第 2 天 Android 基础

第二章 数据存储

- 使用 Android 单元测试
- 使用 Logcat
- 存储数据到 data 目录中
- 存储数据到 sdcard 目录中
- 获取 sdcard 的存储状态
- SharedPreferences 的使用
- 生成、解析 XML 格式的数据

1.1 Android 单元测试

1.1.1 常见测试分类

一、根据是否知道源码

黑盒测试：不知道源码，是以用户的角度，从输入数据与输出数据的对应关系出发进行测试的。

白盒测试：知道源码,又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。

二、根据测试粒度分类

方法测试： FunctionTest，粒度最低，测试单个方法。

单元测试： JunitTest 方法里面会调用其他的方法。

联调测试： IntegrationTest 后台与前台集成，各模块之间的集成测试。

三、根据测试次数分类

冒烟测试：顾名思义，把设备一直测试到冒烟为止。Android 下提供给我们一种冒烟测试的功能：Monkey Test 猴子测试，在命令行输入 adb shell，进入 Linux 命令行。测试整个系统命令：monkey 1000 (事件的数量)；测试某个程序：monkey -p 包名事件的数量

压力测试： PressureTest ，给后台用的，主体向被观察者布置一定量任务和作业，借以观察个体完成任务的行为。

1.1.2 Android Junit

在 JavaSE 中我们可以使用 Junit 进行单元测试，Android 也提供了单元测试框架供我们使用，不同的是该框架可以模拟 Android 上下文环境，使用起来也稍微复杂点。

我将 Android Junit 的使用分为两种情形：

一、在已有的工程中添加单元测试功能

1. 创建一个类继承 AndroidTestCase 类。

在该类中编写测试方法即可，如果需要 Context，则可以直接在该类中调用 `getContext`（）方法获取，该 Context 是 AndroidTestCase 模拟的一个上下文。

2. 在 AndroidManifest.xml 中添加指令集和测试库

必须为当前的工程添加测试指令集（见文件 1-1）和测试库（见文件 1-2）才能使用。

【文件 1-1】 instrumentation 指令集

```
1.  <instrumentation
2.          android:name="android.test.InstrumentationTestRunner"
3.          android:targetPackage="com.itheima.android.junit" >
4.      </instrumentation>
```

【文件 1-2】 测试库

```
1.  <uses-library android:name="android.test.runner"/>
```

注意：

上面两个内容在 AndroidManifest.xml 中的位置不同。指令集位于 application 节点之外，测试库位于 application 节点之内。

指令集中的 android:targetPackage="com.itheima.android.junit" 属性值，指向的是 AndroidManifest.xml 中的包名，也可以指向其他工程的包名（这就是第二种使用情形了）。

