第5章 其他常用编程技术

本节汇集了 Android 应用开发中经常涉及的一些相关编程技术。

5.1 Intent

当从一个 Activity 调用另外的 Activity 时，需要使用 Intent。当使用 Service、Broadcast 或第三方 App 的功能时也需要使用 Intent。Intent 还负责调用或返回 Activity 时的数据传递。

5.1.1 Intent 的显式调用和隐式调用

Intent 的调用分为显示调用和隐式调用。

(1) 显式调用：明确 Intent 要调用的 Activity 名称(AndroidManifest.xml 中 activity 标签的 android:name 或者 Java 中定义的 Activity 类名称)。

(2) 隐式调用：不明确指定启动的 Activity，通过设置 Action、Data、Category，由系统根据 intent-filter 来筛选合适的 Activity。隐式调用的功能非常强大，可以调用系统的应用，如桌面、相机、浏览器等。

本案例设计了两个布局文件，分别取名 main.xml 和 second.xml。关键文件的代码如下：

【AndroidManifest.xml】

```
01  <?xml version = "1.0" encoding = "UTF - 8"?>
02  < manifest xmlns:android = "http://schemas.android.com/apk/res/android"
03       package = "com.xiaj">
04
05       < application
06            android:icon = "@drawable/icon"
07            android:label = "@string/app_name"
08            android:theme = "@android:style/Theme.Holo.Light">
09            < activity
10                 android:name = "com.xiaj.FirstActivity"
11                 android:label = "@string/app_name">
12                 < intent - filter >
13                        < action android:name = "android.intent.action.MAIN" />
14                        < category android:name = "android.intent.category.LAUNCHER" />
15                 </intent - filter >
16            </activity >
17            < activity
18                 android:name = "com.xiaj.SecondActivity"
19                 android:label = "@string/app_name">
```

```
20                <intent-filter>
21                    <action android:name="com.xiaj.SecondActivityAction" />
22                    <category android:name="android.intent.category.DEFAULT" />
23                </intent-filter>
24            </activity>
25
26        </application>
27    </manifest>
```

新添加 Activity 时 Android Studio 会自动在 AndroidManifest.xml 文件中添加 activity 标签进行注册。如果调用未注册的 Activity 将导致运行出错。第一个添加的 Activity 会自动添加第 12～15 行的 intent-filter。其中，第 13 行定义当前 Activity 对应的 action，其值 android.intent.action.MAIN 代表当前 Activity 是 App 的首选运行 Activity。第 14 行的 android.intent.category.LAUNCHER 决定应用程序是否显示在程序列表中。第 13 行和第 14 行合在一起决定 App 启动后会优先启动的 Activity。如果多个 Activity 中都有第 12～15 行的代码，则按 Activity 在 AndroidManifest.xml 中出现的顺序决定首先启动的 Activity。第 20～23 行的 intent-filter 标签是为了讲解 Intent 的隐式调用。

【FirstActivity.java】

```
01    public class FirstActivity extends Activity
02    {
03     @Override
04     public void onCreate(Bundle savedInstanceState)
05     {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.main);
08          Button button1 = (Button) findViewById(R.id.buttonFirst); //获取 id
09          button1.setOnClickListener(new View.OnClickListener()
10          {
11               @Override
12               public void onClick(View v)
13               {
14                    Intent intent = new Intent(); //Intent 对象
15                    //方法一
16                    //intent.setClass(FirstActivity.this, SecondActivity.class);
17                    //intent.setClass(getApplicationContext(), SecondActivity.class);
18
19                    //方法二
20                    //intent.setClassName(getApplicationContext(), "com.xiaj.SecondActivity");
21                    //intent.setClassName("com.xiaj", "com.xiaj.SecondActivity");
22
23                    //方法三
24                    //ComponentName componentName = new ComponentName(FirstActivity.this,
                      //"com.xiaj.SecondActivity");
25                    //intent.setComponent(componentName);
26
```

```
27                    //方法四
28                    //intent.setAction("com.xiaj.SecondActivityAction");
29
30                    //方法五
31                    //intent = new Intent("com.xiaj.SecondActivityAction");
32
33                    //方法六
34                    //intent.setData(Uri.parse("https://www.qq.com"));
35
36                    startActivity(intent); //启动
37                }
38            });
39     }
40  }
```

代码中前 3 种方法属于显式调用,后 3 种方法属于隐式调用。其中,方法四和方法五必须要在 AndroidManifest.xml 中有 intent-filter 标签,其中的 action 和 category 标签也是不可缺少的,否则调用时也会出错。方法六调用后直接使用默认浏览器访问指定网站,这为程序的功能扩展提供了极大的便利。

【注】 当同时使用显式调用和隐式调用时,优先选择显式调用。

【SecondActivity.java】

```
01  public class SecondActivity extends Activity
02  {
03      @Override
04      protected void onCreate(Bundle savedInstanceState)
05      {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.second);
08
09          TextView textView1 = (TextView) findViewById(R.id.textView1);
10          Intent intent = getIntent();        //获取当前 Intent 对象
11          ComponentName componentName = intent.getComponent();          //获取组件名称
12
13          textView1.setText("PackageName: " + componentName.getPackageName());
14          textView1.append("\nClassNam: " + componentName.getClassName());
15          textView1.append("\nShortClassName: " + componentName.getShortClassName());
16          textView1.append("\nAction: " + intent.getAction());
17      }
18  }
```

当程序转到 SecondActivity 时,可用上述代码获取 Intent 中相关信息。其中第 16 行的 getAction()方法需要 FirstActivity 中的方法四和方法五才会显示非 null 值。SecondActivity 运行结果如图 5-1 所示。

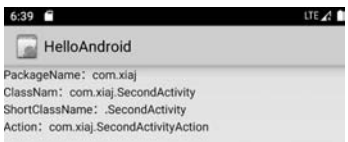

图 5-1 SecondActivity 运行结果

5.1.2 Intent 传值和取值


视频讲解

本案例演示从 FirstActivity 调用并传送数值到 SecondActivity，在 SecondActivity 运行后将新的数值传回 FirstActivity。本案例布局文件与 5.1.1 节案例类似，重点看 Java 文件。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      TextView textViewShow;
04      @Override
05      public void onCreate(Bundle savedInstanceState)
06      {
07          super.onCreate(savedInstanceState);
08          setContentView(R.layout.main);
09
10          EditText editTextUserName = (EditText) findViewById(R.id.editTextUserName);
11          EditText editTextPassword = (EditText) findViewById(R.id.editTextPassword);
12          Button button1 = (Button) findViewById(R.id.button1);
13          textViewShow = (TextView) findViewById(R.id.textViewshow);
14
15          button1.setOnClickListener(new View.OnClickListener()
16          {
17              @Override
18              public void onClick(View v)
19              {
20                  Intent intent = new Intent(getApplicationContext(), SecondActivity.
                    class);
21                  //方法一：传送参数
22                  intent.putExtra("name", editTextUserName.getText().toString());
23                  intent.putExtra("password", editTextPassword.getText().toString());
24
25                  //方法二：传送对象
26                  Bundle bundle1 = new Bundle();
27                  bundle1.putString("name", "张三丰");
28                  bundle1.putInt("age", 80);
29                  bundle1.putStringArray("徒弟们", new String[]{"宋远桥", "俞莲舟"});
30                  intent.putExtras(bundle1);
31                  //当无须返回数据时使用
32                  //startActivity(intent);
33
34                  //需要返回数据时使用
35                  startActivityForResult(intent, 10); //启动 SecondActivity,查看相
                                                        //关组件信息
36              }
37          });
38      }
39
```

```
40        @Override
41        protected void onActivityResult(int requestCode, int resultCode, Intent data)
42        {
43             super.onActivityResult(requestCode, resultCode, data);
44             if (resultCode == 20)         //20 代表 SecondActivity 发回的数据
45             {
46                  textViewShow.append("\nrequestCode:" + requestCode + "\nresultCode:" +
                    resultCode + "\nSecondActivity 传回的密码为:" + data.getStringExtra
                    ("password") + "\nintent:" + data.getExtras());
47             }
48        }
49   }
```

第 20 行使用 Intent 的构造方法显式调用 SecondActivity。

第 22～23 行调用 putExtra()方法按 key-value 键值对方式设定用户名和密码并绑定到变量 intent。比较容易犯错的地方是从 EditText 使用 getText()方法获取的 Editable 对象一定要使用 toString()方法转换为 String 类型,否则接收端按 String 型获取传送值时只能得到 null。

第 26 行演示使用 Bundle 方式传值。与 putExtra()方法传值相比,Bundle 可以传送非 String 类型的对象或数组。

第 27 行使用 putString()方法绑定 String 型键值对,此处 key 与第 22 行的 key 都是 name,字符串“张三丰”将覆盖第 22 行的 value 值“张三”。

第 28 行使用 putInt()方法绑定 int 型数值,也可以使用 putString()方法传送 String 型数值,接收方再将 String 型转换为 int 型。

第 29 行使用 putStringArray()方法将 String 型数组绑定到 bundle1。键值对中的 key 也可以使用中文,如本行中的“徒弟们”。

第 30 行将 bundle1 作为对象绑定到 intent。

第 32 行使用 startActivity()方法启动 intent(如果去掉注释的话),此时系统会运行 SecondActivity,并将相关数据传送给 SecondActivity。此时可以将数据从 FirstActivity 传递到 SecondActivity,但无法将数据回传给 FirstActivity。

如果要实现数据从 SecondActivity 回传给 FirstActivity,需将第 32 行的 startActivity()方法替换成第 35 行的 startActivityForResult()方法。startActivityForResult()方法除了具有 startActivity()的功能以外,还具有等待被调用的 SecondActivity()返回 intent 的功能。startActivityForResult()方法多了一个参数 requestCode。如果 FirstActivity 中发起多个调用 SecondActivity 的 intent,requestCode 用于区分是谁发起的调用 SecondActivity。

当 FirstActivity 使用 startActivityForResult()方法转到 SecondActivity,且 SecondActivity 使用 setResult()方法返回结果时,会自动调用第 41 行的 onActivityResult()方法。方法中第一个参数 requestCode 对应 startActivityForResult()方法中的 requestCode,resultCode 对应 SecondActivity 中 setResult()方法的参数 resultCode。如此就知道是谁发起了请求,是谁给了回应。

【SecondActivity.java】

```
01  public class SecondActivity extends Activity
02  {
03        @Override
04        protected void onCreate(Bundle savedInstanceState)
05        {
06              super.onCreate(savedInstanceState);
07              setContentView(R.layout.second);
08
09              TextView textView1 = (TextView) findViewById(R.id.textViewshow);
10              EditText editTextPassword = (EditText) findViewById(R.id.editTextPassword);
11              Button button2 = (Button) findViewById(R.id.button2);
12              Intent intent = getIntent();
13
14              //方法一
15              String name = intent.getStringExtra("name");
16              //方法二
17              String password = intent.getExtras().getString("password");
18              //int age = intent.getExtras().getInt("age");
19              //int age = intent.getExtras().getInt("age", 20);
20              int age = intent.getIntExtra("age", 20);
21              //String[] 徒弟们 = intent.getExtras().getStringArray("徒弟们");
22              String[] 徒弟们 = intent.getStringArrayExtra("徒弟们");
23
24              textView1.append("用户名: " + name);
25              textView1.append("\n 密码: " + password);
26              textView1.append("\n 年龄: " + age);
27              for (String 徒弟 : 徒弟们)
28              {
29                    textView1.append("\n 徒弟: " + 徒弟);
30              }
31
32              //以下方法是方法二的变形
33              Bundle bundle1 = intent.getExtras();
34              editTextPassword.setText(bundle1.getString("password"));
35              textView1.append("\n 对象: " + bundle1);
36
37              button2.setOnClickListener(new View.OnClickListener()
38              {
39                    @Override
40                    public void onClick(View v)
41                    {
42                          Intent intent = new Intent();
43                          intent.putExtra("password", editTextPassword.getText().toString());
44                          setResult(20, intent);
45                          finish();//关闭当前 Activity
46                    }
47              });
48        }
49  }
```

SecondActivity 可接收 FirstActivity 传来的键值对或 Bundle 对象,然后将处理结果回传给 FirstActivity。

第 15 行使用 getStringExtra()方法通过参数中的字符串 name 获取对应值。此种方式获取键值的方式最简单。

第 17 行通过 getExtras()方法的 getString()方法获取键值对中的值,效果与 getStringExtra()完全相同。

针对不同的数据类型或数组,SDK 提供了 getInt()或 getStringArrayExtra()等不同的方法。第 18～20 行代码效果基本相同。第 18 行的代码如果没有查到键值对 age,则返回默认值 0,而第 19 和 20 行会返回指定的默认值 20。

第 33 行使用 intent.getExtras()方法将返回结果传递给 bundle1,在连续调用键值对时可提高效率。

第 21 行和第 22 行的效果完全相同,与前面键值对的区别是 key 为中文。Java 是支持中文作为变量名的。

【注】 Java 标识符要求:

(1) Java 是大小写敏感的语言。

(2) Java 的保留字不能作为标识符。

(3) 不能以数字开头。

(4) 不能含有空格、@、#、一等非法字符。

(5) 能见名知义。

第 43 行在回传的 intent 中绑定 password 键值对。

第 44 行的 setResult()方法会将第一个参数 resultCode 和第二个参数 intent 回传给 FirstActivity 的 onActivityResult()方法。至此完成了参数值的传送和回传功能。

Activity 间传值运行结果如图 5-2～图 5-4 所示。

图 5-2 FirstActivity 传值

图 5-3 SecondActivity 取值并处理

本案例是不是很像之前讲过的 AlertDialog？外观上的区别就是 SecondActivity 不是对话框。如果在 AndroidManifest.xml 文件 SecondActivity 所在 Activity 标签中增加属性 **android:theme="@style/Theme.AppCompat.Dialog"**，再次运行，SecondActivity 变成对话框外观。对话框风格的 Activity 运行结果如图 5-5 所示。

图 5-4　FirstActivity 取回处理值

图 5-5　对话框风格的 Activity 运行结果

5.2　Activity

布局文件实现 Android 应用程序的界面设计，而 Activity 作为 Android 的应用组件，通过 Java 代码设计实现 UI 交互功能。一个 App 通常由一个或多个 Activity 组成。

5.2.1　系统状态栏、标题栏和导航栏


视频讲解

默认每个 Activity 的界面都会显示系统状态栏、标题栏和导航栏。对于某些应用，需要将以上三者部分或全部隐藏。如果要实现 Activity 调用时就不显示标题栏，最简单的办法是在 AndroidManifest.xml 文件的 application 或 activity 标签中加入以下代码：

```
android:theme = "@android:style/Theme.NoTitleBar"
```

上述代码加在 application 标签中，表示所有的 Activity 都不显示标题栏。上述代码加在 activity 标签中，表示当前 activity 不显示标题栏。如果在上述节点位置加入：

