二个参数模式
        sp = getSharedPreferences("config", 0);
        boolean result = sp.getBoolean("isUpdate", false);
         // 判断是否已经缓冲了自动更新
         // 保存自动更新的状态信息的是用于在主界面进入设置界面时读取之前设置的状态信息，进而改变设
         // 置界面中自动更新的滑动开关的背景图片，这就需要在设置界面加载时读取 sp 中的信息
        if(result){
            mAutoUpDate.setToggle(true);
        }else{
            mAutoUpDate.setToggle(false);
        }
        // 自动更新
        mAutoUpDate = (SettingItemView) findViewById(R.id.sv_auto_date);
        mAutoUpDate.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
        //      ToastUtils.showSafeToast(SettingActivity.this, "哈哈");
                // 如果是打开的。那么点击就变成关闭。如果是关闭点击之后就变成打开
                // 获取到滑动开关返回的值
                if(mAutoUpDate.isToggle()){
                    mAutoUpDate.setToggle(false);
                }else{
                    mAutoUpDate.setToggle(true);
                }
                Editor edit = sp.edit();
                //保存用户的自动更新状态，isUpdate 中保存的便是状态信息
                edit.putBoolean("isUpdate", mAutoUpDate.isToggle());
                edit.commit();
            }
        });
    }
```

保存自动更新的状态信息是也是用于在闪屏页面中，判断应用的自动更新设置开关是否开启，如果开启就需要检查应用版本决定是否升级；如果没有打开开关就直接进入主界面，这里的代码需要在 SplashActivity.java 的 initView()方法中实现判断逻辑，如下所示：

```
    private void initView() {
        setContentView(R.layout.activity_splash);
        mTvVersion = (TextView) findViewById(R.id.tv_splash_version);
        mTvVersion.setText("版本号:" + PackageUtils.getVersionName(this));
        // 获取到本地的版本号
        mLoclVersionCode = PackageUtils.getVersionCode(this);
        //获取存储在 config 文件中 isUpdate 的值
```

```
8.
9.          SharedPreferences sp = getSharedPreferences("config", 0);
10.         boolean result = sp.getBoolean("isUpdate", false);
11.         // 如果是 ture 说明打开了滑动开关
12.         if (result) {
13.             // 检查版本号
14.             checkViersion();
15.         }else{
16.             loadMainUI();
17.         }
18.     }
```

上面的代码完成在闪屏页面读取自动更新状态的功能，即在闪屏页面是否需要检查应用版本，如图 7-9 中的（a）-（d）所示，其中（a）（b）为开启自动更新时闪屏页面弹出应用升级对话框，（c）（d）为关闭自动更新时，闪屏页面直接进入主界面。



（a） （b） （c） （d）

图 7-9 设置自动更新开关状态时的版本更新

7.2.7 封装 SharedPreferences 工具类

在整个项目的开发过程中，需要使用 SharedPreferences 保存信息的时候很多，因此，为了代码更加简洁，这里考虑使用工具类封装 SharedPreferences，根据存储数据的类型进行分类。具体如文件【7-8】所示。

【文件 7-8】 com.itheima.mobilesafe_sh2.utils/SharedPreferencesUtils.java

```
1.  public class SharedPreferencesUtils {
2.      // sp 的名字
3.      public final static String SP_NAME = "config";
4.      private static SharedPreferences sp;
5.      // 保存 boolean 类型的数据
6.      public static void saveBoolean(Context context, String key, boolean value) {
7.          if (sp == null)
```

```
            sp = context.getSharedPreferences(SP_NAME, 0);
        sp.edit().putBoolean(key, value).commit();
    }
    // 保存 string 类型的数据
    public static void saveString(Context context, String key, String value) {
        if (sp == null)
            sp = context.getSharedPreferences(SP_NAME, 0);
        sp.edit().putString(key, value).commit();
    }
    // 获取 string 类型的数据
    public static String getString(Context context, String key, String defValue) {
        if (sp == null)
            sp = context.getSharedPreferences(SP_NAME, 0);
        return sp.getString(key, defValue);
    }
    // 获取 boolean 类型的数据
    public static boolean getBoolean(Context context, String key,
            boolean defValue) {
        if (sp == null)
            sp = context.getSharedPreferences(SP_NAME, 0);
        return sp.getBoolean(key, defValue);
    }
}
```

封装好 sp 的工具类之后，这样就可以修改原代码了，获取 sp 中的状态信息是这样修改，如下所示。