二、创建测试工程测试其他工程

1. 创建一个测试工程

创建新工程时选择 Android Test Project 工程，如图 1-1 所示。然后点击 Next

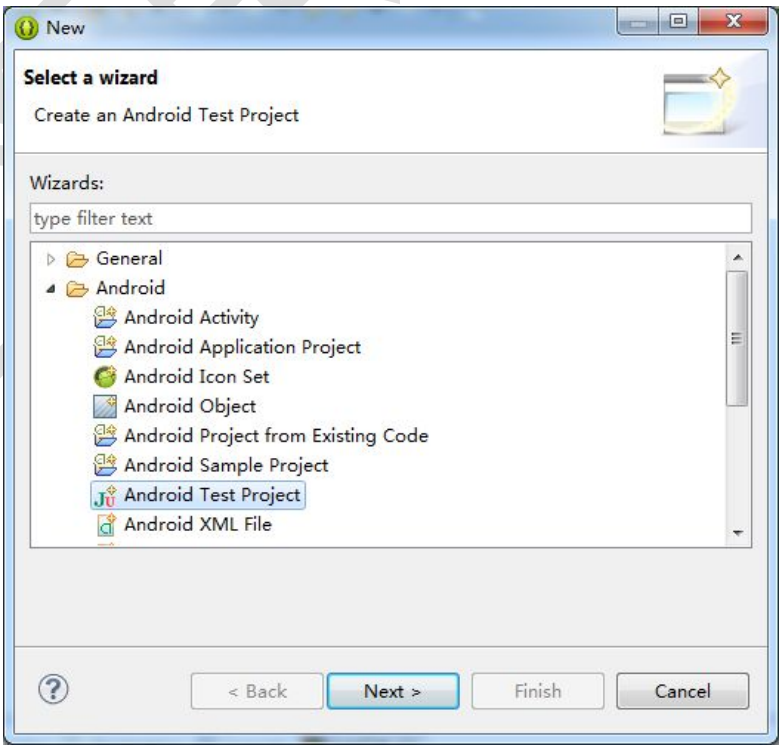

图 1-1　Android Test Project 向导界面 1

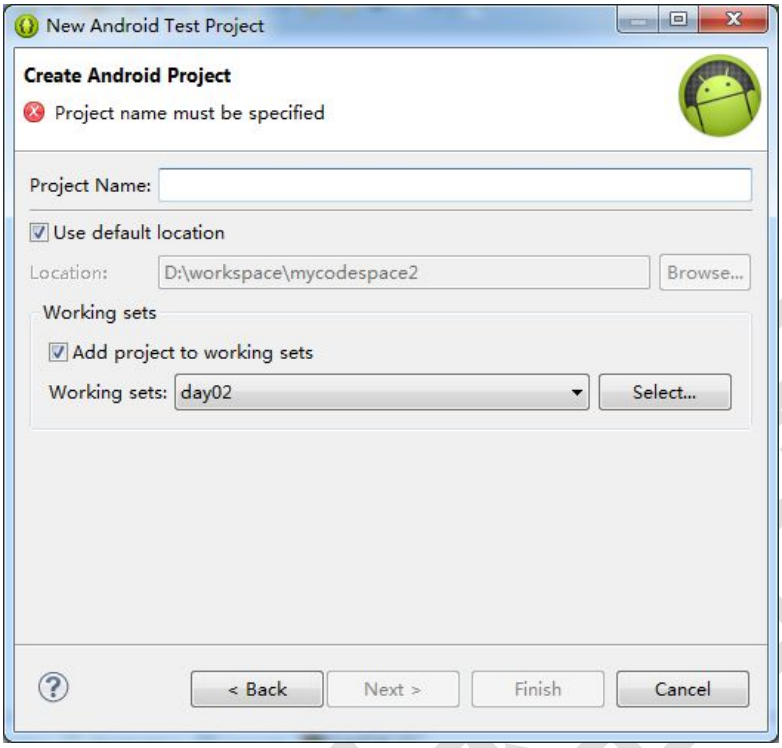

图 1-2　Android Test Project 向导界面 2

在上图中输入工程名，然后单击 Next，进入如图 1-3 界面。

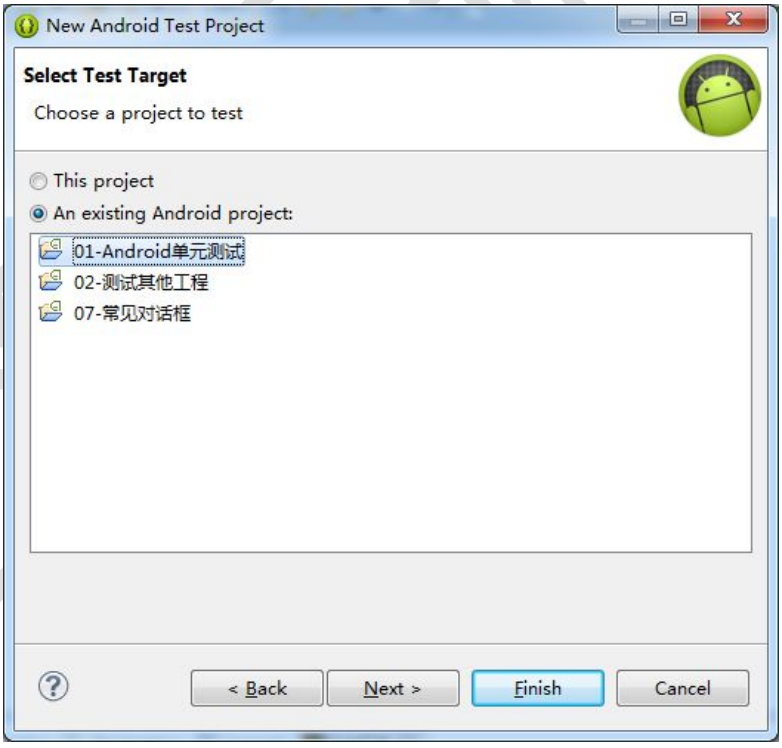

图 1-3　Android Test Project 向导界面 3

在上图界面中可以选择一个已经存在的工程，然后点击 Finish。

2. 在测试工程中编写测试类继承 AndroidTestCase
3. 在该类中编写测试方法即可

注意：

上面第二种应用情形不需要手动添加测试指令集和测试库，如果我们打开 AndroidManifest.xml 会发现，ADT 已经自动帮我们添加好。

上面第二种应用情形下，我们可以在我们测试工程中直接访问被测试工程的源码，是因为在创建该测试工程时 ADT 自动将被测试工程的源码添加到了测试工程的环境变量中。

1.2 Logcat 的使用

Android 的 Logcat 用于显示系统的调试信息，Log 是 android.util.Log 包下的类，用于日志的输出。日志总共有 5 种输出级别，从低到高分别为：verbose、debug、info、warn、error。

1.2.1 Log 日志工具类的使用

在 Android 代码的任意地方，可以直接使用 Log 工具类，该类提供 5 了种静态方法对应 5 种日志输出级别（见文件 1-3）。

【文件 1-3】 Log 工具类的使用

```
1.      Log.v(tag, msg);
2.      Log.d(tag, msg);
3.      Log.i(tag, msg);
4.      Log.w(tag, msg);
5.      Log.e(tag, msg);
```

注意：

上面方法的第一个参数可以用于日志的过滤，第二个参数是日志正文。

上面每一个方法都一个重载（Log.v(tag, msg, tr)）的形式，该方法的第三个参数是 Throwable 类，用于输出异常信息。

1.2.2 LogCat 的使用

LogCat 透视图如图 1-4 所示。

左侧区域是日志过滤器列表，All messages（no filter）是自带的无任何过滤限制。com.itheima.xxx 包名通常是某个程序运行起来后 LogCat 自动添加的过滤器，该过滤器选中后右侧只输出该应用下的日志。此外，我们还可以点击左侧绿色的加号图标去自定义一个过滤规则。

右侧区域分为上下两部分。上部分的左侧文本输入框可以输入关键字对日志进行筛选，比如 tag:Main，那么日志只显示 tag 以 Main 开头的日志。右侧的下拉框可以选择 5 种日志级别，所有的日志过滤配置会对日志的输出同时起效。

在日志输出区有如下列，Level：代表日志级别，Time：代表日志输出时间，PID 和 TID 代表线程 ID，Application：代表应用包名，Tag：代表日志 tag 标签，Text：代表日志正文。

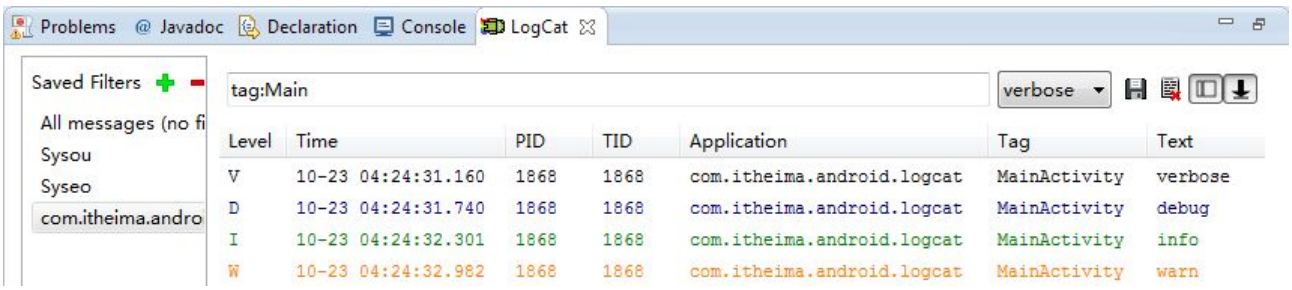

图 1-4　LogCat 界面

1.3　存储数据到 data 目录中

1.3.1　data 目录简介

当应用安装到 Android 后，系统会根据每个应用的包名创建一个/data/data/包名/的文件夹，访问自己包名下的目录是不需要权限的，并且 Android 已经提供了非常简便的 API 可以直接去访问该文件夹。