```
android:theme = "@android:style/Theme.NoTitleBar.Fullscreen"
```

代表相应的 Activity 不显示系统状态栏和标题栏。其他几种隐藏系统状态栏和标题栏的方法都是在 Activity 的 Java 文件中进行设置的。

【FirstActivity.java】

```
01   public class FirstActivity extends Activity
02   {
03        @Override
04        public void onCreate(Bundle savedInstanceState)
05  {
06             super.onCreate(savedInstanceState);
07             //requestWindowFeature(Window.FEATURE_NO_TITLE);
08             setContentView(R.layout.main);
09
10             //getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.
               //LayoutParams.FLAG_FULLSCREEN);
11
12             Button button1 = (Button) findViewById(R.id.button1);
13             Button button2 = (Button) findViewById(R.id.button2);
14
15             //方案五: 隐藏/显示标题栏
16             button1.setOnClickListener(new View.OnClickListener()
17             {
18                  @Override
19                  public void onClick(View v)
20                  {
21                       ActionBar actionBar = getActionBar();
22                       if (button1.getText().toString().equals(getString(R.string.
                         titleVisible)))
23                       {
24                            actionBar.show();
25                            button1.setText(getString(R.string.titleInvisible));
26                            return;
27                       }
28                       actionBar.hide();
29                       button1.setText(getString(R.string.titleVisible));
30                  }
31             });
32
33             //隐藏系统状态栏和导航栏
34             View decorView = getWindow().getDecorView();
35             button2.setOnClickListener(new View.OnClickListener()
36             {
37                  @Override
38                  public void onClick(View v)
39                  {
40
41                       if (button2.getText().toString().equals(getString(R.string.
                         statusVisible)))
42                       {
43                            //decorView.setSystemUiVisibility(View.VISIBLE);
44                            decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_VISIBLE);
45                            button2.setText(getString(R.string.statusInvisible));
46                       } else
```

```
47                          {
48                              decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_FULLSCREEN |
                                View.SYSTEM_UI_FLAG_HIDE_NAVIGATION | View.SYSTEM_UI_FLAG_
                                IMMERSIVE);
49                              button2.setText(getString(R.string.statusVisible));
50                          }
51                      }
52                  });
53          }
54      }
```

Activity 内各 View 间关系如图 5-6 所示。因此显示了 Activity 内各 View 的大致关系，具体逻辑关系可在 Android Studio 运行程序后选择 Android Studio 右下角的 Layout Inspector 选项卡，在弹出的界面中查看各个 View 之间的关系嵌套，如图 5-7 所示。每个 Activity 对应一个 Window。DecorView 是 Window 中的顶层视图，StatusBar 是系统状态栏，ActionBar 是标题栏，Navigation 是导航栏，ContentView 就是第 8 行 setContentView()方法调入的布局文件形成的 View。隐藏或显示系统状态栏、标题栏和导航栏就是对 Activity 中的相应 View 执行隐藏或显示操作。

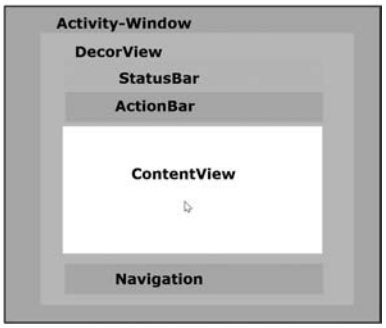

图 5-6　Activity 内各 View 间关系

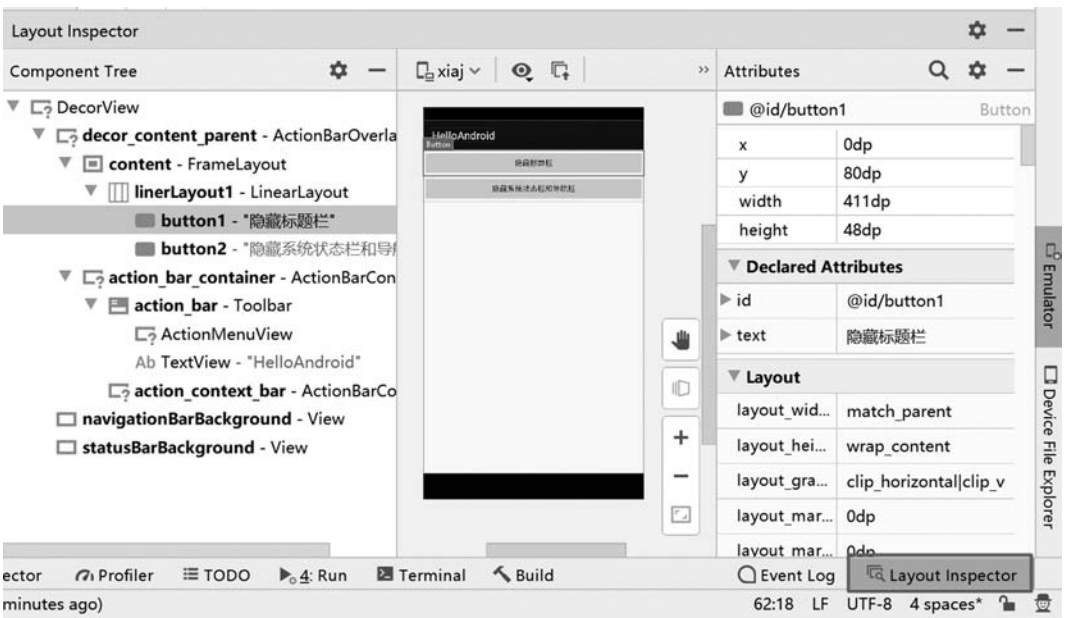

图 5-7　用 Layout Inspector 查看各 View 间关系

如果去掉第 7 行的注释，当前 Activity 将不显示标题栏。此命令必须在第 8 行 setContentView()方法之前，否则程序运行出错。加上之前介绍的隐藏标题栏的方法，如果再执行第 24 行或第 28 行的 actionBar 操作时会因标题栏设置冲突而出错。上述介绍的几

种隐藏系统状态栏和标题栏的方法都无法在 Activity 启动以后再进行动态变更。

第 10 行的代码可放置在 setContentView()方法前后任意位置(有些特殊属性需将此方法放在 setContentView()方法前),通过 getWindow()方法获取 Window,然后用 setFlags()方法设置标记来实现全屏效果。

第 16～31 行定义 button1 的单击监听器,通过第 22 行判断按钮文字内容来执行标题栏的显示或隐藏操作。第 21 行获取标题栏对象并赋予 actionBar。第 24 行 actionBar 的 show()方法用来显示标题栏,第 28 行的 hide()方法用来隐藏标题栏。

第 34 行的 getWindow()方法获取当前 Activity 对应的 Window,getDecorView()方法获取当前 Window 包含的 View。第 43 行和第 44 行效果是一样的,利用 decorView 实例的 setSystemUiVisibility()方法设置 UI 的相关对象是否显示。其中,常量的含义为:

(1) View.SYSTEM_UI_FLAG_VISIBLE:系统状态栏和导航栏都显示。

(2) View.SYSTEM_UI_FLAG_FULLSCREEN:显示界面变为全屏,此时不显示系统状态栏。

(3) SYSTEM_UI_FLAG_HIDE_NAVIGATION:隐藏导航栏。

(4) View.SYSTEM_UI_FLAG_IMMERSIVE:设置为沉浸模式。

以上常量可以多个同时使用,中间用竖线"|"分隔。第 48 行如果去掉 View.SYSTEM_UI_FLAG_IMMERSIVE,单击按钮隐藏系统状态栏和导航栏后再次单击屏幕,系统状态栏和导航栏又会再次显示。加入沉浸模式可以避免此类问题。

对于 API 30 的 Android 系统,引入了新的对象和命令,下列代码实现隐藏系统状态栏和导航栏。如果将 hide()方法换成 show()方法,则变为显示相关对象。

```
WindowInsetsController windowInsetsController = getWindow().getInsetsController();
    if (windowInsetsController != null)
    {
        if (button3.getText().toString().equals(getString(R.string.API30Visible)))
        {
            windowInsetsController.hide(WindowInsets.Type.statusBars());
            windowInsetsController.hide(WindowInsets.Type.navigationBars());
            button3.setText(getString(R.string.API30Invisible));
        }
    }
```

5.2.2 关闭 Activity

可以通过关闭 Activity 来切换 Activity 或者关闭整个 App。本案例在布局文件中添加了一个 EditText 和 Button。当运行不同的关闭代码时,可以通过列表键对比退出效果。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      @Override
04      public void onCreate(Bundle savedInstanceState)
05      {
06          super.onCreate(savedInstanceState);
```

```
07              setContentView(R.layout.main);
08
09              Button button1 = (Button) findViewById(R.id.button1);
10              button1.setOnClickListener(new View.OnClickListener()
11              {
12                    @Override
13                    public void onClick(View v)
14                    {
15                        //方法一
16                        finish();
17                        //方法二
18                        //onDestroy();//注：高版本 SDK 无法关闭 Activity
19                        //方法三
20                        //System.exit(0);
21                        //方法四
22                        //Intent intent = new Intent(Intent.ACTION_MAIN);
23                        //intent.addCategory(Intent.CATEGORY_HOME);
24                        //startActivity(intent);
25                        //方法五 kill 当前进程
26                        //android.os.Process.killProcess(android.os.Process.myPid());
27                    }
28              });
29        }
30    }
```

每次运行程序时先修改文本输入框中的内容，然后再单击“退出”按钮，执行相应的关闭 Activity 命令。为了方便后续的讲解，术语“执行唤回”是指单击列表键，再单击程序列表中的 HelloAndroid 程序。修改文本输入框中内容的目的是对比执行唤回操作后文本输入框是否能保留关闭前的值。

执行第 16 行的 finish()方法，单击列表键能看到 HelloAndroid 程序中的控件，执行唤回操作，文本输入框中的值还原为初始值。

第 18 行的 onDestroy()方法在新版本中已无法关闭 Activity。

执行第 20 行 System.exit(0)代码退出后，单击列表键在运行程序列表中看不到控件，说明相关资源已被清除。执行唤回操作相当于重启 HelloAndroid 程序。

第 22 行开始的方法四是将系统桌面作为切换的 intent 来处理，等效于按 Home 键。所以程序列表中能看到控件，执行唤回操作后文本输入框中的内容也保持不变。相当于将后台的 HelloAndroid 程序切换回前台。

第 26 行的命令是在 Android 操作系统级执行杀死当前进程操作，因此退出后程序列表中看不到控件。

5.2.3 生命周期

在实际的 App 开发中，Activity 往往不止一个。在多个 Activity 间切换时，存在相关控件是否还保留 Activity 切换前的值，或者是否需要刷新以获取新值的问题。在之前的案例中，控件的初始化赋值等代码都是放在 onCreate()方法中的，而为了保证 Activity 切换以后

程序能按正常逻辑运行,有些代码就需要放在 Activity 的其他方法中。Activity 生命周期如图 5-8 所示。

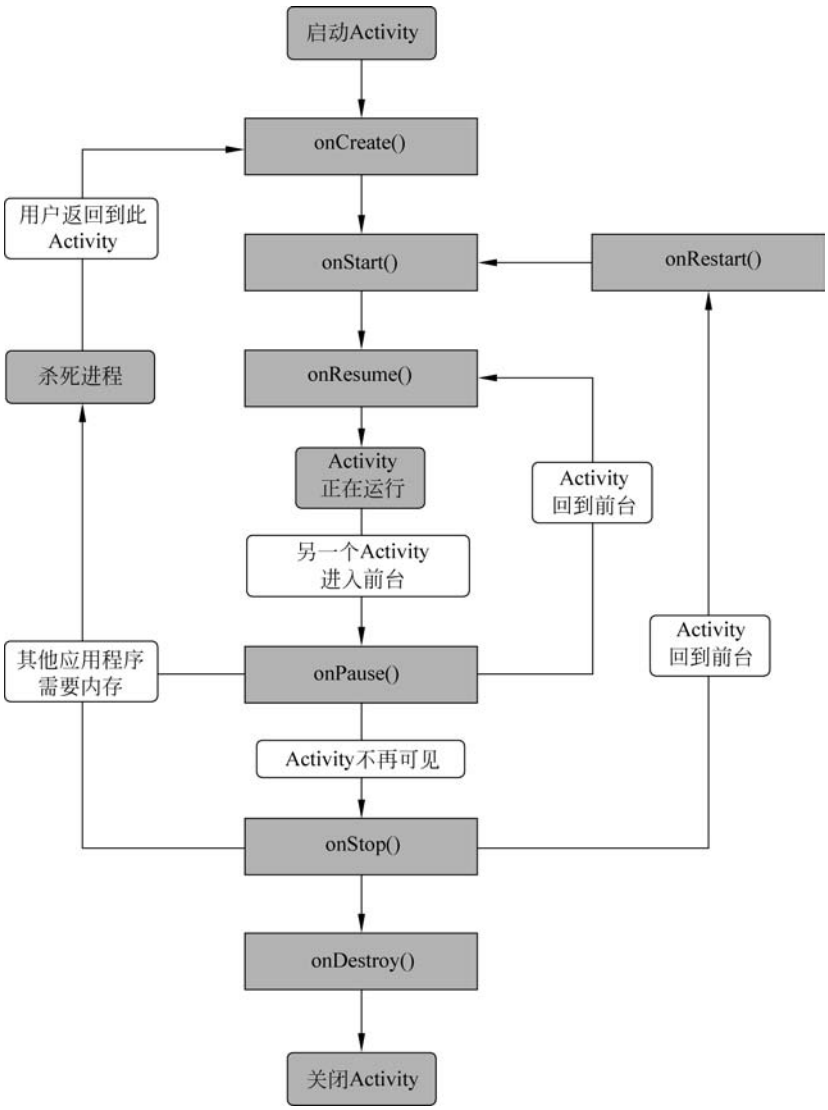

图 5-8　Activity 生命周期

从图 5-8 可以看出,除了第一次启动 Activity 或者已启动的 Activity 资源被 Android 系统释放后重启会执行 onCreate()方法外,其他如 Activity 被遮挡后重新回到前台显示、切换程序后 Activity 回到前台显示等情况都会跳过 onCreate()方法,此时可能从 onRestart()方法、onStart()方法或 onResume()方法开始执行。因此可根据实际流程,将相关代码放在不同的方法中。为了便于理解,可以把 Activity 分成 3 个不同范围的生命周期。

(1) 完整生命周期:从 onCreate()方法到 onDestroy()方法。

(2) 可见生命周期:从 onStart()方法到 onStop()方法。顾名思义,是 Activity 从显现到消失的周期。如果当前 Activity 被其他 View 部分遮挡,当前 Activity 就还在可见生命周

期内；如果是完全遮挡就退出可见生命周期。

(3) 前台生命周期：从 onResume()方法到 onPause()方法，此周期内的控件可见且可交互。只要当前 Activity 被其他 View 遮挡(不论是全部或部分遮挡)都退出前台生命周期。