```
//      SharedPreferences sp = getSharedPreferences("config", 0);
//      boolean result = sp.getBoolean("isUpdate", false);
//      将上面的代码可简化为下面一行
        boolean   result   =   SharedPreferencesUtils.getBoolean(SplashActivity.this,
"isUpdate", false);
```

其次，向 sp 中存储数据可以这样进行修改，如下所示。

```
//      sp = getSharedPreferences("config", 0);
//      Editor edit = sp.edit();
//      edit.putBoolean("isUpdate", mAutoUpDate.isToggle());
//      edit.commit();
//      上面的四行代码可简化为下面一行代码
        SharedPreferencesUtils.saveBoolean(SettingActivity.this,"isUpdate",
mAutoUpDate.isToggle());
```

7.2.8 定义常量类

为了代码更加规范，这里定义一个常量类来管理项目中的所有的常量。为方便起见，这里列出了本章所需要的所有常量，如文件【7-9】所示。

【文件 7-9】 com.itheima.mobilesafe_sh2.utils/Constants.java

```
1.      public interface Constants {
2.       /**
3.        * 升级的 url
4.        */
5.       final static String URL = "http://192.168.56.1/info.json";
6.       /**
7.        * 自动更新
8.        */
9.       final static String IS_UPDATE = "isUpdate";
10.     /**
11.      * 手机防盗密码
12.      */
13.     final static String SJFD_PWD = "sjfd_pwd";
14.     /**
15.      * 手机防盗 sim 卡
16.      */
17.     static final String SJFD_SIM = "sjfd_sim";
18. }
```

7.2.9 自定义对话框

上面我们所定义的吐司是蓝色的背景，为了实现更好的美观效果，这里需要提供不同的样式风格给用户，如图 7-10 所示，在弹出的对话框中为用户提供了多种样式的选择，

图 7-10 归属地样式的选择

为了这样的对话框样式效果，我们需要进行自定义对话框，在 com.itheima.mobilesafe_sh2.act.view 包

创建 AddressDialog.java 文件，在该文件中负责对话框样式的设计。

首先，创建一个没有标题的对话框，即如图 4-18 样式的对话框，然后使用数据适配器进行数据填充。

对话框 AddressDialog 的代码如下所示，

```
1.  public class AddressDialog extends Dialog {
2.      private Window window;
3.      private OnItemClickListener mOnItemClickListener;
4.      //把我们自己的样式传给父类
5.      //屏蔽父类里面的样式
6.      public AddressDialog(Context context) {
7.          super(context, R.style.AddressDialogStyle);
8.      }
9.      @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11.         // TODO Auto-generated method stub
12.         super.onCreate(savedInstanceState);
13.         setContentView(R.layout.dialog_address_style);
14.         window = getWindow();
15.         //获取到当前窗体的样式
16.         LayoutParams params = window.getAttributes();
17.
18.         params.gravity = Gravity.BOTTOM;
19. //              | Gravity.CENTER_HORIZONTAL;
20.         //设置窗体的属性
21.         window.setAttributes(params);
22.     }
```

其中，AddressDialogStyle 定义了对话框的样式，将我们的样式传给父类以屏蔽父类中的样式，该样式存储在 res/values/styles.xml 中，对应的样式内容如下所示。

```
1.      <!--
2.         自定义对话框的样式
3.         parent="@android:style/Theme.Dialog" 告诉 android 系统我是一个对话框
4.      -->
5.      <style name="AddressDialogStyle" parent="@android:style/Theme.Dialog">
6.          <!-- 背景默认是透明色  背景设置为白色 -->
7.          <item name="android:windowBackground">@android:color/white</item>
8.          <item name="android:windowFrame">@null</item>
9.          <item name="android:windowContentOverlay">@null</item>
10.         <item name="android:windowAnimationStyle">@style/AddressAnimationDialog</item>
11.         <!-- 设置背景为黑色 -->
12.         <item name="android:backgroundDimEnabled">true</item>
13.         <!-- 默认是透明色 -->
14.         <!-- <item name="android:windowIsTranslucent">true</item> -->
15.         <!-- 没有标题头 -->
16.         <item name="android:windowNoTitle">true</item>
```

```
17.         <!-- 点击对话框的后面，对话框消失 -->
18.         <item name="android:windowCloseOnTouchOutside">true</item>
19.     </style>
```

另外，对话框加载的布局文件 dialog_address_style.xml 中是一个 TextView 和 ListView，具体的代码如文件【7-10】所示。

【文件 7-10】 res/layout/dialog_address_style..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="wrap_content"
5.      android:orientation="vertical" >
6.
7.      <TextView
8.          style="@style/TitleBarTextView"
9.          android:text="选择归属地样式" />
10.     <!--
11.       android:divider="#66000000" 设置 listview 的下划线
12.       android:dividerHeight="0.5dp" 设置下划线的高度
13.       android:listSelector="@drawable/bt_selected"设置 listview item 的背景颜色
14.       android:scrollbars="none" 去掉 listview 的滚动条
15.     -->
16.
17.     <ListView
18.         android:id="@+id/list_view"
19.         android:layout_width="match_parent"
20.         android:layout_height="200dp"
21.         android:divider="#66000000"
22.         android:dividerHeight="0.5dp"
23.         android:scrollbars="none"
24.         android:listSelector="@drawable/bt_selected" >
25.     </ListView>
26. </LinearLayout>
```