下面以一个案例来演示 data 目录的使用。

1.3.2　案例-用户登录

一、需求

如图 1-5 所示，在用户将 CheckBox 选中的前提下点击登录，就将用户名和密码保存在 data 文件夹中，在用户不勾选 CheckBox 的前提下点击登录，就不保存用户名和密码，同时删除 data 文件中的历史数据。

如果数据已经保存到了 data 目录中，那么下次用户重新启动该界面的时候要求可以将用户名和密码回显到界面。

图 1-5　用户登录界面

二、编写布局

【文件 1-4】 activity_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical">
6.
7.     <EditText
8.         android:id="@+id/et_name"
9.         android:layout_width="match_parent"
10.         android:layout_height="wrap_content"
11.         android:hint="请输入用户名"
12.         />
13.     <EditText
14.         android:id="@+id/et_pwd"
```

```
15.         android:layout_width="match_parent"
16.         android:layout_height="wrap_content"
17.         android:inputType="textPassword"
18.         android:hint="请输入密码"
19.         />
20.     <RelativeLayout
21.         android:layout_width="match_parent"
22.         android:layout_height="wrap_content"
23.         >
24.         <CheckBox
25.             android:id="@+id/cb"
26.             android:layout_width="wrap_content"
27.             android:layout_height="wrap_content"
28.             android:text="记住密码"
29.             android:layout_centerVertical="true"
30.             />
31.         <Button
32.             android:layout_alignParentRight="true"
33.             android:onClick="login"
34.             android:layout_width="wrap_content"
35.             android:layout_height="wrap_content"
36.             android:text="登录"
37.             />
38.
39.     </RelativeLayout>
40. </LinearLayout>
```

三、编写代码

【文件 1-5】 MainActivity.java

```
1. package com.itheima.android.save.data;
2.
3. import java.io.BufferedReader;
4. import java.io.File;
5. import java.io.FileInputStream;
6. import java.io.FileOutputStream;
7. import java.io.InputStreamReader;
8. import android.os.Bundle;
9. import android.app.Activity;
10. import android.text.TextUtils;
11. import android.view.View;
12. import android.widget.CheckBox;
13. import android.widget.EditText;
14. import android.widget.Toast;
15. /**
16.  *
```

```
17.  * @author wzy 2015-10-23
18.  * 将用户名和密码保存到 data 目录
19.  *
20.  */
21. public class MainActivity extends Activity {
22.
23. private EditText et_name;
24. private EditText et_pwd;
25. private CheckBox cb;
26.
27. @Override
28. protected void onCreate (Bundle savedInstanceState){
29.     super.onCreate(savedInstanceState);
30.     setContentView(R.layout.activity_main);
31.     //获取控件对象
32.     et_name = (EditText) findViewById(R.id.et_name);
33.     et_pwd = (EditText) findViewById(R.id.et_pwd);
34.     cb = (CheckBox) findViewById(R.id.cb);
35.     /*
36.      * 数据的回显
37.      * getFilesDir () 方法是父类提供的方法可以直接访问 data/data/包名/files 目录
38.      */
39.     File file = new File(getFilesDir(), "info.txt");
40.     //如果文件存在则回显数据
41.     if (file.exists()) {
42.         try {
43.             /*
44.              * openFileInput(String)是父类提供的方法，
45.              * 可以直接获取 data 目录下指定文件的的输入流
46.              */
47.             FileInputStream fis = openFileInput("info.txt");
48.             //将文件输入流转化为缓存流
49.             BufferedReader bf = new BufferedReader(new InputStreamReader(fis));
50.             //因为保存时数据只有一行，因此读取一次就可以
51.             String readLine = bf.readLine();
52.             bf.close();
53.             /*
54.              * 数据是用户名##密码的形式存储的，
55.              * 因此需要根据##对字符串进行切割
56.              */
57.             String[] split = readLine.split("##");
58.             //给 EditText 控件设置数据
59.             et_name.setText(split[0]);
60.             et_pwd.setText(split[1]);
```

```
61.             //让 CheckBox 变为勾选状态，默认不勾选
62.             cb.setChecked(true);
63.         } catch (Exception e) {
64.             e.printStackTrace();
65.         }
66.     }
67. }
68.
69. /*
70.  * 给登录按钮绑定的点击事件
71.  */
72. public void login(View view) {
73.     //从 EditText 中获取数据
74.     String name = et_name.getEditableText().toString().trim();
75.     String pwd = et_pwd.getText().toString().trim();
76.     //校验数据
77.     if (TextUtils.isEmpty(name)||TextUtils.isEmpty(pwd)) {
78.         /*
79.          * 这里使用到了 makeText 的方法的重载形式，第二个参数是 int 类型，
80.          * 其对应 values/strings.xml 资源文件中的字符串常量
81.          */
82.         Toast.makeText(this, R.string.info_txt, Toast.LENGTH_SHORT).show();
83.         return;
84.     }
85.     //获取 CheckBox 的选中状态
86.     boolean checked = cb.isChecked();
87.     //保存数据 name##pwd
88.     File file = new File(getFilesDir(), "info.txt");
89.     if (checked) {
90.         try {
91.             /*
92.              * openFileOutput（）方法是父类提供的直接打开 data 目中中指定文件的输出流，
93.              * 输出地址：/data/data/包名/files/info.txt
94.              */
95.             FileOutputStream fos = openFileOutput("info.txt", MODE_PRIVATE);
96.             fos.write((name+"##"+pwd).getBytes());
97.             fos.close();
98.
99.         } catch (Exception e) {
100.                e.printStackTrace();
101.            }
102.        }else {
103.            //如果用户没有勾选 CheckBox 则删除历史文件
104.            if (file.exists()) {
105.                boolean delete = file.delete();
```