为了验证 Activity 在生命周期的流转过程，案例中设计两个 Activity，因篇幅限制，本书只列出了 FirstActivity.java 的代码，SecondActivity.java 代码与其大同小异。程序重写了所有生命周期中的方法，两个 Activity 代码不同的地方是各个方法中使用 Log.i()方法输出的内容，指明分别是由哪一个 Activity 的哪一个方法执行的。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      private Button button1;
04
05      @Override
06      public void onCreate(Bundle savedInstanceState)
07      {
08          Log.i("xj", "第一个 Activity ---> onCreate");
09          super.onCreate(savedInstanceState);
10          setContentView(R.layout.main);
11          button1 = (Button) findViewById(R.id.myButton);
12          button1.setOnClickListener(new ButtonOnClickListener()); //设置监听器
13      }
14
15      @Override
16      protected void onStart()
17      {
18          Log.i("xj", "第一个 Activity ---> onStart");
19          super.onStart();
20      }
21
22      @Override
23      protected void onRestart()
24      {
25          Log.i("xj", "第一个 Activity ---> onRestart");
26          super.onRestart();
27      }
28
29      @Override
30      protected void onResume()
31      {
32          Log.i("xj", "第一个 Activity ---> onResume");
33          super.onResume();
34      }
35
36      @Override
37      protected void onPause()
38      {
```

```
39              Log.i("xj", "第一个 Activity ---> onPause");
40              super.onPause();
41          }
42
43          @Override
44          protected void onStop()
45          {
46              Log.i("xj", "第一个 Activity ---> onStop");
47              super.onStop();
48          }
49
50          @Override
51          protected void onDestroy()
52          {
53              Log.i("xj", "第一个 Activity ---> onDestory");
54              super.onDestroy();
55          }
56
57  //以下方法不属于生命周期触发事件 ========================================
58          @Override
59          protected void onSaveInstanceState(Bundle outState)
60          {
61              super.onSaveInstanceState(outState);
62              Log.i("xj", "第一个 Activity ---> onSaveInstanceState");
63          }
64
65          @Override
66          public void onBackPressed()
67          {
68              super.onBackPressed();
69              Log.i("xj", "第一个 Activity ---> onBackPressed()");
70          }
71
72          class ButtonOnClickListener implements OnClickListener
73          {
74              @Override
75              public void onClick(View v)
76              {
77                  Intent intent = new Intent();
78                  intent.setClass(FirstActivity.this, SecondActivity.class);
79                  FirstActivity.this.startActivity(intent); //启动第二个 Activity
80                 //finish();                                  //注意观察有无此命令时的变化
81                 //System.exit(0);                            //注意观察有无此命令时的变化
82              }
83          }
84  }
```

第 58～70 行的方法不属于生命周期的方法，onSaveInstanceState()方法主要在切换 Activity、单击 HOME 键、开关电源键等情况时被调用。onBackPressed()方法在单击返回

键时被调用。

结合生命周期图，变更以下条件并切换 Activity，观察调试信息的输出变化。

(1) 观察两个 Activity 切换时的状态变化。

(2) 观察单击 HOME 键并再次运行 App 的变化。

(3) 观察横竖屏切换的状态变化(新版本无变化)。

(4) 观察单击列表键后的变化。

(5) 观察单击返回键后的变化。

(6) 观察将 SecondActivity 变为 Dialog 主题后的变化(部分遮挡)，可细分为单击按钮返回 FirstActivity、单击 FirstActivity 区域返回和单击返回键 3 种情况。

(7) 观察添加 finish()命令时的变化。

(8) 观察添加 System.exit(0)命令时的变化。

(9) 观察开关电源键时的变化。

5.3 电话及动态授权

Android 系统已经内置完善的拨打电话功能，并提供了相应的调用接口。本案例演示使用系统界面拨号和拨打电话功能。通过拨打电话功能还可以了解 Android 的动态授权的工作机制。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      @Override
04      public void onCreate(Bundle savedInstanceState)
05      {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.main);
08
09          Button button1 = (Button) findViewById(R.id.button1);
10          Button button2 = (Button) findViewById(R.id.button2);
11          Uri uri = Uri.parse("tel:5556");          //设置电话号码
12          //直接调用拨号界面，但并不拨出
13          button1.setOnClickListener(new View.OnClickListener()
14          {
15              @Override
16              public void onClick(View v)
17              {
18                  Intent intent = new Intent(Intent.ACTION_DIAL, uri);
19                  startActivity(intent);
20              }
21          });
22
23          //直接调用拨号界面并拨出
24          button2.setOnClickListener(new View.OnClickListener()
25          {
```

```
26                  @Override
27                  public void onClick(View v)
28                  {
29                      if (checkSelfPermission(Manifest.permission.CALL_PHONE) !=
                        PackageManager.PERMISSION_GRANTED)
30                      {
31                          requestPermissions(new String[]{Manifest.permission.CALL_
                            PHONE}, 1);
32                          return;
33                      }
34                      Intent intent = new Intent(Intent.ACTION_CALL, uri);
35                      startActivity(intent);
36                  }
37              });
38      }
39      @Override
40      public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
        @NonNull int[] grantResults)
41      {
42          Log.i("xj", "permissions[0] = " + permissions[0] + "\ngrantResults[0] = "
            + grantResults[0]);
43      }
44  }
```

第 11 行定义 Uri 设置要拨打的电话号码,短信、邮件、网址等也可以采用此方式定义。tel 代表电话,5556 是要拨打电话的号码。当前模拟器电话号码是 5554,同时也是模拟器与 adb 服务通信的 TCP 端口号。模拟器完整的电话号码是 15555215554。5555 是 adb 服务的 TCP 端口号。如果启动多个模拟器,第二个模拟器的号码就是 5556,以此类推。本案例为了验证拨出电话功能,需要启动两个模拟器。在 Terminal 窗口中输入 **adb devices** 命令,返回如下已连接的模拟器和真实 Android 设备信息:

```
List of devices attached
9YE0218418013349        device
emulator-5554   device
emulator-5556   device
```

其中,第 1 行的设备信息是真实的手机信息,后面两行是模拟器的信息,模拟器信息中的数字就是电话号码,也是 TCP 端口号。早期版本的模拟器会将此号码直接显示在模拟器窗口的标题栏上。

第 13～21 行定义 button1 单击监听器,完成拨号功能。其中,第 18 行使用 Intent 的 ACTION_DIAL 常量调用 Android 系统的电话拨号界面,并预置 uri 中的电话号码 5556。执行第 19 行启动 intent 命令显示程序初始界面,如图 5-9 所示。

单击 button1(显示“电话拨号,并不拨出”)按钮显示电话拨号界面,如图 5-10 所示。此时要拨打的电话号码 5556 已显示,用户还需要单击拨出键才能完成拨打电话。

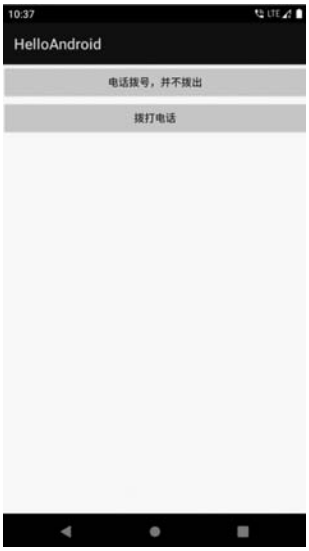

图 5-9　程序初始界面

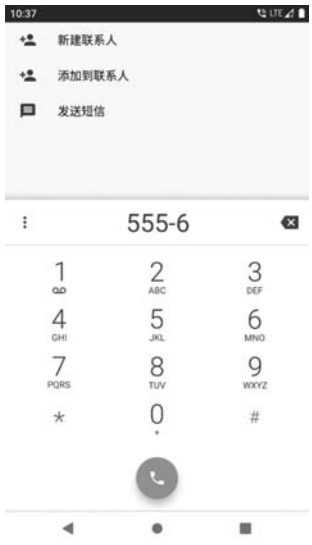

图 5-10　电话拨号界面

第 24～37 行定义 button2 单击监听器，完成拨打电话功能。拨打电话需要申请电话呼叫权限。早期的 Android 版本申请权限只要在 AndroidManifest. xml 中添加**< uses-permission android：name＝"android. permission. CALL_PHONE" />**权限即可。当用户安装 App 时，会显示需要电话呼叫权限界面，但绝大部分用户都不会查看权限列表，而是直接单击“下一步”按钮继续安装，这将导致一些恶意软件乘虚而入。从 API 23 开始，对于一些关键权限（如电话呼叫、发送短信、位置信息、存储读取等）增加了动态授权功能，即用户在运行 App 并使用到相关功能时弹出权限申请界面，如图 5-11 所示。如果选择拒绝，则下一次拨打电话时又会弹出动态授权界面，同时选项中多了一项“拒绝，不要再询问”，如图 5-12 所示。如果选择“拒绝，不要再询问”选项，后续再单击拨打电话按钮时不再弹出动态授权界面，

图 5-11　首次提示电话拨打动态授权

图 5-12　多次提示电话拨打动态授权

也就无法完成拨打电话功能。如果选择“允许”选项,则返回程序界面,当再次单击“拨打电话”按钮时,自动调用拨号界面并完成电话拨号和拨打功能,接收方手机将显示来电提示。第 29 行用来判断是否已经授予 CALL_PHONE 权限,如果没有相应权限就执行第 31 行弹出动态权限申请界面。如果已经有权限,则直接执行第 34 行和第 35 行调用系统拨号界面并拨打电话。

“动态授权”对话框与案例程序是异步执行的。本案例中执行第 31 行将弹出“动态授权”对话框,此时用户不进行任何操作,程序也会继续执行。因此在第 32 行加入 return 命令,防止因为未获取拨打电话权限而继续执行第 34～35 行引发异常。也可以将第 34～35 行代码放入 try-catch 中捕获异常,保证程序正常运行。还有一种方式是将第 34～35 行代码放入第 40～43 行的回调方法 onRequestPermissionsResult()中。当调用第 31 行的 requestPermissions()方法时,系统都会自动调用 onRequestPermissionsResult()方法返回动态授权的用户操作结果,其中字符串数组 permissions 是动态授权申请的权限,数组长度由动态授权申请的权限数量决定,整型数组 grantResults 是动态授权用户操作结果,－1 代表用户选择了“拒绝”选项,0 代表用户选择了“允许”选项。

5.4 发送短信

下面的案例演示用两种方法发送短信。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      @Override
04      public void onCreate(Bundle savedInstanceState)
05      {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.main);
08
09          Button button1 = (Button) findViewById(R.id.button1);
10          Button button2 = (Button) findViewById(R.id.button2);
11          EditText editText1 = (EditText) findViewById(R.id.editText1);
12          EditText editText2 = (EditText) findViewById(R.id.editText2);
13          editText1.setText("5554");
14          editText2.setText("张三又来了");
15
16          button1.setOnClickListener(new View.OnClickListener()
17          {
18              @Override
19              public void onClick(View v)
20              {
21                  if (checkSelfPermission(Manifest.permission.SEND_SMS) !=
                    PackageManager.PERMISSION_GRANTED)
22                  {
23                      requestPermissions(new String[]{Manifest.permission.SEND_
                        SMS}, 1);
```

```
24                        } else
25                        {
26                              //方法一：使用 SmsManager
27                              SmsManager sms = SmsManager.getDefault();
28                              PendingIntent pendingIntent = PendingIntent.getBroadcast
                                (FirstActivity.this, 0, new Intent(), 0);
29                               sms.sendTextMessage(editText1.getText().toString(), null,
                                 editText2.getText().toString(), pendingIntent, null);
30                        }
31                  }
32            });
33
34            button2.setOnClickListener(new View.OnClickListener()
35            {
36                  @Override
37                  public void onClick(View v)
38                  {
39                        //方法二：使用 Intent 调用系统短信
40                        Uri uri = Uri.parse("smsto://" + editText1.getText().toString());
41                        Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
42                        intent.putExtra("sms_body", editText2.getText().toString());
43                        startActivity(intent);
44                  }
45            });
46      }
47  }
```

方法一是使用 SmsManager 发送短信。第 21 行和第 23 行完成动态授权判断和申请功能。第 27～28 行初始化发送短信对象，第 29 行用 sendTextMessage()发出短信。

方法二是使用 Android 系统自带发送短信界面，第 40～43 行的代码只是设置接收短信号码和发送短信的内容，短信还未发送，所以不需要动态授权。

使用 SmsManager 发送短信的代码存在一个问题，每个短信只能容纳 1120b，按每个 ASCII 码 7b 计算，有 160 个字符，折合 GBK 汉字有 70 个字符。如果 ASCII 和汉字混用则全部按汉字计算，总字符也不能超过 70 个。包含汉字的短信如果长度超出 70 个汉字，一种解决方案是在程序中将超出长度的短信进行拆分，然后依次发送。此时接收端会收到多条独立的短信。在实际生活中经常会收到一封短信中包含超过 70 个汉字的内容，这是调用 sendMultipartTextMessage()方法实现的。当长短信超过 70 个汉字时，每条短信的前 48b 作为长短信头部结构标识，所以每条长短信可以发送 67 个汉字。当短信长度小于或等于 70 时，为 1 条短信；当 70＜短信长度≤134 时，为 2 条短信，以此类推。接收方按长短信格式将若干条短信合成一条长短信，逻辑上是一条短信，实际上还是由多条短信合成的。使用下述代码即可实现长短信发送。

```
01  SmsManager sms = SmsManager.getDefault();
02  ArrayList<String> smsList = sms.divideMessage(editText2.getText().toString());
03  PendingIntent pendingIntent = PendingIntent.getBroadcast(FirstActivity.this, 0, new
    Intent(), 0);
```

```
04  ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
05  for (int i = 0; i < smsList.size(); i++)
06  {
07        sentIntents.add(pendingIntent);
08  }
09    sms. sendMultipartTextMessage ( editText1. getText ( ). toString ( ), null, smsList,
      sentIntents, null);
```

第 2 行调用 divideMessage()方法实现长短信字符串的分割。

第 3～8 行设置长短信相关 Intent。

第 9 行调用 sendMultipartTextMessage()方法发送长短信。

由于发送短信属于重要权限,很多国产手机的 Android 系统会对发送短信权限做出更多提示,有的会弹出风险提示框,提示哪一个应用在试图发送短信,用户选择的权限又分为“本次允许”和“始终允许”等,最终目的都是提高设备使用的安全性。

5.5 Menu

Menu 菜单属于 Android 变动比较大的控件。早期 Android 设备的导航栏按键分别为返回键、主页键和菜单键。如果 Activity 中包含菜单,用户单击菜单键就可以调出菜单。后来 Android 将菜单键改为了列表键,调出菜单的方式也改成了在标题栏上单击菜单图标 。菜单的外观也由早期的在屏幕下方显示的 2×3 列表变成了右上方弹出单列列表。

视频讲解

5.5.1 构建菜单

【FirstActivity.java】