在常用设置界面，我们加入了归属地显示风格这一功能，对应的布局文件也有了一定的改变，如文件【7-11】所示。

【文件 7-11】 res/layout/activity_setting.xml

```
1.    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:itheima="http://schemas.android.com/apk/res/com.itheima.mobilesafe_sh2"
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="设置中心" />
9.      <!-- android:layout_alignParentRight="true"  在屏幕的右侧 -->
```

```
10.
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.
19.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
20.         android:id="@+id/siv_address"
21.         android:layout_width="match_parent"
22.         android:layout_height="wrap_content"
23.         itheima:itbackground="middle"
24.         itheima:title="归属地显示设置" />
25.
26.     <com.itheima.mobilesafe_sh2.act.view.SettingItemView
27.         android:id="@+id/siv_address_style"
28.         android:layout_width="match_parent"
29.         android:layout_height="wrap_content"
30.         itheima:isToggle="true"
31.         itheima:itbackground="last"
32.         itheima:title="归属地设置风格" />
33. </LinearLayout>
```

设置界面 SettingActivity 中，归属地显示风格选项的点击是这样实现的，如下所示。

```
1.  case R.id.siv_address_style:
2.              //初始化对话框
3.              final AddressDialog dialog = new AddressDialog(this);
4.              dialog.show();
5.              break;
```

上面的代码设置好之后，我们简单运行程序，效果如图 7-11 所示。

图 7-11　自定义对话框

为了实现上图 7-11 的效果，我们需要填充自定义对话框中 ListView 的数据，首先我们在 SettingActivity 定义一个数据适配器，用于填充数据，对应的代码如下所示。

```
1.      //填充的数据
2.      private String[] titles = new String[] { "半透明", "活力橙", "卫士蓝", "金属灰", "苹果
3.  绿" };
4.      private int[] icons = new int[] { R.drawable.call_locate_white,
5.              R.drawable.call_locate_orange, R.drawable.call_locate_blue,
6.              R.drawable.call_locate_gray, R.drawable.call_locate_green };
7.      //数据适配器
8.      private class AddressAdapter extends BaseAdapter{
9.          @Override
10.         public int getCount() {
11.             // TODO Auto-generated method stub
12.             return titles.length;
13.         }
14.         @Override
15.         public Object getItem(int position) {
16.             // TODO Auto-generated method stub
17.             return null;
18.         }
19.         @Override
20.         public long getItemId(int position) {
21.             // TODO Auto-generated method stub
22.             return 0;
23.         }
24.         @Override
25.         public View getView(int position, View convertView, ViewGroup parent) {
26.             View view = View.inflate(SettingActivity.this,
27. R.layout.item_setting_address_dialog, null);
```

```
28.             //图片的颜色
29.             ImageView iv_icon_color = (ImageView) view.findViewById(R.id.iv_icon_color);
30.             //颜色的名字
31.             TextView tv_color_name = (TextView) view.findViewById(R.id.tv_color_name);
32.             //颜色的状态
33.             ImageView iv_color_state = (ImageView) view.findViewById
34. (R.id.iv_color_state);
35.             //设置 icon 的图片
36.             iv_icon_color.setImageResource(icons[position]);
37.             //设置颜色的名字
38.             tv_color_name.setText(titles[position]);
39.             //获取当前的颜色
40.             int style = SharedPreferencesUtils.getInt(SettingActivity.this,
41. Constants.ADDRESS_COLOR, 0);
42.             System.out.println("style---" + style);
43.             //判断当前选中的颜色，设置右边箭头的显示或者隐藏
44.             if(style == position){
45.                 iv_color_state.setVisibility(View.VISIBLE);
46.             }else{
47.                 iv_color_state.setVisibility(View.INVISIBLE);
48.             }
49.             return view;
50.         }
51.     }
```

其中，item_setting_address_dialog.xml 是用于显示 ListView 每个条目的布局，用于展示不同归属地风格的图片，具体的代码如文件【7-12】所示。