```
106.                if (delete) {
107.                    Toast.makeText(this, "删除成功", Toast.LENGTH_SHORT).show();
108.                }else {
109.                    Toast.makeText(this, "删除失败", Toast.LENGTH_SHORT).show();
110.                }
111.
112.            }else {
113.                Toast.makeText(this,  "该文件不存在，不用删除",
Toast.LENGTH_SHORT).show();
114.            }
115.        }
116.    }
117. }
118.
```

注意：

上面使用到了如下方法，这些方法都是在 ContextWrapper 类中定义的，是 Activity 的父类，因此可以直接使用。

getFilesDir()：直接获取 /data/data/包名/files 文件

openFileInput("info.txt")：直接打开/data/data/包名/files/info.txt 文件

openFileOutput("info.txt", MODE_PRIVATE)：直接创建 /data/data/包名/files/info.txt 文件

在开发过程中凡是涉及到 data 目录操作的，很少自己去创建输入流和输出流，而应该尽量去使用 Google 已经提供的方法。

openFileOutput("info.txt", MODE_PRIVATE)方法的第二个参数是设置文件的权限，这里使用了私有常量值，也就是只能当前应用访问该文件。关于文件权限的详细介绍请看 1.3.3 节。

1.3.3 文件的权限

Android 是基于 Linux 的操作系统，因此 Android 的文件权限其实就是 Linux 的文件权限。

在 Linux 中一个文件一共有三个组别：用户、群组、其它。其中每个组包含三种权限：读 r、写 w、执行 x ，也就是说一个文件共有 9 个权限属性（如图 1-6 所示）。

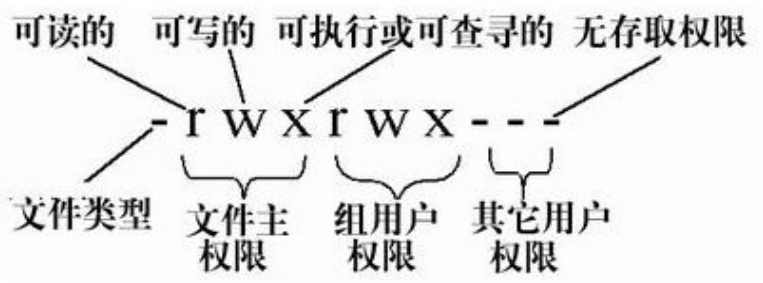

图 1-6　文件权限属性

图 1-6 中的第一个属性代表文件类型，不属于权限范畴。剩下的 9 位从左往右 1 到 3 位是［用户］，4 到 6 位是［群组］，7 到 9 位是［其它］。

举例：通过 DDMS 的 File Explorer 查看文件信息（如图 1-7），可以看到某个文件的权限为：-rwxrw-r--

他的意思就是［用户］对其享有读写执行的权限，［群组］享有读写权限，［其它］享有读权限。

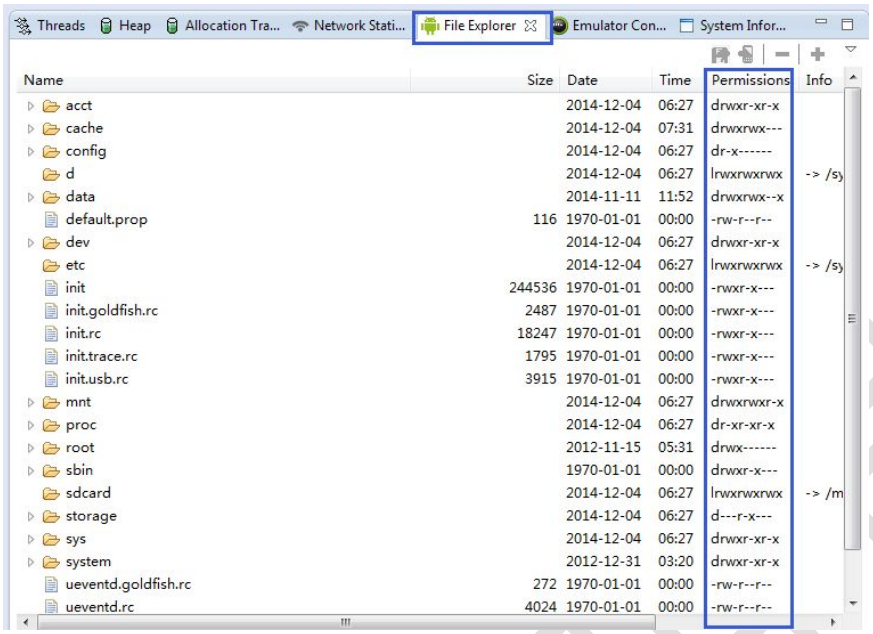

图 1-7　File Explorer 界面

1.4　存储数据到 sdcard 目录中

在本节中依然使用 1.3 中的需求，只不过将数据的存储目录改为 sdcard。因此需求和布局就不再赘述。该案例能成功运行的前提是使用的模拟器必须已经分配了 sdcard 空间。

1.4.1 编写代码

【文件 1-6】 MainActivity.java

```
1.  package com.itheima.android.save.data;
2.
3.  import java.io.BufferedReader;
4.  import java.io.File;
5.  import java.io.FileReader;
6.  import java.io.FileWriter;
7.  import com.itheima.android.save.sdcard.R;
8.  import android.os.Bundle;
9.  import android.os.Environment;
10. import android.app.Activity;
11. import android.text.TextUtils;
12. import android.view.View;
13. import android.widget.CheckBox;
14. import android.widget.EditText;
15. import android.widget.Toast;
16. /**
17.  *
```