```
01   public class FirstActivity extends Activity
02   {
03        @Override
04        public void onCreate(Bundle savedInstanceState)
05        {
06             super.onCreate(savedInstanceState);
07             setContentView(R.layout.main);
08        }
09
10        @Override
11        public boolean onCreateOptionsMenu(Menu menu)
12        {
13             menu.add(0, 1, 2, "红色"); //添加菜单(int groupId,int itemId,int order)
14             menu.add(0, 2, 2, "绿色");
15
16             menu.add(1, 1, 6, "蓝色");
17             menu.add(1, 2, 5, "紫色");
18
19             menu.add("白色");
20             menu.add("黑色");
```

```
21
22              menu.add(0, 3, 4, "橙色 1");
23              menu.add(0, 4, 5, "橙色 2");
24              //menu.removeGroup(0); //移除指定组的菜单
25              //menu.setGroupEnabled(0, false);
26              //menu.setGroupVisible(0, false);
27              return super.onCreateOptionsMenu(menu);
28          }
29      }
```

Activity 的菜单是通过第 11～28 行的 onCreateOptionsMenu()方法来构建的，对其自带的形参 menu 执行 add()方法建立菜单项。第 13 行的 add()方法的参数分别为 groupId(菜单项分组号)、itemId(菜单项 id)、order(菜单项排列顺序，按升序排序)和 title(菜单项文字)。如果 order 相同则按 add()方法执行的先后顺序排列。

第 19 行的 add()方法只有 title 参数，则 groupId、itemId 和 order 都为默认值 0。建议对添加的菜单项按功能分组，所有菜单项的 itemId 设置不同的值，便于后续编程时简化判断选择的菜单项。

第 24 行的 removeGroup(0)的作用是将 groupId 为 0 的分组菜单移除。这里有个缺陷一直未修复：当移除分组的菜单项 order 大于或等于后续其他分组菜单项的 order 值，或者 order 小于或等于前方其他分组单项的 order 值时，无法用 removeGroup()方法移除。

第 25 行是将 groupId 为 0 的分组菜单项设置为不可用，菜单文字变为灰色。此时菜单项无法单击使用。

第 26 行是将 groupId 为 0 的分组菜单项设置为不可见。此时菜单项还是分配了地址空间，这与 removeGroup()删除分配空间是不同的。

setGroupVisible(0, false)运行结果如图 5-13 所示。

图 5-13　setGroupVisible(0,false)运行结果

5.5.2　响应菜单项单击

视频讲解

```
【FirstActivity.java】
01  public class FirstActivity extends Activity
02  {
03      TextView textView1;
04      @Override
05      public void onCreate(Bundle savedInstanceState)
06      {
07          super.onCreate(savedInstanceState);
08          setContentView(R.layout.main);
09          textView1 = (TextView) findViewById(R.id.textView1);
10          textView1.append("\n 执行了 onCreate()");
```

```
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuItem menu1 = menu.add(0, 1, 1, "红色");
        MenuItem menu2 = menu.add(0, 2, 2, "黑色");
        textView1.append("\n 执行了 onCreateOptionsMenu()")
        //方法一: 使用监听器响应单击菜单操作,其优先级高于 onOptionsItemSelected()方法
        MenuItem.OnMenuItemClickListener onMenuItemClickListener = new MenuItem.
        OnMenuItemClickListener()
        {
            @Override
            public boolean onMenuItemClick(MenuItem item)
            {
                switch (item.getItemId())
                {
                    case 1:
                        textView1.setTextColor(Color.RED);
                        textView1.append("\nmenu1.setOnMenuItemClickListener:" +
                        item.getTitle());
                        break;
                    case 2:
                        textView1.setTextColor(Color.BLACK);
                        textView1.append("\nmenu2.setOnMenuItemClickListener:" +
                        item.getTitle());
                        break;
                }
                //return true;  //不再响应 onOptionsItemSelected()方法
                return false;
            }
        };

        menu1.setOnMenuItemClickListener(onMenuItemClickListener);
        menu2.setOnMenuItemClickListener(onMenuItemClickListener);
        return super.onCreateOptionsMenu(menu);
    }

    //方法二: 内置 onOptionsItemSelected()方法
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            case 1:
                textView1.setTextColor(Color.RED);
                break;
             case 2:
                textView1.setTextColor(Color.BLACK);
                break;
```

```
58              }
59              textView1.append("\n 菜单组" + item.getGroupId() + "\n 菜单项" + item.
                getItemId() + "\n 菜单顺序: " + item.getOrder() + "\n 菜单标题: " + item.
                getTitle());
60              return super.onOptionsItemSelected(item);
61          }
62      }
```

当用户单击菜单项时,响应单击操作的方法有如下两种。

方法一:使用 OnMenuItemClickListener 监听器响应菜单操作,其优先级高于 Activity 下的 onOptionsItemSelected()方法。第 25 行通过 getItemId()方法获取菜单项的 itemId,如果 itemId 是唯一的,就无须再去判断分组或者是菜单项的 title,可以简化判断选中菜单项的代码。监听器的返回值如果采用第 36 行的 true,将不再执行 onOptionsItemSelected()方法。监听器返回 true 的 Menu 运行结果如图 5-14 所示。如果设为 false,在执行完监听器以后接着执行 onOptionsItemSelected()方法。监听器返回 false 的 Menu 运行结果如图 5-15 所示。

图 5-14　监听器返回 true 的 Menu 运行结果

图 5-15　监听器返回 false 的 Menu 运行结果

方法二:使用 Activity 下的 onOptionsItemSelected()方法。方法的框架可依次选择 Android Studio 菜单 Code→Override Methods,在弹出的窗口中选择 onOptionsItemSelected()方法,Android Studio 会构建此方法的代码框架。onOptionsItemSelected()方法的使用与方法一中的 onCreateOptionsMenu()方法类似,两者的关系:onCreateOptionsMenu()方法在 Activity 启动时运行一次,可初始化菜单和菜单项单击监听器,用户单击菜单项由菜单项单击监听器响应,如果监听器的返回值为 false,则继续调用 onOptionsItemSelected()方法。

用户在使用 App 的过程中可能会无暇关注右上角的菜单按钮,有两种解决方案。

(1) 在按钮的监听器中加入"**openOptionsMenu**();"命令,当用户单击按钮时执行此命令,自动运行 onCreateOptionsMenu()方法实现弹出菜单。

(2) 长按某个控件弹出上下文菜单 ContextMenu。

5.5.3　ContextMenu


视频讲解

用户可长按不同的控件弹出不同的上下文菜单 ContextMenu,前提条件是这些控件都注册了 ContextMenu。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      TextView textView1, textView2;
```

```
    EditText editText1;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView1 = (TextView) findViewById(R.id.textView1);
        textView2 = (TextView) findViewById(R.id.textView2);
        editText1 = (EditText) findViewById(R.id.editText1);

        registerForContextMenu(textView1); //注册 textView1 上下文菜单
        registerForContextMenu(editText1); //注册 editText1 上下文菜单
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
    {
        textView2.append("\n 重新初始化 menu");
        if (v == textView1)                    //如果长按 TextView
        {
            menu.add(0, 1, 1, "红色");
            menu.add(0, 2, 2, "黑色");
            menu.setHeaderTitle("请选择 TextView 字体颜色");  //设置标题栏文字
            return;
        }
        //如果长按 EditText
        menu.add(1, 3, 11, "红色");
        menu.add(1, 4, 12, "蓝色");
        menu.setHeaderTitle("请选择 EditText 字体颜色");
        menu.setHeaderIcon(R.drawable.icon); //设置标题栏图标

        super.onCreateContextMenu(menu, v, menuInfo);
    }

    @Override
    public boolean onContextItemSelected(MenuItem item)
    {
        switch (item.getItemId())              //判断被单击的菜单项
        {
            case 1:
                textView1.setTextColor(Color.RED);
                break;
            case 2:
                textView1.setTextColor(Color.BLACK);
                break;
            case 3:
                editText1.setTextColor(Color.RED);
                break;
            case 4:
```

```
54                      editText1.setTextColor(Color.BLUE);
55                      break;
56              }
57              return super.onContextItemSelected(item);
58          }
59
60          @Override
61          public void onContextMenuClosed(Menu menu)
62          {
63              textView2.append("\n退出了上下文菜单:" + menu.toString());
64              super.onContextMenuClosed(menu);
65          }
66      }
```

第15～16行通过registerForContextMenu()方法将注册textView1和editText1上下文菜单。长按这两个控件就会自动调用第20行的onCreateContextMenu()方法。

第22行的代码是为了演示每次调用上下文菜单时,都会重新运行onCreateContextMenu()方法,这与Menu的onCreateOptionsMenu()方法只运行一次是不同的。

第61行的onContextMenuClosed()方法在退出上下文菜单时运行,menu.toString()会显示上下文菜单的地址,注意观察输出结果,即使单击相同的菜单项,每一次的上下文菜单地址也是不同的。

第23～29行是生成textView1的ContextMenu,第31～34行生成editText1的ContextMenu。

第40～58行是上下文菜单项被单击后的响应代码,与之前案例的Menu代码类似。

【注】 如果没有特别提示,长按控件才弹出上下文菜单的方式很容易被用户忽略。每一次都要重新构建上下文菜单,效率也略显低下。

5.6 Notification

Notification(通知)可独立于App显示在系统下拉通知栏中。由于很多App会发送大量的通知,Android对通知的限制也日趋严格。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      Notification notification;                  //Notification对象
04      NotificationManager notificationManager;    //NotificationManager对象
05      static final int NOTIFY_ID = 1234;          //通知管理器通过通知id来标识不同的通
                                                    //知,创建和删除通知需要提供通知id
06
07      @Override
08      public void onCreate(Bundle savedInstanceState)
09      {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.main);
```

```
12
13          Button button1 = (Button) findViewById(R.id.button1);
14          Button button2 = (Button) findViewById(R.id.button2);
15          TextView textView1 = (TextView) findViewById(R.id.textView1);
16
17          button1.setOnClickListener(new View.OnClickListener()
18          {
19              @Override
20              public void onClick(View v)
21              {
22                  //NotificationManager 是一个系统 Service,必须通过
                    //getSystemService()方法来获取
23                  notificationManager = (NotificationManager) getSystemService(NOT-
                    IFICATION_SERVICE);
24
25                  //API 26 以上需要建立 CHANNEL
26                  if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_
                    CODES.O)
27                  {
28                      String Channel_Id = "1";                    //定义通道 id
29                      CharSequence name = "my_channel";           //通道名称
30                      int importance = NotificationManager.IMPORTANCE_HIGH;
                                                                    //通知的重要级别
31
32                      NotificationChannel notificationChannel = new NotificationChannel
                        (Channel_Id, name, importance);
33                      notificationChannel.setDescription("Description");
34                      notificationChannel.enableLights(true);
35                      notificationChannel.setLightColor(Color.RED);
36                      notificationChannel.enableVibration(true);  //允许振动
37                      notificationChannel.setVibrationPattern(new long[]{100, 200, 300,
                        400, 500, 400, 300, 200, 400});
38                      notificationManager.createNotificationChannel(notificationChannel);
39
40                      Notification.Builder builder = new Notification.Builder
                        (getApplicationContext(), Channel_Id)
41                              .setSmallIcon(R.drawable.icon)
                                //设置状态栏中的小图片,尺寸一般建议为 24×24,这
                                //个图片同样也是在下拉状态栏中所显示,如果在那里
                                //需要更换更大的图片,可以使用 setLargeIcon
42                              .setContentTitle("Notification 标题")
43                              .setContentText("Notification 内容");
44
45                      Intent resultIntent = new Intent(getApplicationContext(),
                        FirstActivity.class);
46                      TaskStackBuilder stackBuilder = TaskStackBuilder.create(get-
                        ApplicationContext());
47                      stackBuilder.addParentStack(FirstActivity.class);
48                      stackBuilder.addNextIntent(resultIntent);
```

```
                PendingIntent resultPendingIntent = stackBuilder.getPendingIntent
                (0, PendingIntent.FLAG_UPDATE_CURRENT);
                builder.setContentIntent(resultPendingIntent);
                builder.setAutoCancel(true);//单击通知就自动消除.必须与
                                            //setContentIntent一起使用才有效

                notificationManager.notify(NOTIFY_ID, builder.build());
                                            //发出通知
            }
            else
            {
                //API 26版本以下
                Intent intent = new Intent(getApplicationContext(),FirstAc-
                tivity.class);
                PendingIntent pendingIntent = PendingIntent.getActivity(get-
                ApplicationContext(), 0, intent, 0);
                notification = new Notification.Builder(FirstActivity.this)
                        .setSmallIcon(R.drawable.icon)
                        .setTicker("Notification测试")
                        .setContentTitle("老版本Notification标题")
                        .setContentText("老版本Notification内容")
                        .setVibrate(new long[] { 500L, 200L, 200L, 500L })
                        .setContentIntent(pendingIntent)
                        .build();
                notificationManager.notify(NOTIFY_ID, notification);
            }
        }
    });

    button2.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            try
            {
                notificationManager.cancel(NOTIFY_ID);
                //取消通知.如果相关通知已经手工清除,再次调用此命令会出错
            //notificationManager.cancelAll();  //清除所有通知
            }
            catch (Exception e)
            {
                textView1.append(e.toString());
            }
          }
    });
  }
}
```

从 API 26 起,使用 NotificationManager 时需要加入通道号。程序提供了新老两个版本发送通知的代码,集合了灯光、振动等功能。Notification 可视为一个容器,显示的文字是在其中的 TextView 中,还可以添加图片甚至进度条。很多音乐播放软件设计了在通知栏中显示简单的播放界面。

第 23 行定义了 NotificationManager,用于发送通知。

第 26～54 行是 API 26 及以上版本发送通知代码,老版本的代码在第 56～69 行。

第 51 行一般设置为 true,即用户单击通知栏中的通知就自动删除该通知,也可以单击“取消通知”按钮清除通知。如果设置为 false 或者注释此行,通知栏中通知需要执行第 80 行或第 81 行代码才能删除。如果通知已经手工删除,再次单击“取消通知”按钮会抛出异常。

5.7 Service

Service(服务)是在后台可长时间运行的组件。Service 有如下两种状态。

(1) Started:此状态是 Service 通过 startService()启动服务,且可保持 Service 一直运行。其 Service 生命周期如图 5-16 的左边流程所示。

(2) Bounded:Service 是通过 bindService()绑定了服务,此时允许组件与 Service 进行“请求-应答”方式的数据交互。其 Service 生命周期如图 5-16 的右边流程。

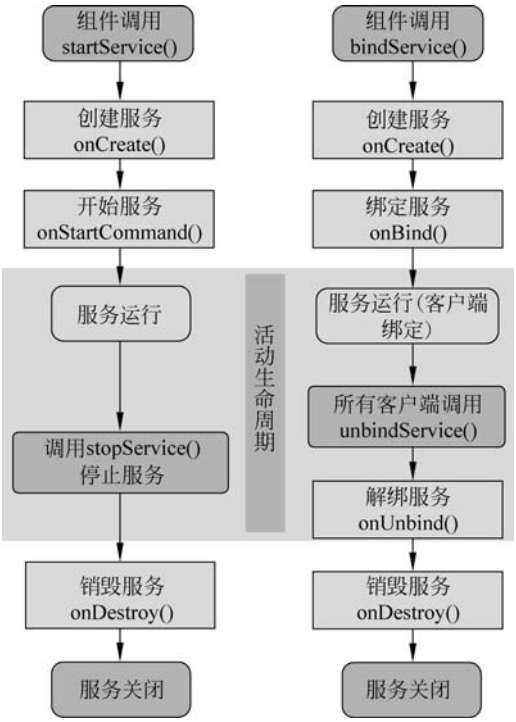

图 5-16 Service 生命周期

将 Service 生命周期中对应方法放入 MyService.java 中。

【MyService.java】