【文件 7-12】 res/layout/item_setting_address_dialog.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:layout_margin="10dp"
6.      android:gravity="center_vertical" >
7.      <ImageView
8.          android:id="@+id/iv_icon_color"
9.          android:layout_width="wrap_content"
10.         android:layout_height="wrap_content"
11.         android:src="@drawable/call_locate_blue" />
12.     <TextView
13.         android:id="@+id/tv_color_name"
14.         android:layout_width="wrap_content"
15.         android:layout_height="wrap_content"
16.         android:layout_centerInParent="true"
```

```
17.         android:textSize="18sp"
18.         android:text="半透明" />
19.     <ImageView
20.         android:id="@+id/iv_color_state"
21.         android:layout_width="wrap_content"
22.         android:layout_height="wrap_content"
23.         android:layout_alignParentRight="true"
24.         android:layout_centerVertical="true"
25.         android:layout_margin="10dp"
26.         android:src="@drawable/ic_selected" />
27. </RelativeLayout>
```

数据适配器设置好之后，接下来开始在对话框的 ListView 上 Adapter 填充数据，由于 ListView 是在 AddressDialog.java 中，而数据适配器是在 SettingActivity.java，二者不在同一个文件中，因此，我们需要在 SettingActivity 中将数据适配器对象传给 AddressDialog，对应的代码如下所示。

首先，SettingActivity 中的代码这样修改，使用 SharedPreferences 保存当前选择的风格颜色。

```
1.  case R.id.siv_address_style:
2.              //初始化对话框
3.              final AddressDialog dialog = new AddressDialog(this);
4.              //初始化适配器
5.              final AddressAdapter adapter = new AddressAdapter();
6.              //设置适配器
7.              dialog.setAdapter(adapter);
8.              //设置点击事件
9.              dialog.setOnItemClickListener(new OnItemClickListener() {
10.                 @Override
11.                 public void onItemClick(AdapterView<?> parent, View view,
12.                         int position, long id) {
13.                     //保存号码归属地的颜色
14.                     SharedPreferencesUtils.saveInt(SettingActivity.this,
15. Constants.ADDRESS_COLOR, position);
16.                     //对话框消失
17.                     dialog.dismiss();
18.                     //刷新当前页面
19.                     adapter.notifyDataSetChanged();
20.                 }
21.             });
22.             dialog.show();
23.             break;
```

其中，Constants.ADDRESS_COLOR 是常量接口中用于保存归属地显示颜色的常量。

最后，对应的 AddressDialog.java 也要进行修改，具体的代码如下所示。

```
1.  public class AddressDialog extends Dialog {
2.      private Window window;
3.      private BaseAdapter mAdapter;
4.       private ListView mListView;
```

```
5.      private OnItemClickListener mOnItemClickListener;
6.      //把我们自己的样式传给父类
7.      //屏蔽父类里面的样式
8.      public AddressDialog(Context context) {
9.          super(context, R.style.AddressDialogStyle);
10.     }
11.     @Override
12.     protected void onCreate(Bundle savedInstanceState) {
13.         // TODO Auto-generated method stub
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.dialog_address_style);
16.         mListView = (ListView) findViewById(R.id.list_view);
17.         setAdapter(mAdapter);
18.         setOnItemClickListener(mOnItemClickListener);
19.         window = getWindow();
20.         //获取到当前窗体的样式
21.         LayoutParams params = window.getAttributes();
22.         params.gravity = Gravity.BOTTOM;
23. //              | Gravity.CENTER_HORIZONTAL;
24.         //设置窗体的属性
25.         window.setAttributes(params);
26.     }
27.     /**
28.      * listview 设置适配器
29.      * @param adapter
30.      */
31.     public void setAdapter(BaseAdapter adapter) {
32.         this.mAdapter = adapter;
33.         if(null != mListView){
34.             mListView.setAdapter(adapter);
35.         }
36.     }
37.     /**
38.      * listView 添加点击事件
39.      * @param mOnItemClickListener
40.      */
41.     public void setOnItemClickListener(OnItemClickListener mOnItemClickListener){
42.         this.mOnItemClickListener = mOnItemClickListener;
43.         if(null != mListView){
44.             mListView.setOnItemClickListener(mOnItemClickListener);
45.         }
46.     }
47. }
```

最后，代码书写好之后，运行程序，效果如图 7-12 所示。

图 7-12　归属地样式

7.3 本章小结

本章主要是针对常用设置模块进行讲解，首先针对该模块的程序锁和看门狗介绍。该模块的代码量较大，实现的功能较多也比较实用，编程者需要动手将这些功能全部实现，加强代码编写能力，如遇到难以理解的逻辑或功能，可以先将程序打断点观察程序执行逻辑。