```
18.  * @author wzy 2015-10-23
19.  * 将数据存储到 sdcard
20.  *
21.  */
22. public class MainActivity extends Activity {
23.
24.     private EditText et_name;
25.     private EditText et_pwd;
26.     private CheckBox cb;
27.
28.     @Override
29.     protected void onCreate(Bundle savedInstanceState) {
30.         super.onCreate(savedInstanceState);
31.         setContentView(R.layout.activity_main);
32.         //获取控件对象
33.         et_name = (EditText) findViewById(R.id.et_name);
34.         et_pwd = (EditText) findViewById(R.id.et_pwd);
35.         cb = (CheckBox) findViewById(R.id.cb);
36.         /*
37.          * 通过 Environment 工具类提供的方法获取 sdcard 根路径
38.          */
39.         File file = new File(Environment.getExternalStorageDirectory(), "info.txt");
40.         //实现数据的回显
41.         if (file.exists()) {
42.             try {
43.                 FileReader reader = new FileReader(file);
44.                 BufferedReader bf = new BufferedReader(reader);
45.                 String readLine = bf.readLine();
46.                 bf.close();
47.                 // 显示
48.                 String[] split = readLine.split("##");
49.                 et_name.setText(split[0]);
50.                 et_pwd.setText(split[1]);
51.                 cb.setChecked(true);
52.             } catch (Exception e) {
53.                 e.printStackTrace();
54.             }
55.         }
56.     }
57.
58.     /*
59.      * 给登录按钮绑定的点击事件
60.      */
61.     public void login(View view) {
```

```
62.         //从 EditText 中获取数据
63.         String name = et_name.getEditableText().toString().trim();
64.         String pwd = et_pwd.getText().toString().trim();
65.         // 校验数据
66.         if (TextUtils.isEmpty(name) || TextUtils.isEmpty(pwd)) {
67.             Toast.makeText(this, R.string.info_txt, Toast.LENGTH_SHORT).show();
68.             return;
69.         }
70.         // 获取 CheckBox 的选中状态
71.         boolean checked = cb.isChecked();
72.         /*
73.          * 使用 Environment 提供的静态方法获取 sdcard 根路径
74.          */
75.         File file = new File(Environment.getExternalStorageDirectory(), "info.txt");
76.         if (checked) {
77.             try {
78.                 FileWriter writer = new FileWriter(file);
79.                 writer.write(name + "##" + pwd);
80.                 writer.close();
81.
82.             } catch (Exception e) {
83.                 e.printStackTrace();
84.             }
85.         } else {
86.             // 删除数据
87.             if (file.exists()) {
88.                 boolean delete = file.delete();
89.                 if (delete) {
90.                     Toast.makeText(this, "删除成功", Toast.LENGTH_SHORT).show();
91.                 } else {
92.                     Toast.makeText(this, "删除失败", Toast.LENGTH_SHORT).show();
93.                 }
94.             } else {
95.                 Toast.makeText(this, "该文件不存在，不用删除", Toast.LENGTH_SHORT).show();
96.             }
97.         }
98.     }
99. }
100.
```

注意：

在上面的代码中使用了一个 Environment 的工具类，该类提供了一个静态方法 getExternalStorageDirectory 可以直接返回 sdcard 的根目录，因此我们在写代码的时候一定不要将该路径“写死”。

1.4.2 添加权限

该案例中我们对 sdcard 进行了读写操作，sdcard 是受保护的目录，因此需要在 AndroidManifest.xml 中声明权限。

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

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

第一个权限是读权限，第二个是写权限。如果第二个权限声明了，通常第一个权限可以不声明。

1.5 获取 sdcard 的状态

在 1.4 节中我们对 sdcard 进行了读写操作，如果用户的手机没有 sdcard 或者 sdcard 存储空间不足那么我们的程序肯定会挂掉。因此在使用 sdcard 前应该先获取 sdcard 的状态。

本节用 2 种式来获取 sdcard 的状态，程序运行界面如图 1-8 所示，左图是初始化后界面，分别点击两个按钮后效果图如右图。

图 1-8　短信发送器主界面

1.5.1 编写布局

【文件 1-7】 activity_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <TextView
8.         android:id="@+id/tv_sdcard_info1"
9.         android:layout_width="wrap_content"
10.         android:layout_height="wrap_content"
11.         android:text="方式一获取 sdcard 状态" />
12.     <TextView
```

```
13.         android:id="@+id/tv_sdcard_info2"
14.         android:layout_width="wrap_content"
15.         android:layout_height="wrap_content"
16.         android:text="方式二获取 sdcard 状态" />
17.     <Button
18.         android:layout_width="match_parent"
19.         android:layout_height="wrap_content"
20.         android:text="获取 sdcard 信息 1"
21.         android:onClick="click1"
22.         />
23.     <Button
24.         android:layout_width="match_parent"
25.         android:layout_height="wrap_content"
26.         android:text="获取 sdcard 信息 2"
27.         android:onClick="click2"
28.         />
29. </LinearLayout>
```