```
01  public class MyService extends Service
02  {
03   /**
04     * 在 MyBinder 直接继承 Binder 而不是 IBinder,因为 Binder 实现了 IBinder 接口,使用更方便。
05     */
06   public class MyBinder extends Binder
07   {
08         //定义加法
09         public int add(int a, int b)
10         {
11               return a + b;
12         }
13   }
14
15   public MyBinder myBinder;            //定义公共成员变量
16
17   @Override
18   public void onCreate()
19   {
20         myBinder = new MyBinder();  //生成实例
21         com.xiaj.FirstActivity.textView1.append("Service 的 onCreate 方法\n");
22         super.onCreate();
23   }
24
25   @Override
26   public IBinder onBind(Intent arg0)
27   {
28         com.xiaj.FirstActivity.textView1.append("Service 的 onBind 方法\n");
29         return myBinder;             //必须返回实例才能激活 ServiceConnection 中的回调
                                        //方法 onServiceConnected()
30   }
31
32   @Override
33   public boolean onUnbind(Intent intent)
34   {
35         com.xiaj.FirstActivity.textView1.append("Service 的 onUnbind()方法\n");
36         return super.onUnbind(intent);
37   }
38
39   @Override
40   public void onDestroy()
41   {
42         com.xiaj.FirstActivity.textView1.append("Service 的 onDestroy()方法\n");
43         super.onDestroy();
44   }
45
46   @Override
47   public int onStartCommand(Intent intent, int flags, int startId)
```

```
48   {
49         com.xiaj.FirstActivity.textView1.append("Service 的 onStartCommand()方法\n");
50         return super.onStartCommand(intent, flags, startId);
51   }
52
53   @Override
54   public void onRebind(Intent intent)
55   {
56         com.xiaj.FirstActivity.textView1.append("Service 的 onRebind()方法\n");
57         super.onRebind(intent);
58   }
59  }
```

第 6～13 行定义了继承 Binder 的 MyBinder 类,内部定义了 add()方法实现两个整数相加。其他方法都对应生命周期中的方法。

第 21 行在 Service 中要对 FirstActivity 中控件操作需加 Activity 前缀,如果有同名 Activity 还需加 package 前缀。完整名称如下:

```
package 名称 + Activity 名称 + 控件名称
```

Service 要在 AndroidManifest.xml 文件中注册才能使用。

【AndroidManifest.xml】

```
01  <?xml version = "1.0" encoding = "UTF - 8"?>
02  < manifest xmlns:android = "http://schemas.android.com/apk/res/android"
03       package = "com.xiaj">
04
05       < application
06            android:icon = "@drawable/icon"
07            android:label = "@string/app_name">
08            < activity
09                 android:name = "com.xiaj.FirstActivity"
10                 android:label = "@string/app_name">
11                 < intent - filter >
12                      < action android:name = "android.intent.action.MAIN" />
13                      < category android:name = "android.intent.category.LAUNCHER" />
14                 </intent - filter >
15            </activity >
16            < service android:name = "com.xiaj.MyService">
17                 < intent - filter >
18                      < action android:name = "com.xiaj.MY_SERVICE" />
19                 </intent - filter >
20            </service >
21
22       </application >
23  </manifest >
```

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03   public static TextView  textView1;
04
05   @Override
06   public void onCreate(Bundle savedInstanceState)
07   {
08        super.onCreate(savedInstanceState);
09        setContentView(R.layout.main);
10        Button button1 = (Button) findViewById(R.id.button1);
11        Button button2 = (Button) findViewById(R.id.button2);
12        Button button3 = (Button) findViewById(R.id.button3);
13        Button button4 = (Button) findViewById(R.id.button4);
14        textView1 = (TextView) findViewById(R.id.textView1);
15
16        Intent intent = new Intent(getApplicationContext(), MyService.class);
          //将 MyService 绑定到 intent
17            //intent.setAction(MY_SERVICE); //对新版本,service 的调用必须是显式 intent
                                              //调用,用隐式 intent 调用会出错
18
19        //启动 Service 按钮
20        button1.setOnClickListener(new View.OnClickListener()
21        {
22             @Override
23             public void onClick(View v)
24             {
25                  startService(intent); //启动 Service
26                  //第一次单击会依次调用 onCreate()和 onStart()方法,第二次单击只调用
                    //onStart()方法
27                  //并且系统只会创建 Service 的一个实例(因此停止 Service 只需要一次
                    //stopService()调用)
28             }
29        });
30        //停止 Service 按钮
31        button2.setOnClickListener(new View.OnClickListener()
32        {
33             @Override
34             public void onClick(View v)
35             {
36                  stopService(intent); //停止 Service
37             }
38        });
39
40        ServiceConnection serviceConnection = new ServiceConnection()
41        {
42             //连接对象,在正常单击按钮时不会触发以下方法
43             @Override
44             public void onServiceDisconnected(ComponentName name)
45             {
```

```
46                    textView1.append("Service 连接断开\n");
47               }
48
49               @Override
50               public void onServiceConnected(ComponentName name, IBinder service)
51               {
52                    textView1.append("Service 连接成功\n");
53                    MyService.MyBinder myBinder = (MyService.MyBinder)service;
54                    textView1.append("调用服务计算: 1 + 2 = " + myBinder.add(1, 2) + "\n");
                      //调用服务中的 add()方法
55               }
56          };
57
58          //绑定 Service 按钮
59          button3.setOnClickListener(new View.OnClickListener()
60          {
61               @Override
62               public void onClick(View v)
63               {
64                    bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
65               }
66          });
67
68          //解除绑定 Service 按钮
69          button4.setOnClickListener(new View.OnClickListener()
70          {
71               @Override
72               public void onClick(View v)
73               {
74                    try
75                    {
76                         //执行 unbindService()之后会依次调用 onUnbind()和 onDestroy()
77                         unbindService(serviceConnection); //解除绑定 Service
78                    }
79                    catch(Exception ex)
80                    {
81                         textView1.append("Service 未绑定,无法执行 unbindService 解除绑
                           定\n");
82                    }
83
84               }
85          });
86     }
87   }
```

单击 button1 按钮运行第 25 行 startService()方法启动 MyService 服务,MyService 执行图 5-16 的 onCreate()和 onStartCommand()方法。

单击 button2 按钮执行第 36 行 stopService()方法停止 MyService 服务,MyService 执行 onDestroy()方法。

单击 button3 执行图 5-16 的 onCreate()和 onBind()方法。onBind()方法中返回 myBinder 对象会激活 FirstActivity. java 的 ServiceConnection 对象的回调方法 onServiceConnected()(第 50 行),其中的形参 service 对应 MyService 中的 myBinder。在第 54 行 myBinder. add()方法就是向 Service 调用 MyBinder 类中的 add()方法,由此完成 FirstActivity 中单击 button3 调用 Service 中的 add()计算。

单击 button4 执行 onUnbind()和 onDestroy()方法。两条线的生命周期是可以交叉的,如单击 button1 再单击 button2,BindService 会跳过已经运行的 onCreate()方法,直接从 onBind()方法开始执行。

5.8 Broadcast

Broadcast(广播)分为发送者和接收者,可实现跨应用的消息传递。重启手机、闹钟、来电、接收短信等都会发出广播,通过 BroadcastReceiver 就可以接收广播并进行相应处理。

5.8.1 静态注册


视频讲解

静态注册是指在 AndroidManifest. xml 中注册广播接收器。定义一个 MyReceiver,需要添加如下标记:

```
<receiver android:name = "com.xiaj.MyReceiver">
      <intent - filter>
            <action android:name = "com.xiaj.MY_RECEIVER" />
      </intent - filter>
</receiver>
```

广播接收器定义在 MyReceiver. java 文件中。

【MyReceiver. java】

```
01 public class MyReceiver extends BroadcastReceiver
02 {
03  @Override
04  public void onReceive(Context context, Intent intent)
05  {
06        String message = "我是 MyReceiver,收到的广播为: " + intent.getStringExtra("message");
07        Toast.makeText(context, message, Toast.LENGTH_LONG).show();
08  }
09 }
```

MyReceiver 类中只有一个方法 onReceive(),当广播接收器收到广播时就运行 onReceive()方法。其中的形参 intent 包含收到的广播消息键值对,通过 getStringExtra()就可以得到对应的广播消息。

【FirstActivity. java】

```
01   public class FirstActivity extends Activity
02   {
```

```
03    @Override
04    public void onCreate(Bundle savedInstanceState)
05    {
06        super.onCreate(savedInstanceState);
07        setContentView(R.layout.main);
08        Button button1 = (Button) findViewById(R.id.button1);
09        button1.setOnClickListener(new View.OnClickListener()
10        {
11            @Override
12            public void onClick(View v)
13            {
14                Intent intent = new Intent();
15                intent.setAction("com.xiaj.MY_RECEIVER"); //通过 Action 查找 MyReceiver
16                intent.putExtra("message", "开始点名了"); //设置广播的消息
17                intent.setPackage(getPackageName());      //指定广播接收者的包名(老
                                                            //版本可以不用这条命令)
18                sendBroadcast(intent);                    //发送广播
19            }
20        });
21    }
22  }
```

第 15 行通过 setAction()方法设置要在 AndroidManifest.xml 文件中查找的接收器。

第 16 行设置要广播的消息键值对。对于早期的版本到此 Intent 设置完毕,可以直接运行第 18 行的 sendBroadcast()方法发送广播。由于发送广播的 App 实在太多,Android 的新版本 SDK 对发送广播做了限制,需要指定广播接收者的包名,第 17 行的 setPackage()方法可完成此项功能。

发送和接收广播运行结果如图 5-17 所示。

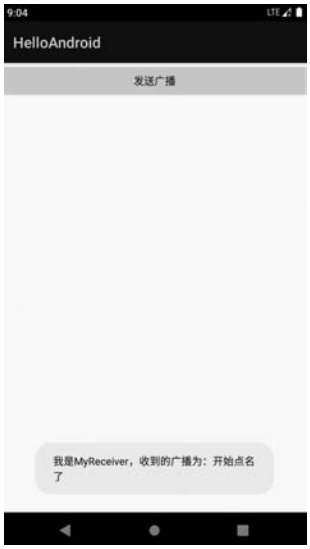

图 5-17　发送和接收广播运行结果

5.8.2 动态注册


视频讲解

动态注册广播接收器的使用方式更加灵活，可以不用在 AndroidManifest.xml 中注册，发送广播报文时也不用指定接收者的包名。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      final String MY_RECEIVER = "com.xiaj.MY_RECEIVER";
04      MyReceiver receiver; //定义一个自定义的 myReceiver 类对象
05
06      @Override
07      public void onCreate(Bundle savedInstanceState)
08      {
09          super.onCreate(savedInstanceState);
10          setContentView(R.layout.main);
11          Button button1 = (Button) findViewById(R.id.button1);
12          Button button2 = (Button) findViewById(R.id.button2);
13          Button button3 = (Button) findViewById(R.id.button3);
14          button1.setOnClickListener(new View.OnClickListener()
15          {
16              @Override
17              public void onClick(View v)
18              {
19                  Intent intent = new Intent();              //Intent 对象
20                  intent.setAction(MY_RECEIVER);             //设置 Action
21                  intent.putExtra("message", "开始点名了");  //设置广播的消息
22                  sendBroadcast(intent);                     //发送广播
23              }
24          });
25
26          button2.setOnClickListener(new View.OnClickListener()
27          {
28              @Override
29              public void onClick(View v)
30              {
31                  IntentFilter filter = new IntentFilter(MY_RECEIVER);
32                  receiver = new MyReceiver();     //初始化自定义的 MyReceiver 类
33                  registerReceiver(receiver, filter);//注册广播接收器,可以多次注册,
                                                       //意味着收到一条广播时会有多个
                                                       //Receiver 接收(注意 toast 显示的
                                                       //次数和时间)
34              }
35          });
36
37          button3.setOnClickListener(new View.OnClickListener()
38          {
39              @Override
40              public void onClick(View v)
```

```
41                  {
42                      unregisterReceiver(receiver); //注销广播接收器
43                  }
44              });
45          }
46      }
```

第 19～22 行封装 Intent 后发送广播,注意代码中不需要静态注册中设定广播接收者的包名"intent.setPackage(getPackageName());"代码。此时单击 button1 发出的广播对于接收器 MyReceiver 是收不到的,因为此时 MyReceiver 既没有静态注册,也没有动态注册。

第 26～35 行 button2 的单击监听器实现广播接收器的动态注册,关键代码是第 33 行的 registerReceiver()方法动态绑定广播接收器。如果连续单击 button2 按钮,将会注册同等数量的广播接收器,此时再单击 button1,发出的一次广播会被多个广播接收器接收,并依次弹出 Toast。

第 37～44 行 button3 的单击监听器实现注销广播接收器的功能。如果连续多次单击 button3 会出错。可对第 42 行代码设置 try-catch 捕获异常,查看出错的原因,这里就不再赘述。

视频讲解

5.8.3 多接收器接收普通广播

本案例定义两个广播接收器,当发出广播时,两个广播接收器都会收到广播报文。发送广播报文的代码与静态注册案例相同,本节就不再重复讲解。以下是定义的两个接收器类。

【FirstReceiver.java】

```
01  public class FirstReceiver extends BroadcastReceiver
02  {
03   @Override
04   public void onReceive(Context context, Intent intent)
05   {
06        String str = "FirstReceiver 接收到的广播为: " + intent.getStringExtra("message");
07        Log.i("xj", "FirstReceiver: " + str);
08   }
09  }
```

【SecondReceiver.java】

```
01  public class SecondReceiver extends BroadcastReceiver
02  {
03   @Override
04   public void onReceive(Context context, Intent intent)
05   {
06       String str = "SecondReceiver 接收到的广播为: " + intent.getStringExtra("message");
07       Log.i("xj", "SecondReceiver: " + str);
08   }
09  }
```

当 FirstActivity 中发送广播后,两个接收器都会收到广播,调用各自接收器中的

onReceive 方法，通过 Log.i()方法输出如下信息：

```
I: FirstReceiver: FirstReceiver 接收到的广播为: 开始点名了
I: SecondReceiver: SecondReceiver 接收到的广播为: 开始点名了
```

此时接收广播的顺序与广播接收器的注册先后顺序相关。

5.8.4 有序广播


视频讲解

如果发送的广播是有序广播，多个广播接收器可按指定的优先级依次接收处理报文。如果优先级高的接收器将消息处理后重新发出，后续的接收器可以同时接收处理前和处理后的广播消息。优先级高的广播接收器也可以终止后续广播接收器接收广播。

【AndroidManifest.xm】

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< manifest xmlns:android = "http://schemas.android.com/apk/res/android"
    package = "com.xiaj">

    < permission
        android:name = "myreceiver.permission.MY_BROADCAST_PERMISSION"
        android:protectionLevel = "normal" />
    < uses - permission android:name = "myreceiver.permission.MY_BROADCAST_PERMISSION" />

    < application
        android:icon = "@drawable/icon"
        android:label = "@string/app_name">
        < activity
            android:name = "com.xiaj.FirstActivity"
            android:label = "@string/app_name">
            < intent - filter >
                < action android:name = "android.intent.action.MAIN" />
                < category android:name = "android.intent.category.LAUNCHER" />
            </intent - filter >
            </activity >

            < receiver android:name = "com.xiaj.FirstReceiver">
                < intent - filter android:priority = "999">
                    < action android:name = "com.xiaj.MY_RECEIVER" />
                </intent - filter >
            </receiver >
            < receiver android:name = "com.xiaj.SecondReceiver">
                < intent - filter android:priority = "990">
                    < action android:name = "com.xiaj.MY_RECEIVER" />
                </intent - filter >
            </receiver >

    </application >
</manifest >
```

第 5～7 行的 permission 标签声明自定义权限,第 8 行的 uses-permission 标签申请自定义权限,其中的权限名称对应 FirstActivity.java 中 sendOrderedBroadcast()方法的第二个参数 receiverPermission 的值。如果没有权限则有序广播无效。

第 23～26 行和第 28～31 行定义了广播接收器及优先级,数字越大,优先级越高,将优先接收到有序广播。

FirstActivity.java 文件的代码与 5.8.3 节案例的代码基本相同,只是将发送广播的代码“sendBroadcast(intent);”改为:

```
sendOrderedBroadcast(intent, "myreceiver.permission.MY_BROADCAST_PERMISSION");
```

上述代码发送的广播即为有序广播,下面是两个接收器处理有序广播的代码。

【FirstReceiver.java】

```
01   public class FirstReceiver extends BroadcastReceiver
02   {
03    @Override
04    public void onReceive(Context context, Intent intent)
05    {
06         String str = "接收到的广播消息为: " + intent.getStringExtra("message");
07         Log.i("xj", "FirstReceiver: " + str);
08
09         //将 message 重新处理后发送出去
10         Bundle bundle = new Bundle();
11         bundle.putString("message", "此广播已被 FirstReceiver 处理过.");
12         setResultExtras(bundle);
13
14            //终止低优先级的 Receiver 接收广播,应用场景: 制作短信接收 app,阻止
              //Android 系统再接收短信
15            //abortBroadcast();
16   }
17  }
```

第 6 行是按正常方式获取广播报文。

第 10～11 行是重新封装新的广播报文,再通过第 12 行的 setResultExtras()方法将处理过的广播消息发送出去,加上 FirstActivity 中发送广播,此时就有两条广播报文了。

第 15 行如果去掉注释,将阻止后续的广播接收器接收广播,即 SecondReceive 将忽略之前发送的两条广播报文。

【SecondReciever.java】

```
01  public class SecondReceiver extends BroadcastReceiver
02  {
03   @Override
04   public void onReceive(Context context, Intent intent)
05   {
06        String str = "接收到的广播消息为: \n 已处理消息: " + getResultExtras(true).getString
```

```
            ("message") + "\n 未处理消息: " + intent.getStringExtra("message");
        //注意: 已处理消息不再是 intent.getStringExtra("message")
        Log.i("xj", "SecondReceiver: " + str);
    }
}
```

使用 getResultExtras(true).getString("message")获取处理过的广播,原始的广播报文还是用 getStringExtra()方法获取。

有序广播程序运行结果如下:

```
I: FirstReceiver:接收到的广播消息为: 来自 FirstActivity 广播的消息!
I: SecondReceiver:接收到的广播消息为:
    已处理消息: 此广播已被 FirstReceiver 处理过.
    未处理消息: 来自 FirstActivity 广播的消息!
```

如果 FirstReceiver.java 中运行"**abortBroadcast**();"命令,程序运行结果如下:

```
I: FirstReceiver:接收到的广播消息为: 来自 FirstActivity 广播的消息!
```

可以看到 SecondReceiver 无法接收任何广播了。

5.9 SQLiteDatabase

Android 内置了 SQLite 数据库,可通过 SQLiteDatabase 类对 SQLite 数据库进行增、删、改、查操作。SQLite 数据库属于文件数据库,能满足 Android 设备日常应用的数据管理需求。本案例主要对数据库和数据表的创建及增、删、改、查等基本操作做一个大致的介绍。

【FirstActivity.java】

```
public class FirstActivity extends Activity
{
    SQLiteDatabase myDB; //声明数据库
    EditText editText1, editText2, editText3;
    TextView textViewShow;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button button1 = (Button) findViewById(R.id.button1);
        Button button2 = (Button) findViewById(R.id.button2);
        Button button3 = (Button) findViewById(R.id.button3);
        editText1 = (EditText) findViewById(R.id.editTextName);
        editText2 = (EditText) findViewById(R.id.editTextScore);
```

```
18              textViewShow = (TextView) findViewById(R.id.textViewShow);
19
20              myDB = openOrCreateDatabase("mydb.db", MODE_PRIVATE, null);
                //第一次创建数据库,后续打开数据库
21              //如用 openDatabase()则数据库的路径为/data/data/com.xiaj/databases/mydb.db
22              try
23              {
24                   myDB.execSQL("CREATE TABLE Score (Id INTEGER PRIMARY KEY, Name VARCHAR
                     (30) NOT NULL DEFAULT '' COLLATE NOCASE, Score FLOAT NOT NULL DEFAULT 0,
                     InDate DATETIME NOT NULL DEFAULT(DATETIME('now', 'localtime') ) );");
                                                        //执行无返回数据的 SQL 命令
25              } catch (Exception e)
26              {
27              }
28              myDB.execSQL("delete from Score");      //清空表中数据
29
30              ContentValues cv = new ContentValues();
31              cv.put("Name", "刘备");
32              cv.put("Score", 95);
33              myDB.insert("Score", null, cv);
34
35              //使用 execSQL().注: Sqlite 中的 execSQL()一次只能执行一条 SQL 语句,解决的
                //方法使用存储过程或事务
36              try
37              {
38                   myDB.beginTransaction();
39                   myDB.execSQL("insert into Score(Name, Score) values('关羽', 80)");
40                   myDB.execSQL("insert into Score(Name, Score) values('张飞', 90)");
41                   //设置事务标志为成功,当结束事务时就会提交事务
42                   myDB.setTransactionSuccessful();
43              } catch (Exception e)
44              {
45                   e.printStackTrace();
46              } finally
47              {
48                   myDB.endTransaction();             //结束事务
49              }
50              showTable();
51              myDB.close();                           //数据库用完一定要关闭
52
53              View.OnClickListener onClickListener = new View.OnClickListener()
54              {
55                   @Override
56                   public void onClick(View v)
57                   {
58                        myDB = openOrCreateDatabase("mydb.db", MODE_PRIVATE, null);
59                        switch (v.getId())
60                        {
61                             case R.id.button1:
62                                  //mydb.execSQL("insert into Score(Name, Score) values('" +
                                    edit1.getText() + "'" + ", " + edit2.getText() + ")");
```

```
63                                  //Sqlite 数字字段时可以输入字符,SQL 语句要加入单引号
64                                  myDB.execSQL("insert into Score(Name, Score) values('" +
                                    editText1.getText() + "'" + ", '" + editText2.
                                    getText() + "')");
65                                  break;
66                              case R.id.button2:
67                                  myDB.execSQL("delete from Score where Name = '" +
                                    editText1.getText() + "'");
68                                  break;
69                              case R.id.button3:
70                                  myDB.execSQL("update Score set Score = '" + editText2.
                                    getText() + "' where Name = '" + editText1.getText() + "'");
71                                  break;
72                          }
73                          showTable();
74                          myDB.close();
75                      }
76                  };
77                  button1.setOnClickListener(onClickListener);
78                  button2.setOnClickListener(onClickListener);
79                  button3.setOnClickListener(onClickListener);
80              }
81
82              public void showTable()
83              {
84                  Cursor cursor = myDB.rawQuery("select * from Score", null);
85                  String column = "";
86                  String data = "";
87                  while (cursor.moveToNext())
88                  {
89                      for (int i = 0; i < cursor.getColumnCount(); i++)
90                      {
91                          if (cursor.isFirst())           //输出列名
92                          {
93                              column += cursor.getColumnName(i);
94                              if (i != cursor.getColumnCount() - 1)
95                              {
96                                  column += "\t\t\t";
97                              }
98                          }
99                          data += cursor.getString(i);   //输出数据
100                         if (i != cursor.getColumnCount() - 1)
101                         {
102                             data += "\t\t\t\t";
103                         }
104                     }
105                     data += "\n";
106                 }
107                 textViewShow.setText(column + "\n……
                    ……\n" + data + "\n 共计: " + cursor.getCount() + "行,
                    " + cursor.getColumnCount() + "列");
```

```
108             cursor.close();
109         }
110     }
```


SQLite 数据库运行结果如图 5-18 所示。

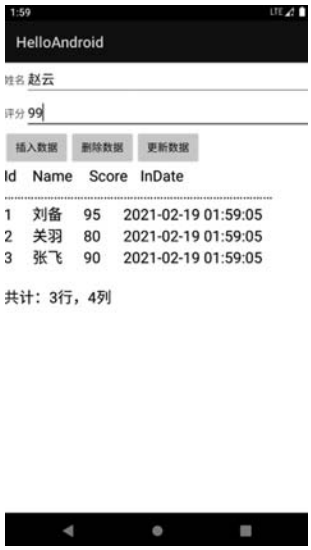

图 5-18　SQLite 数据库运行结果

第 3 行声明 SQLiteDatabase 变量 myDB,供后续对数据库进行增、删、改、查操作。

第 20 行调用 openOrCreateDatabase()方法建立或打开数据库 mydb.db。当第一次运行此命令时建立数据库,当数据库已经建立后再运行此命令则变为打开数据库。新建立的数据库所在目录是/data/data/com.xiaj/databases。当卸载 App 时,数据库和相关目录也会自动删除。

第 22～27 行是调用 execSQL()方法执行在数据库中创建数据表操作。execSQL()方法用于对数据库进行无返回数据的 SQL 操作,基本囊括了除查询语句以外的其他 SQL 命令。Score 表中的字段有常见的字符型、数字型和日期型。对于 VARCHAR 字符型字段设置 COLLATE NOCASE,意思是忽略大小写字母的区别。推荐对字段设置默认值,早期很多数据库都默认允许字段值为空值 null,主要是从节省存储空间考虑,而目前存储空间已经不是主要问题。取消 null 可简化后续编程的返回值判断。第 24 行建表命令放在 try-catch 中是为了简化判断是否已有 Score 表的逻辑,如果没有 Score 表此行命令就正常运行;如果已经有 Score 表,则捕获异常后继续执行下一条命令。

第 28 行使用 SQL 的 delete 命令删除 Score 表中的数据,保证每次演示开始时都只有后面代码添加的 3 条记录。

第 30～33 行演示用 ContentValues()加 insert()方法向 Score 表中添加数据。

第 36～49 行演示用 SQL 的 insert 命令添加两条记录。添加记录的命令是第 39～40 行,多出的代码是为了演示事务处理。execSQL()一次只能执行一条 SQL 语句,如果执行多条 SQL 语句,就需要执行多次 execSQL()方法。如果希望所有 execSQL()方法都执行成功并且提交 SQL 对数据库的修改,如果有一条 SQL 语句执行失败就全部回滚到修改前的状态。为实现上述目标,可以使用事务处理。第 38 行的 beginTransaction()方法代表事务处理开始。当执行到第 42 行时说明之前的所有 SQL 语句都没有错误,通过 setTransactionSuccessful()方法通知数据库将之前的修改全部提交。如果有异常产生就转而执行第 45 行进行异常处理。最后执行 48 行调用 endTransaction()方法结束事务。

第 50 行的 showTable()是自定义方法,用于显示 Score 表中的记录。

第 53 行开始的 OnClickListener 监听器主要处理单击按钮后数据库的增、删、改操作。需要说明的是,SQLite 数据库的数字字段也是可以保存字符的,只需要将相应 SQL 语句的字段值加上单引号括起来(不建议使用,因为后续的数据处理还需额外判断是否能转换为数字型,增加了数据处理和迁移的难度)。

第 82～109 行是自定义的 showTable()方法。使用 rawQuery()方法执行查询命令，返回 Cursor(游标)。查询的后续操作都是基于 Cursor 的方法实现，Cursor 的常用方法如表 5-1 所示。

表 5-1　Cursor 的常用方法

方　　法	作　　用
getCount()	获取满足条件的记录数
isFirst()	判断是否是第一条记录
isLast()	判断是否是最后一条记录
moveToFirst()	移动到第一条记录
moveToLast()	移动到最后一条记录
move(int offset)	移动到指定记录，正数前移，负数后移
moveToNext()	移动到下一条记录
moveToPrevious()	移动到上一条记录
getColumnName(int columnIndex)	返回 columnIndex 列的字段名
getColumnIndex(String columnName)	返回字段名为 columnName 的列索引
getColumnIndexOrThrow(String columnName)	根据字段名获得列索引，不存在的列抛出异常
getInt(int columnIndex)	获取指定列索引的 int 型值
getString(int columnIndex)	获取指定列索引的 String 型值
getCount()	返回查询结果的记录数
getColumnCount()	返回查询结果的列数


视频讲解

5.10　SQLiteOpenHelper

当 App 升级新版本时可能会对数据库的结构进行调整，如何实现 App 版本升级时都能匹配对应的数据库版本呢？SQLiteOpenHelper 就是用来处理数据库版本升级维护的。