1.5.2 编写代码

在 MainActivity 中实现业务逻辑，如文件【1-8】所示。

【文件 1-8】 MainActivity.java

```
1. package com.itheima.android.sdcard.info;
2.
3. import java.io.File;
4. import android.os.Bundle;
5. import android.os.Environment;
6. import android.os.StatFs;
7. import android.app.Activity;
8. import android.text.format.Formatter;
9. import android.view.View;
10. import android.widget.TextView;
11. /**
12.  *
13.  * @author wzy 2015-10-23
14.  * 获取 sdcard 的状态
15.  *
16.  */
17. public class MainActivity extends Activity {
18.
19. private TextView tv_info2;
20. private TextView tv_info1;
21.
22. @Override
```

```
23. protected void onCreate(Bundle savedInstanceState) {
24.     super.onCreate(savedInstanceState);
25.     setContentView(R.layout.activity_main);
26.     tv_info1 = (TextView) findViewById(R.id.tv_sdcard_info1);
27.     tv_info2 = (TextView) findViewById(R.id.tv_sdcard_info2);
28. }
29.
30. /**
31.  * 第一种方式获取 sdcard 信息
32.  *
33.  * @param view
34.  */
35. public void click1(View view) {
36.     if (!Environment.MEDIA_MOUNTED.equals(Environment.
37. getExternalStorageState())) {
38.         // sdcard 没有挂载好
39.         tv_info1.setText("sdcard 未发现！");
40.         return;
41.     }
42.     File file = Environment.getExternalStorageDirectory();
43.     // 获取总字节数
44.     long totalSpace = file.getTotalSpace();
45.     // 获取可用字节数
46.     long freeSpace = file.getFreeSpace();
47.     // 格式化数据
48.     String totalString = Formatter.formatFileSize(this, totalSpace);
49.     String freeString = Formatter.formatFileSize(this, freeSpace);
50.     // 显示
51.     tv_info1.setText("总空间：" + totalString + "\n" + "可用空间：" + freeString);
52. }
53.
54. /**
55.  * 通过 StatFs 获取存储空间的大小
56.  *
57.  * @param view
58.  */
59. @SuppressWarnings("deprecation")
60. public void click2(View view) {
61. StatFs statFs =new StatFs(Environment.getExternalStorageDirectory().
62. getAbsolutePath());
63.     // 获取总块数*每个块的大小==总空间
64.     // long blockCountLong = statFs.getBlockCountLong();
65.     // 获取总 block 个数
66.     int blockCount = statFs.getBlockCount();
```

```
67.     // 获取可用的块个数
68.     int availableBlocks = statFs.getAvailableBlocks();
69.     // 获取 block 大小
70.     int blockSize = statFs.getBlockSize();
71.     // 计算总空间=块个数*块大小
72.     long totalSpace = blockCount * blockSize;
73.     // 计算可用空间==可用块个数*块大小
74.     long freeSpace = availableBlocks * blockSize;
75.     // 格式化数据
76.     String totalString = Formatter.formatFileSize(this, totalSpace);
77.     String freeString = Formatter.formatFileSize(this, freeSpace);
78.     // 显示
79.     tv_info2.setText("总空间：" + totalString + "\n" + "可用空间：" + freeString);
80. }
81.
82. }
83.
```

注意：

上面代码使用了如下新的 API：

Environment.getExternalStorageState()：获取 sdcard 的状态，返回值为字符串类型，常见返回值可能为：Environment.MEDIA_MOUNTED 已挂载好、Environment.MEDIA_UNMOUNTED 未挂载好。我们只需要判断返回值是否为 Environment.MEDIA_MOUNTED 即可。

Formatter.formatFileSize(Context,long):该类可以根据当前上下文将字节长度转换为用户易读的字符串文本。

StatFs 类是专门用于统计文件系统文件空间信息的工具类。

1.6 SharedPreferences 的使用

SharedPreferences 简称 sp，是 Android 平台上一个轻量级的存储类，一般应用程序都会提供“设置”或者“首选项”等这样的界面，那么这些设置就可以通过 sp 来保存。

在 Android 系统中该文件保存在：/data/data/包名 /shared_prefs 目录下。

在本节中依然使用 1.3 中的需求，只不过将数据的存储方式改为 sp。因此需求和布局就不再赘述。

1.6.1 编写代码

【文件 1-9】 MainActivity.java

```
1.  package com.itheima.android.save.data;
2.
3. import com.itheima.android.save.sp.R;
4. import android.os.Bundle;
5. import android.app.Activity;
6. import android.content.SharedPreferences;
7. import android.content.SharedPreferences.Editor;
```

```
8. import android.text.TextUtils;
9. import android.view.View;
10. import android.widget.CheckBox;
11. import android.widget.EditText;
12. import android.widget.Toast;
13.
14. /**
15.  *
16.  * @author wzy 2015-10-23
17.  * 将数据存储到 SharedPreferences 中
18.  */
19. public class MainActivity extends Activity {
20.
21. private EditText et_name;
22. private EditText et_pwd;
23. private CheckBox cb;
24. private SharedPreferences sp;
25.
26. @Override
27. protected void onCreate(Bundle savedInstanceState) {
28.     super.onCreate(savedInstanceState);
29.     setContentView(R.layout.activity_main);
30.     // 获取控件
31.     et_name = (EditText) findViewById(R.id.et_name);
32.     et_pwd = (EditText) findViewById(R.id.et_pwd);
33.     cb = (CheckBox) findViewById(R.id.cb);
34.     /*
35.      * 调用父类的方法获取 sp 对象
36.      * 第一个参数：sp 文件的名字，没有则创建
37.      * 第二个参数：文件权限
38.      */
39.     sp = getSharedPreferences("info", MODE_PRIVATE);
40.
41.     // 数据的回显
42.     // 从 sp 中获取数据
43.     String name = sp.getString("name", "");
44.     String pwd = sp.getString("pwd", "");
45.     // 给 EditText 设置数据
46.     et_name.setText(name);
47.     et_pwd.setText(pwd);
48. }
49.
50. /*
51.  * 给登录按钮绑定的点击事件
```

```
52.  */
53. public void login(View view) {
54.     // 获取 EditText 中的数据
55.     String name = et_name.getEditableText().toString().trim();
56.     String pwd = et_pwd.getText().toString().trim();
57.     // 校验数据
58.     if (TextUtils.isEmpty(name) || TextUtils.isEmpty(pwd)) {
59.         Toast.makeText(this, R.string.info_txt, Toast.LENGTH_SHORT).show();
60.         return;
61.     }
62.     // 获取 CheckBox 的选中状态
63.     boolean checked = cb.isChecked();
64.     // 保存数据 name##pwd
65.     if (checked) {
66.         try {
67.             /*
68.              * 如果想往 sp 中添加、修改、删除数据则需要通过 sp 获取到 Editor
69.              */
70.             Editor editor = sp.edit();
71.             // 设置数据
72.             editor.putString("name", name);
73.             editor.putString("pwd", pwd);
74.             // 一定要记得执行提交方法，不然前面保存的数据没有任何效果
75.             editor.commit();
76.
77.         } catch (Exception e) {
78.             e.printStackTrace();
79.         }
80.     } else {
81.         // 删除数据
82.         Editor edit = sp.edit();
83.         edit.clear();
84.         edit.commit();
85.     }
86. }
87. }
```