【FirstActivity.java】

```
01   public class FirstActivity extends Activity
02   {
03        @Override
04        public void onCreate(Bundle savedInstanceState)
05        {
06             super.onCreate(savedInstanceState);
07             setContentView(R.layout.main);
08
09             DatabaseHelper databaseHelper = new DatabaseHelper(this);
10             try
11             {
12                  //当数据库降级时,此命令会抛出异常
13                  SQLiteDatabase myDb = databaseHelper.getWritableDatabase();
14             }
15             catch (Exception e)
16             {
```

```
17                  Log.i("xj", e.toString());
18              }
19              /**
20              //myDb.close();//别忘了在程序退出前关闭数据库连接
21          }
22
23          /**
24            * 用于生成或更新数据库
25            */
26           class DatabaseHelper extends SQLiteOpenHelper
27           {
28                private static final int DATABASE_VERSION = 4;
                  //数据库版本。数字变小会引起 databaseHelper.getWritableDatabase()异常
29                TextView textView1 = (TextView) findViewById(R.id.textView1);
30
31                DatabaseHelper(Context context)
32                {
33                     super(context, "mydb.db", null, DATABASE_VERSION);
34                }
35
36                //第一次创建数据库时调用
37                @Override
38                public void onCreate(SQLiteDatabase db)
39                {
40                     textView1.append("\nonCreate:新建数据库\n" + db.toString());
41                }
42
43                //升级时调用
44                @Override
45                public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
46                {
47                     textView1.append("\nonUpgrade:从版本" + oldVersion + "升级到版本 " +
                       newVersion + "\n" + db.toString());
48                }
49
50                //以下方法都要自行添加
51                //最先调用,所有情况都会调用
52                @Override
53                public void onConfigure(SQLiteDatabase db)
54                {
55                     textView1.append("\nonConfigure:");
56                     super.onConfigure(db);
57                }
58
59                //最后调用,降级时不调用。为什么
60                @Override
61                public void onOpen(SQLiteDatabase db)
62                {
63                     textView1.append("\nonOpen:");
```

```
64                  super.onOpen(db);
65              }
66
67              //降级时调用
68              @Override
69              public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)
70              {
71                  textView1.append("\nonDowngrade:从版本" + oldVersion + "降级到版本 " +
                    newVersion + "\n" + db.toString());
72                  super.onDowngrade(db, oldVersion, newVersion);
73              }
74          }
75      }
```

第 9 行将自定义的 DatabaseHelper 类实例化赋给变量 databaseHelper。

第 13 行调用 getWritableDatabase()方法[如果只读就调用 getReadableDatabase()方法]。此方法会根据版本号分别调用 DatabaseHelper 类中的方法。当版本号从大变到小时,执行此命令会抛出异常。

第 26～74 行定义了继承于 SQLiteOpenHelper 的子类 DatabaseHelper,并重写父类的方法。第 28 行定义常量 DATABASE_VERSION,当数据库有变化时,开发人员可修改源码中的此值,DatabaseHelper 会根据升级 App 前后的 DATABASE_VERSION 值变化分别调用不同的方法。SQLiteOpenHelper 版本变化和调用方法顺序关系如表 5-2 所示。

表 5-2　SQLiteOpenHelper 版本变化和调用方法顺序关系

数据库版本情况	调用方法				
	onConfigure()	onCreate()	onUpgrade()	onDowngrade()	onOpen()
第 1 次运行 App	①	②			③
版本无变化	①				②
版本升级	①		②		③
版本降级	①			②	

数据库版本升降级时会显示变动前后的数据库版本号,开发者需考虑跨版本升级问题。无论什么情况都会调用 onConfigure()方法,当降级时不调用 onOpen()方法。很多开发人员在开发 App 时只使用了 onCreate()和 onUpgrade()方法。读者可根据实际情况将代码放在相应的方法中。

5.11　数据库调试

视频讲解

在开发数据库相关应用时少不了访问数据库以获取开发相关信息。早期最常用的方式是使用 adb shell 命令(如果有多台设备就使用 adb -s emulator-5554 shell 命令,具体的设备名称可以使用 adb devices 命令获取)。操作步骤如下:

(1) 输入 adb shell 命令进入 Android 模拟器命令行界面。

(2) 输入 su 命令进入 root 模式(真实的 Android 设备要进入 root 模式以后才行)。

(3) 输入 cd /data/data/com.xiaj/databases/命令进入 mydb.db 所在目录。

(4) 输入 sqlite3 mydb.db 命令进入 mydb.db 数据库。此时提示符变为 sqlite>。

(5) 输入.table 命令可以查看数据表。注意,table 前有一个点。

(6) 输入"select * from Score;"语句可查询表记录。

输入命令级显示结果如下所示:

```
D:\> adb - s emulator - 5554 shell
generic_x86_64:/ $ su
generic_x86_64:/ # cd /data/data/com.xiaj/databases/
generic_x86_64:/data/data/com.xiaj/databases # ls
mydb.db mydb.db - journal
generic_x86_64:/data/data/com.xiaj/databases # sqlite3 mydb.db
SQLite version 3.22.0 2018 - 12 - 19 01:30:22
Enter ".help" for usage hints.
sqlite> .table
Score                  android_metadata
sqlite> .schema
CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE Score (Id INTEGER PRIMARY KEY, Name VARCHAR (30) NOT NULL DEFAULT '' COLLATE NOCASE,
Score FLOAT NOT NULL DEFAULT 0, InDate DATETIME NOT NULL DEFAULT(DATETIME('now', 'localtime') ) );
sqlite> sqlite> select * from Score;
1|刘备|95.0|2021 - 02 - 19 09:11:48
2|关羽|80.0|2021 - 02 - 19 09:11:48
3|张飞|90.0|2021 - 02 - 19 09:11:48
sqlite>
```

可以输入 inert、delete、update 命令对数据表中的记录执行增、删、改操作。但此种方法有两个缺点:

(1) 进入/data/data 目录需要 root 权限。真实的 Android 设备需要刷机才能获取 root 权限,但也意味着失去厂商保修资格。

(2) 当输入的 SQL 命令中包含中文时会显示为乱码(早期版本的 Windows 10 还需先运行 chcp 65001 命令才能在查询结果中显示中文,最新版本的 Windows 10 无须再输此命令)。

简单的替代方法是开发人员在 App 中单独开发一个 Activity,输入预置的 SQL 语句来查询数据。但如果面对的是复杂多样的查询或数据表结构改变操作,此方法有点力不从心。此时各式各样的插件应运而生,有的插件在 Android 应用中提供 Web 服务,开发人员可以通过浏览器访问数据库,此类插件的实时交互性有待改善。有的插件在 Android 中提供中转服务,再由客户端提供图形化实时交互操作界面,典型的代表就是 SQLiteStudio。以下以 SQLiteStudio 方式讲解实时管理 Android 设备上的 SQLite 数据库。

打开 SQLiteStudio 应用程序,选择"工具"→"打开配置对话框"菜单,如图 5-19 所示。

在弹出的配置对话框中选择"插件"选项,选中 Android SQLite 复选框,如图 5-20 所示,单击 OK 按钮。

此时"工具"菜单中多出一项 Get Android connector JAR file,如图 5-21 所示。

单击此菜单项下载 SQLiteStudioRemote.jar 文件。

图 5-19　选择“工具”→“打开配置对话框”菜单

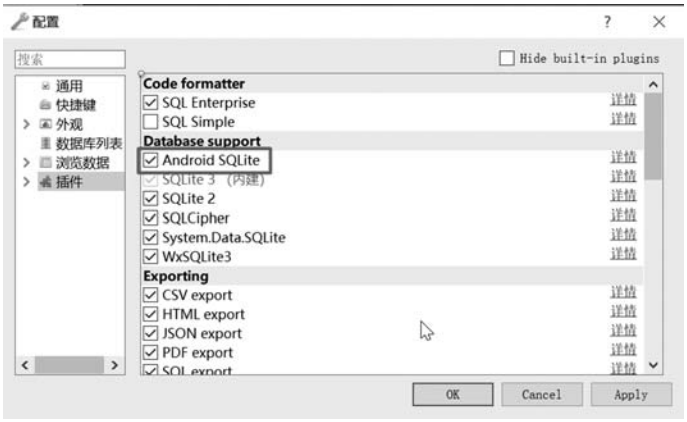

图 5-20　选中 Android SQLite 复选框

图 5-21　下载.jar 文件

将下载的 SQLiteStudioRemote.jar 文件放到 Android 项目的 libs 目录下(如果没有就新建 libs 目录)。在 app 目录下的 build.gradle 文件中添加如下配置(具体路径视文件所在路径而定):

```
implementation files('src/main/libs/SQLiteStudioRemote.jar')
```

在 Activity 的 onCreate()方法中添加以下代码:

```
SQLiteStudioService.instance().start(this);
```

此时运行 Android 项目,SQLiteStudio 的服务就同步启动。

最后配置 SQLiteStudio 连接 App 的数据库。在 SQLiteStudio 软件中选择“添加数据

库”菜单,在“数据库”对话框中选择数据类型为 Android SQLite。单击 Android database URL 图标,在弹出的对话框中选中 USB cable-port forwarding 单选按钮,如图 5-22 所示,单击 OK 按钮。

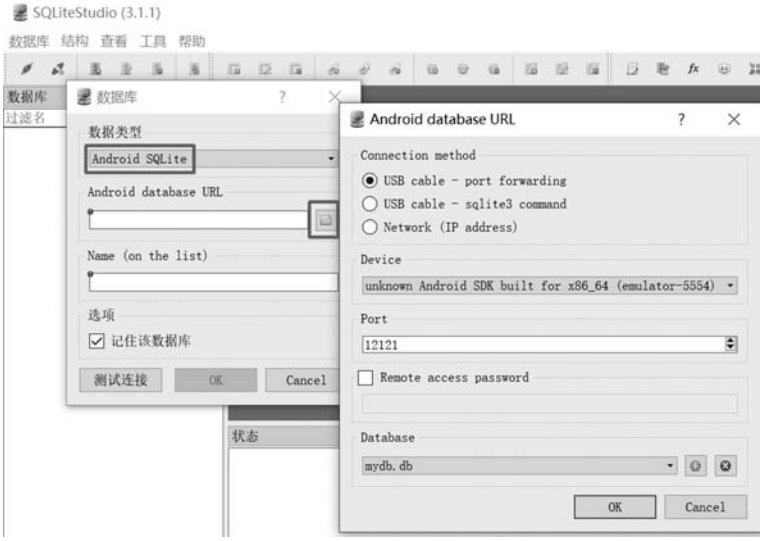

图 5-22　配置数据库连接

连接数据库成功后就可以实时查看、修改 SQLite 数据库,如图 5-23 所示。

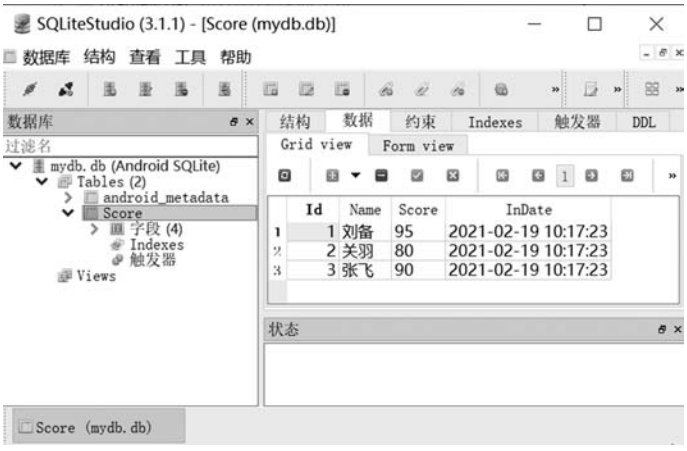

图 5-23　SQLiteStudio 运行界面

SQLiteStudio 软件的数据库管理功能非常完善,完全能满足开发要求,连接真实的 Android 设备也没有问题。

Android Studio 从 4.1 版本开始引入 Database Inspector 功能,可以连接 API level 26 以上版本的 SQLite 数据库连接。选择 View→Tool Windows→Database Inspector 菜单,在下方出现 Database Inspector 选项卡,运行 App 程序,出现如图 5-24 所示的界面。界面左边显示数据库及所包含的表。单击 按钮,界面右侧出现查询选项卡,在文本框中输入 SQL 语句,单击 Run 按钮执行相应的 SQL 命令。数据的变更可以实时显现(单击 Run 按

钮可实时查询表记录。如果选中图 5-24 的 Live updates 复选框，App 中对表数据的修改会自动更新显示)。此方案是目前连接数据库并实现实时查询操作的最简单方案。但功能上与 SQLiteStudio 方案相比还有较大的差距(如在图形化界面中修改表结构等)。

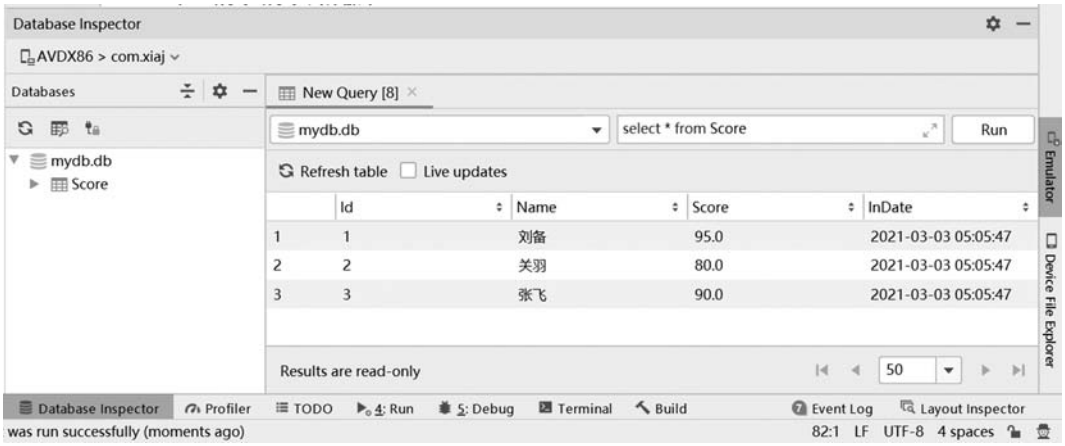

图 5-24　Database Inspector

【注】 目前的 Android Studio 版本使用 Database Inspector 功能时，Activity 中的 onCreate()方法中最好保持数据库连接打开。如果 onCreate()方法中对数据库执行了关闭操作，需要后续执行打开数据库操作才能显示出数据库和表。从实际运行看，程序启动后过几秒才连接到数据库，在 onCreate()方法中关闭数据库连接会导致 Database Inspector 无法找到数据库。

视频讲解

5.12　SharedPreferences

有时需要判断用户是第几次启动 App，如果是第一次启动，就要先显示广告或软件功能介绍界面，或者 App 试用版启动试用次数限制功能。解决上述设计需求的一种方案是开发人员自己设定一个变量记录启动次数，然后保存到 SD 卡，每次启动 App 时都判断此变量的值。此方案需要 SD 卡的读写权限，并需要添加大量代码完成 SD 卡的读写，一种更简便的方案是使用 SharedPreferences。SharedPreferences 是一个轻量级的存储类，提供简单数据类型和包装类的数据存取接口。

【FirstActivity.java】

```
01   public class FirstActivity extends Activity
02   {
03        @Override
04        protected void onCreate(Bundle savedInstanceState)
05        {
06             super.onCreate(savedInstanceState);
07             setContentView(R.layout.main);
08
09             TextView textView1 = (TextView) findViewById(R.id.textView1);
```

```
10              SharedPreferences sharedPreferences = getSharedPreferences("MyApp", Context.MODE_
                PRIVATE);
11
12              boolean isFirst = sharedPreferences.getBoolean("IsFirst", true);
13              SharedPreferences.Editor editor = sharedPreferences.edit();
14              if (!isFirst)
15              {
16                    editor.putBoolean("IsFirst", fals);
17                    editor.putString("Name", "灰太狼");
18                    editor.putInt("LoginCount", 1);
19                    editor.commit();         //保存更新
20
21                    textView1.append("这是" + sharedPreferences.getString("Name", "") +
                      "第一次光临");
22              }
23              else
24              {
25                    int loginCount = sharedPreferences.getInt("LoginCount", 1) +1;
26                    editor.putInt("LoginCount", loginCount);
27                    editor.commit();         //保存更新,否则还是 2
28                    textView1.append(sharedPreferences.getString("Name", "") + "又回来
                      了!这是第" + loginCount + "次了.");
29              }
30        }
31    }
```

第 10 行生成一个 sharedPreferences 对象实例,其存储数据区域取名 MyApp,后续保存的键值对都放在此区域。

第 12 行调用 sharedPreferences 中的 getBoolean()方法来获取键值对中名为 IsFirst 的对应值,如果查不到 IsFirst 的对应值就取第二个参数 true 作为默认值。

第 13 行调用 edit()方法,返回的对象赋予变量 editor,便于后续对 editor 的连续调用完成保存键值对操作。

第 14 行判断变量 isFirst,如果条件成立代表是第一次启动 App,程序从第 16 行开始执行,使用 putBoolean()、putString()、putInt()方法将布尔型、字符串和整型值传递给相应的 key。如果条件不成立则从第 25 行开始执行,将登录次数值执行加 1 操作后提交 editor 的修改。

第 19 行是将变更后的键值对保存到当前应用程序所属储存空间,所以即使关闭 App 或关机也不影响键值对的内容,只有卸载 App 时应用程序所属储存空间才会被删除。

5.13 精 度 问 题

以下是运行 1.2-0.1 的 Log.i()方法及输出结果:

```
Log.i("xj","1.2 - 0.1 = " + (1.2 - 0.1));
I: 1.2 - 0.1 = 1.0999999999999999
```

输出结果不是我们认为的 1.1，究其原因在于十进制数 1.2 在计算前需要转为二进制数，其中十进制的小数部分 0.2 转换为二进制小数会变为无限循环小数 $0.\dot{0}01\dot{1}$，而 Java 中双精度数是用 64b 表示的，意味着转换为二进制后会有精度丢失。本案例演示了如下两种精度处理方案。

（1）使用 DecimalFormat 设置保留小数点后位数。

（2）使用 BigDecimal 进行计算。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      @Override
04      public void onCreate(Bundle savedInstanceState)
05      {
06          super.onCreate(savedInstanceState);
07          setContentView(R.layout.main);
08
09          TextView textView1 = (TextView) findViewById(R.id.textView1);
10          Button button1 = (Button) findViewById(R.id.button1);
11          Button button2 = (Button) findViewById(R.id.button2);
12          Button button3 = (Button) findViewById(R.id.button3);
13
14          textView1.setText("各数据类型范围: ");
15          textView1.append("\nByte.MAX_VALUE:" + Byte.MAX_VALUE);
16          textView1.append("\nByte.MIN_VALUE:" + Byte.MIN_VALUE);
17          textView1.append("\nShort.MAX_VALUE:" + Short.MAX_VALUE);
18          textView1.append("\nShort.MIN_VALUE:" + Short.MIN_VALUE);
19          textView1.append("\nInteger.MAX_VALUE:" + Integer.MAX_VALUE);
20          textView1.append("\nInteger.MIN_VALUE:" + Integer.MIN_VALUE);
21          textView1.append("\nLong.MAX_VALUE:" + Long.MAX_VALUE);
22          textView1.append("\nLong.MIN_VALUE:" + Long.MIN_VALUE);
23
24          textView1.append("\nFloat.MAX_VALUE:" + Float.MAX_VALUE);
25          textView1.append("\nFloat.MIN_VALUE:" + Float.MIN_VALUE);
26          textView1.append("\nDouble.MAX_VALUE:" + Double.MAX_VALUE);
27          textView1.append("\nDouble.MIN_VALUE:" + Double.MIN_VALUE);
28
29          button1.setOnClickListener(new OnClickListener()
30          {
31              @Override
32              public void onClick(View v)
33              {
34                  textView1.setText("1.2 - 0.1 = " + (1.2 - 0.1));
35                  textView1.append("\n\n1/3 + 1/3 + 1/3 = " + (1 / 3 + 1 / 3 + 1 / 3));
36              }
37          });
38
39          button2.setOnClickListener(new OnClickListener()
40          {
41              @Override
```

```
42                  public void onClick(View v)
43                  {
44
45                      textView1.setText("DecimalFormat 方法指定的 3 位精度:1.2 - 0.1 = " +
                        decimalFormatPrecision(1.2 - 0.1));
46                      textView1.append("\n\nDecimalFormat 超出 3 位精度则四舍五入: 1.1 -
                        0.01 = " + decimalFormatPrecision(1.1 - 0.01));
47
48                      textView1.append("\n\nBigDecimal: 1.2 - 0.1 = " + sub("1.2", "0.1"));
49                      textView1.append("\n\nBigDecimal: 1.1 - 0.01 = " + sub("1.1", "0.01"));
50                      textView1.append("\n\n1.0/3.0 + 1.0/3.0 + 1.0/3.0 = " + (1.0 / 3.0 +
                        1.0 / 3.0 + 1.0 / 3.0));
51                      textView1.append("\n\nBigDecimal: 1/3 + 1/3 + 1/3 = " + (div(1, 3) +
                        div(1, 3) + div(1, 3)));
52                  }
53              });
54
55              button3.setOnClickListener(new OnClickListener()
56              {
57                  @Override
58                  public void onClick(View v)
59                  {
60                      textView1.setText("new BigDecimal(2.3) = " + new BigDecimal(2.3));
61                      textView1.append("\n\nnew BigDecimal(\"2.3\") = " + new BigDecimal
                        ("2.3"));
62                      textView1.append("\n\nBigDecimal.valueOf(2.3) = " + BigDecimal.
                        valueOf(2.3));
63                  }
64              });
65          }
66          double decimalFormatPrecision(double n)
67          {
68              //DecimalFormat 需要 API 24 以上才支持
69              DecimalFormat decimalFormat = new DecimalFormat("0.###");
70              return Double.parseDouble(decimalFormat.format(n));
71          }
72
73          BigDecimal sub(String num1, String num2)
74          {
75              BigDecimal bd1 = new BigDecimal(num1);
76              BigDecimal bd2 = new BigDecimal(num2);
77              return bd1.subtract(bd2);
78          }
79
80          double div(double num1, double num2)
81          {
82              int scale = 16; //精度 16(含)位以上即可
83              BigDecimal bd1 = new BigDecimal(Double.toString(num1));
84              BigDecimal bd2 = new BigDecimal(Double.toString(num2));
85              //return bd1.divide(bd2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
86              return bd1.divide(bd2, scale, RoundingMode.HALF_UP).doubleValue();
87          }
88      }
```

第 15～27 行显示不同数据类型的最小值和最大值。运行结果如图 5-25 所示。

第 29～37 行 button1 的单击监听器演示双精度小数减法和整型除法结果再相加计算。运行结果如图 5-26 所示。第 34 行的结果误差来源于进制转换代码的精度丢失。第 35 行的输出结果是 0,原因是 int 型的 1 除以 3 结果为取整后的 0,三个 0 相加还是 0。

第 39～53 行演示对精度的常用处理方式,如指定保留位数的四舍五入和使用 BigDecimal 类的精度计算处理。

第 45 和 46 行调用第 66～71 行的自定义方法 decimalFormatPrecision()来处理精度,其中使用 DecimalFormat 来设定数值的精度保留位数。此方案对不同小数位数的计算显得灵活性不够,会因为保留小数位数较短导致计算结果精度丢失。

第 48～49 行调用第 73～78 行的自定义方法 sub()实现 BigDecimal 的减法操作。为保证计算结果的精度,原则上将 BigDecimal 转回相应的数据类型的时机尽可能后延。

第 50 行使用 double 型的数值进行除法和加法运算。

第 60 行将 double 型的 2.3 作为 BigDecimal 的构造方法实参,其输出结果有精度丢失。

第 61 行将 String 型的 2.3 作为 BigDecimal 的构造方法实参,其输出结果没有精度丢失。

第 62 行使用 BigDecimal 的 valueOf()方法也能保证精度。

程序运行结果如图 5-25～图 5-28 所示。

图 5-25　数值范围

图 5-26　未处理精度计算结果

第 85 行的 BigDecimal.ROUND_HALF_UP 已被 Java 弃用,改为第 86 行的 RoundingMode.HALF_UP。RoundingMode 舍入含义如表 5-3 所示。RoundingMode 舍入保留个位数样例如表 5-4 所示。

图 5-27　处理精度问题

图 5-28　参数类型对精度的影响

表 5-3　RoundingMode 舍入含义

舍入关键字	舍 入 方 式
UP	向远离 0 的方向舍入
DOWN	向 0 方向舍入
CEILING	向正无穷方向舍
FLOOR	向负无穷方向舍入
HALF_UP	四舍五入
HALF_DOWN	五舍六入
HALF_EVEN	四舍六入,五留双
UNNECESSARY	计算结果是精确的,不需要舍入模式，会抛出 ArithmeticException 异常

表 5-4　RoundingMode 舍入保留个位数样例

数字	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	6	5	6	5	6	5	6	抛出 ArithmeticException 异常
2.5	3	2	3	2	3	2	2	抛出 ArithmeticException 异常
1.6	2	1	2	1	2	2	2	抛出 ArithmeticException 异常
1.1	2	1	2	1	1	1	1	抛出 ArithmeticException 异常
1	1	1	1	1	1	1	1	1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1.1	−2	−1	−1	−2	−1	−1	−1	抛出 ArithmeticException 异常
−1.6	−2	−1	−1	−2	−2	−2	−2	抛出 ArithmeticException 异常
−2.5	−3	−2	−2	−3	−3	−2	−2	抛出 ArithmeticException 异常
−5.5	−6	−5	−5	−6	−6	−5	−6	抛出 ArithmeticException 异常

5.14 横 竖 屏

有时设备上的屏幕自动旋转功能可能被用户关闭，而 App 的 Activity 可能需要横屏显示。本案例演示通过代码设置横屏或竖屏显示。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03   @Override
04   public void onCreate(Bundle savedInstanceState)
05   {
06        super.onCreate(savedInstanceState);
07        setContentView(R.layout.main);
08        Button button1 = (Button) this.findViewById(R.id.Button1);
09        Button button2 = (Button) this.findViewById(R.id.Button2);
10
11        Point point = new Point();
12        getWindowManager().getDefaultDisplay().getSize(point);
13        Log.i("xj","\n 设备分辨率为: " + point.toString());
14
15        button1.setOnClickListener(new OnClickListener()
16        {
17              public void onClick(View arg0)
18              {
19                   setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                     //横屏
20              }
21        });
22
23        button2.setOnClickListener(new OnClickListener()
24        {
25              public void onClick(View arg0)
26              {
27                   setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                     //竖屏
28              }
29        });
30   }
31
32   @Override
33    public void onConfigurationChanged(Configuration config)
34   {
35        super.onConfigurationChanged(config);
36        if (config.orientation == Configuration.ORIENTATION_LANDSCAPE)
37        {
38              Log.i("xj","\n 现在是横屏");
39        } else if (config.orientation == Configuration.ORIENTATION_PORTRAIT)
```

```
40          {
41                Log.i("xj","\n 现在是竖屏");
42          }
43          Log.i("xj", "onConfigurationChanged:" + config.toString());
44      }
45  }
```

第 11～13 行是获取屏幕分辨率并在 Logcat 中输出。

第 19 行将 App 设为横屏显示,第 27 行将 App 设为竖屏显示。

第 33～44 行重写 onConfigurationChanged()方法,其中对象变量 config 的 orientation 属性代表当前设备是处于横屏还是竖屏状态。

当运行程序时,单击两个按钮会将屏幕设为相应的横屏或竖屏,同时会调用当前 Activity 的 onCreate()方法。Logcat 中的输入如下:

```
I: 设备分辨率为: Point(1080, 1794)
I: 设备分辨率为: Point(1794, 1080)
I: 设备分辨率为: Point(1080, 1794)
```

可以看出每次改变横竖屏都会执行 onCreate()方法并执行第 13 行的 Log.i()方法输出分辨率。注意观察横竖屏时分辨率的变化。

在 AndroidManifest.xml 文件的 Activity 标签中添加以下属性:

```
android:configChanges = "orientation|screenSize"
```

再次运行程序并单击按钮,输出结果如下:

```
I: 设备分辨率为: Point(1080, 1794)
I: 现在是横屏
I: onConfigurationChanged:{1.0 310mcc260mnc [zh_CN_#Hans,en_US] ldltr sw411dp w683dp h387dp
420dpi nrml land finger qwerty/v/v - nav/h winConfig = { mBounds = Rect(0, 0 - 1920, 1080)
mAppBounds = Rect(0, 0 - 1794, 1080) mWindowingMode = fullscreen mDisplayWindowingMode =
fullscreen mActivityType = standard mAlwaysOnTop = undefined mRotation = ROTATION_90} s.3}
```

当横竖屏变化时不再调用 onCreate()方法,而是调用第 33～44 行的 onConfiguration-Changed()方法。此方式可防止重复调用 onCreate()方法导致控件数据被重新初始化。

5.15 获取 App 信息

在判断 App 是否需要升级等应用场景,首先需要获取当前 App 的版本信息。本案例演示获取 App 的信息。

【FirstActivity.java】

```
01  public class FirstActivity extends Activity
02  {
03      @Override
```

```
04        public void onCreate(Bundle savedInstanceState)
05        {
06             super.onCreate(savedInstanceState);
07             setContentView(R.layout.main);
08
09             TextView textView1 = (TextView) findViewById(R.id.textView1);
10             PackageManager packageManager1 = getPackageManager();
11
12
13             try
14             {
15                  PackageInfo packageInfo = packageManager1.getPackageInfo
                    (getPackageName(), 0);
16                  textView1.append("\nPackageName:" + packageInfo.packageName);
17                  textView1.append("\nVersionCode:" + packageInfo.versionCode);
18                  textView1.append("\nVersionName:" + packageInfo.versionName);
19                  textView1.append("\nVersionName:" + packageInfo.applicationInfo);
20                  textView1.append("\nVersionName:" + packageInfo.firstInstallTime);
21                  textView1.append("\nVersionName:" + packageInfo.lastUpdateTime);
22                  textView1.append("\nVersionName:" + packageInfo.toString());
23             } catch (NameNotFoundException e)
24                  {
25                  e.printStackTrace();
26             }
27        }
28   }
```

第 10 行通过 getPackageManager()方法获取当前 App 的 PackageManager 对象实例。

第 16～22 行获取当前 App 的包名、版本号、版本名称、第一次安装时间、上次升级时间等信息。获取 App 信息结果如图 5-29 所示。

图 5-29　获取 App 信息结果