注意：

将数据保存到 sp 中后通过 DDMS 观测 data/data/包名/目录结构如图 1-9。发现其实 SharedPreferences 在该目录下创建了一个 shared_prefs 文件夹，在该文件夹中维护 xml 文件。

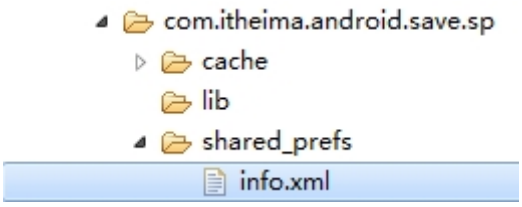

图 1-9　　sp 操作的数据目录

1.6.2 sp 的使用方法总结

一、获取 SharedPreferences 对象

```
1.  sp = getSharedPreferences("info", MODE_PRIVATE);
```

二、添加/修改数据

```
1.  /*
2.                   * 如果想往 sp 中添加、修改、删除数据则需要通过 sp 获取到 Editor
3.                   */
4.                  Editor editor = sp.edit();
5.                  // 设置数据
6.                  editor.putString("name", name);
7.                  editor.putString("pwd", pwd);
8.                  // 一定要记得执行提交方法，不然前面保存的数据没有任何效果
9.                  editor.commit();
```

三、获取数据

```
1.  /*
2.           * 从 sp 中获取数据
3.           * 第一个参数相当于 key
4.           * 第二个参数是该值如果获取不到的默认值
5.           */
6.          String name = sp.getString("name", "");
7.          String pwd = sp.getString("pwd", "");
```

四、删除数据

```
1.  // 删除数据
2.              Editor edit = sp.edit();
3.              //清空所有
4.              edit.clear();
5.              //删除 key 为 name 的数据
6.              edit.remove("name");
7.              //提交
8.              edit.commit();
```

五、sp 的连点操作

Editor 的每个方法都返回了自己本身，因此支持连点操作。将添加数据使用连点操作的方式修改后如下：

```
1. //连点操作
2.  sp.edit().putString("name",name).putString("pwd", pwd).commit();
```

1.7 Xml 格式数据的生成和解析

使用 xml 作为数据交互的载体是 Android 中非常重要的功能，比如天气预报数据、短信备份数据、通讯录数据都可以以 xml 的格式通过网络传输。

为了演示 Xml 数据的操作，我模拟了一个短信备份的案例。

需求：界面如图 1-10 所示。上面是三个 Button，前两个分别对应两种不同方式生成 xml，第三个 Button 点击后解析备份的 xml 文件，然后将数据展现在下面的 ScrollView 中。短信数据是模拟的假数据。

图 1-10　Xml 的操作

生成的 xml 格式如文件 1-10。

【文件 1-10】　xml 文件格式

```
1.  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
2.   <smses>
3.     <sms>
4.     <address>5554</address>
5.     <body>我是内容<>0</body>
6.     <time>1445595309201</time>
7.    </sms>
8.    <sms>
9.     <address>5555</address>
10.    <body>我是内容<>1</body>
11.    <time>1445595309201</time>
12.    </sms>
13.  ............
14. </smses>
```

xmlns:tools="http://sch

1.7.1 编写布局

【文件 1-11】 activity_main.xml

```
15. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
16.     xmlns:tools="http://schemas.android.com/tools"
17.     android:layout_width="match_parent"
18.     android:layout_height="match_parent"
19.     android:orientation="vertical" >
20.
21.     <Button
22.         android:layout_width="match_parent"
23.         android:layout_height="wrap_content"
24.         android:onClick="click1"
25.         android:text="生成 xml1" />
26.
27.     <Button
28.         android:layout_width="match_parent"
29.         android:layout_height="wrap_content"
30.         android:onClick="click2"
31.         android:text="生成 xml2" />
32.
33.     <Button
34.         android:layout_width="match_parent"
35.         android:layout_height="wrap_content"
36.         android:onClick="click3"
37.         android:text="解析 xml" />
38.
39.     <ScrollView
40.         android:layout_width="match_parent"
41.         android:layout_height="wrap_content" >
42.
43.         <TextView
44.             android:id="@+id/tv_sms"
45.             android:layout_width="match_parent"
46.             android:layout_height="wrap_content" />
47.     </ScrollView>
48.
49. </LinearLayout>
```

技能：

在上面布局中首次使用了 ScrollView 控件，其特点如下：

1、只能有一个子节点，但是子节点可以再包含多个子节点，也就是只能有一个孩子，但可以有多个孙子。

2、当包含的子控件高度超过 ScrollView 的高度时可以垂直滚动子控件。

1.7.2 拼接字符串方式生成 Xml 文件

【文件 1-12】 MainActivity.java 代码片段

```
1. /*
2.   * 第一种方式生成 xml
3.   */
4.  public void click1(View view) throws Exception {
5.      StringBuilder sb = new StringBuilder();
6.      sb.append("<?xml version='1.0' encoding='utf-8' standalone='yes' ?>");
7.      sb.append("<smses>");
8.      for (int i = 0; i < 50; i++) {
9.          sb.append("<sms>");
10.         sb.append("<address>");
11.         sb.append(5554 + i);
12.         sb.append("</address>");
13.         sb.append("<body>");
14.         sb.append("我是短信内容" + i);
15.         sb.append("</body>");
16.         sb.append("<time>");
17.         sb.append(new Date().getTime());
18.         sb.append("</time>");
19.         sb.append("</sms>");
20.     }
21.     sb.append("</smses>");
22.     /*
23.      * 使用系统提供的方法 获取文件输出流
24.      */
25.     FileOutputStream fos = this.openFileOutput("info.xml", MODE_PRIVATE);
26.     fos.write(sb.toString().getBytes());
27.     fos.close();
28. }
```

技能：

上面的代码虽然也可以生成 xml 文件，但是无法对特殊字符进行处理，比如如果短信内容包含“</>”符号，那么 xml 解析器就无法完成正确的解析。因此使用的前提是你确定数据内容没有特殊字符。

1.7.3 使用 XmlSerializer 生成 Xml 文件

【文件 1-13】 MainActivity.java 代码片段

```
1.  /*
2.       * 第二种方式生成 xml
3.       * 使用 Android 提供的 API
4.       */
5.      public void click2(View view) throws Exception {
```

```
6.          //在 data 目录中创建 info2.xml 文件，获取输出流
7.          FileOutputStream fos = openFileOutput("info2.xml", MODE_PRIVATE);
8.          //通过 Xml 类创建一个 Xml 序列化器
9.          XmlSerializer serializer = Xml.newSerializer();
10.         //给序列化器设置输出流和输出流编码
11.         serializer.setOutput(fos, "utf-8");
12.         /*
13.          * 让序列化器开发写入 xml 的头信息，其本质是写入内容：
14.          * "<?xml version='1.0' encoding='utf-8' standalone='yes' ?>"
15.          */
16.         serializer.startDocument("utf-8", true);
17.         /*
18.          * 开始写入 smses 标签，
19.          * 第一个参数是该标签的命名空间， 这里不需要
20.          */
21.         serializer.startTag(null, "smses");
22.         for (int i = 0; i < 50; i++) {
23.             //开始 sms 标签
24.             serializer.startTag(null, "sms");
25.             //开始 address 标签
26.             serializer.startTag(null, "address");
27.             //在 address 标签中间写入文本
28.             serializer.text("" + (5554 + i));
29.             //结束 address 标签
30.             serializer.endTag(null, "address");
31.             //开始 body 标签
32.             serializer.startTag(null, "body");
33.             //在 body 标签中间写入文本
34.             serializer.text("我是内容<>" + i);
35.             //结束 body 标签
36.             serializer.endTag(null, "body");
37.             serializer.startTag(null, "time");
38.             serializer.text(new Date().getTime()+"");
39.             serializer.endTag(null, "time");
40.             //结束 sms 标签
41.             serializer.endTag(null, "sms");
42.         }
43.         //结束 smses 标签
44.         serializer.endTag(null, "smses");
45.         //结束文档
46.         serializer.endDocument();
47.         fos.close();
48.     }
```

注意：

使用 XmlSerializer 生成 xml 文件是推荐的方式，因为该 api 内部已经实现了对特殊字符的处理。

1.7.4 使用 Pull 解析 Xml 格式数据

asserts 资源目录中的文件只能读不能写，多用于随 apk 一起发布的固定不变的数据，比如可以将国内所有的城市列表放在里面。

这里将 1.7.3 节中生成的 info2.xml 放到 asserts 目录中，然后读取、解析、展现。

【文件 1-14】 MainActivity.java 代码片段

```
1.  //使用 pull 解析 xml 数据
2.      public void click3(View v)throws Exception{
3.          /*
4.           * 将解析出来的数据封装在 Sms 中，然后保存到 ArrayList 中
5.           * Sms 是自定义的一个 JavaBean
6.           */
7.          ArrayList<Sms> smses = null;
8.          Sms sms = null;
9.          /*
10.          * 调用父类提供的 getAssets（）方法获取 AssertManager 对象
11.          */
12.         AssetManager assetManager = getAssets();
13.         //使用 assetManager 的 open 方法直接获取输入流对象
14.         InputStream inputStream = assetManager.open("info2.xml");
15.         //通过 Xml 的静态方法获取 Xml 解析器
16.         XmlPullParser parser = Xml.newPullParser();
17.         //设置输入流和编码
18.         parser.setInput(inputStream, "utf-8");
19.         //获取事件类型
20.         int event = parser.next();
21.         //如果没有解析到文档的结尾，则循环解析
22.         while(event!=XmlPullParser.END_DOCUMENT){
23.             //获取当前解析到的标签名称
24.             String tagName = parser.getName();
25.             //如果是“开始标签”事件
26.             if (event==XmlPullParser.START_TAG) {
27.                 //判断当前解析到的开始标签是哪个
28.                 if ("smses".equals(tagName)) {
29.                     smses = new ArrayList<Sms>();
30.                 }else if ("sms".equals(tagName)) {
31.                     sms = new Sms();
32.                 }else if ("address".equals(tagName)) {
33.                     sms.setAddress(parser.nextText());
34.                 }else if ("body".equals(tagName)) {
35.                     sms.setBody(parser.nextText());
36.                 }else if ("time".equals(tagName)) {
```

```
                    sms.setTime(parser.nextText());
                }
                //如果是“结束标签”事件
            }else if (event == XmlPullParser.END_TAG) {
                if ("sms".equals(tagName)) {
                    //如果是 sms 结尾，则将创建的 sms 对象添加到集合中
                    smses.add(sms);
                }
            }
            //继续获取下一个事件
            event = parser.next();
        }
        inputStream.close();
        //将数据展示到界面
        showSms(smses);

    }
    /*
     * 将短信显示到 TextView 中
     */
    private void showSms(ArrayList<Sms> smses){
        StringBuilder sb = new StringBuilder();
        for(Sms s : smses){
            sb.append(s.toString()+"\n");
        }
        tv_sms.setText(sb.toString());
    }
```

注意：

在上面的代码中我们不仅学到如何解析 xml，还学到了如何读取 aserts 目录中的数据。

1.7.5 Pull 解析和 SAX 解析对比

Pull 解析器的运行方式与 SAX 解析器相似，都属于事件驱动模式。它提供了类似的事件，如：开始元素和结束元素事件，使用 parser.next()可以进入下一个元素并触发相应事件。事件将作为数值代码被发送，因此可以使用一个 switch 对感兴趣的事件进行处理。当元素开始解析时，调用 parser.nextText()方法可以获取下一个 Text 类型元素的值。

SAX 解析器的工作方式是自动将事件推入事件处理器进行处理，因此你不能控制事件的处理主动结束；而 Pull 解析器的工作方式为允许你的应用程序代码主动从解析器中获取事件，正因为是主动获取事件，因此可以在满足了需要的条件后不再获取事件，结束解析。