第 6 章　数据存储全方案，详解持久化技术

任何一个应用程序其实说白了就是在不停地和数据打交道，我们聊 QQ、看新闻、刷微博所关心的都是里面的数据，没有数据的应用程序就变成了一个空壳子，对用户来说没有任何实际用途。那么这些数据都是从哪来的呢？现在多数的数据基本都是由用户产生的了，比如你发微博、评论新闻，其实都是在产生数据。

而我们前面章节所编写的众多例子中也有用到各种各样的数据，例如第 3 章最佳实践部分在聊天界面编写的聊天内容，第 5 章最佳实践部分在登录界面输入的账号和密码。这些数据都有一个共同点，即它们都是属于瞬时数据。那么什么是瞬时数据呢？就是指那些存储在内存当中，有可能会因为程序关闭或其他原因导致内存被回收而丢失的数据。这对于一些关键性的数据信息来说是绝对不能容忍的，谁都不希望自己刚发出去的一条微博，刷新一下就没了吧。那么怎样才能保证让一些关键性的数据不会丢失呢？这就需要用到数据持久化技术了。

6.1　持久化技术简介

数据持久化就是指将那些内存中的瞬时数据保存到存储设备中，保证即使在手机或电脑关机的情况下，这些数据仍然不会丢失。保存在内存中的数据是处于瞬时状态的，而保存在存储设备中的数据是处于持久状态的，持久化技术则是提供了一种机制可以让数据在瞬时状态和持久状态之间进行转换。

持久化技术被广泛应用于各种程序设计的领域当中，而本书中要探讨的自然是 Android 中的数据持久化技术。Android 系统中主要提供了三种方式用于简单地实现数据持久化功能，即文件存储、SharedPreference 存储以及数据库存储。当然，除了这三种方式之外，你还可以将数据保存在手机的 SD 卡中，不过使用文件、SharedPreference 或数据库来保存数据会相对更简单一些，而且比起将数据保存在 SD 卡中会更加的安全。

那么下面我就将对这三种数据持久化的方式一一进行详细的讲解。

6.2 文件存储

文件存储是 Android 中最基本的一种数据存储方式，它不对存储的内容进行任何的格式化处理，所有数据都是原封不动地保存到文件当中的，因而它比较适合用于存储一些简单的文本数据或二进制数据。如果你想使用文件存储的方式来保存一些较为复杂的文本数据，就需要定义一套自己的格式规范，这样方便于之后将数据从文件中重新解析出来。

那么首先我们就来看一看，Android 中是如何通过文件来保存数据的。

6.2.1 将数据存储到文件中

Context 类中提供了一个 openFileOutput ()方法，可以用于将数据存储到指定的文件中。这个方法接收两个参数，第一个参数是文件名，在文件创建的时候使用的就是这个名称，注意这里指定的文件名不可以包含路径，因为所有的文件都是默认存储到/data/data/<package name>/files/ 目录下的。第二个参数是文件的操作模式，主要有两种模式可选，MODE_PRIVATE 和 MODE_APPEND。其中 MODE_PRIVATE 是默认的操作模式，表示当指定同样文件名的时候，所写入的内容将会覆盖原文件中的内容，而 MODE_APPEND 则表示如果该文件已存在就往文件里面追加内容，不存在就创建新文件。其实文件的操作模式本来还有另外两种，MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE，这两种模式表示允许其他的应用程序对我们程序中的文件进行读写操作，不过由于这两种模式过于危险，很容易引起应用的安全性漏洞，现已在 Android 4.2 版本中被废弃。

openFileOutput ()方法返回的是一个 FileOutputStream 对象，得到了这个对象之后就可以使用 Java 流的方式将数据写入到文件中了。以下是一段简单的代码示例，展示了如何将一段文本内容保存到文件中：

```
public void save() {
    String data = "Data to save";
    FileOutputStream out = null;
    BufferedWriter writer = null;
    try {
        out = openFileOutput("data", Context.MODE_PRIVATE);
        writer = new BufferedWriter(new OutputStreamWriter(out));
        writer.write(data);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (writer != null) {
```

```
                writer.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

如果你已经比较熟悉 Java 流了，理解上面的代码一定轻而易举吧。这里通过 openFileOutput()方法能够得到一个 FileOutputStream 对象，然后再借助它构建出一个 OutputStreamWriter 对象，接着再使用 OutputStreamWriter 构建出一个 BufferedWriter 对象，这样你就可以通过 BufferedWriter 来将文本内容写入到文件中了。

下面我们就编写一个完整的例子，借此学习一下如何在 Android 项目中使用文件存储的技术。首先创建一个 FilePersistenceTest 项目，并修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/edit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        />

</LinearLayout>
```

这里只是在布局中加入了一个 EditText，用于输入文本内容。其实现在你就可以运行一下程序了，界面上肯定会有一个文本输入框。然后在文本输入框中随意输入点什么内容，再按下 Back 键，这时输入的内容肯定就已经丢失了，因为它只是瞬时数据，在活动被销毁后就会被回收。而这里我们要做的，就是在数据被回收之前，将它存储到文件当中。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private EditText edit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        edit = (EditText) findViewById(R.id.edit);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        String inputText = edit.getText().toString();
        save(inputText);
    }

    public void save(String inputText) {
        FileOutputStream out = null;
        BufferedWriter writer = null;
        try {
            out = openFileOutput("data", Context.MODE_PRIVATE);
            writer = new BufferedWriter(new OutputStreamWriter(out));
            writer.write(inputText);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

可以看到，首先我们在 onCreate()方法中获取了 EditText 的实例，然后重写了 onDestroy()方法，这样就可以保证在活动销毁之前一定会调用这个方法。在 onDestroy()方法中我们获取了 EditText 中输入的内容，并调用 save()方法把输入的内容存储到文件中，文件命名为 data。save()方法中的代码和之前的示例基本相同，这里就不再做解释了。现在重新运行一下程序，并在 Editext 中输入一些内容，如图 6.1 所示。

图　6.1

然后按下 Back 键关闭程序，这时我们输入的内容就已经保存到文件中了。那么如何才能证实数据确实已经保存成功了呢？我们可以借助 DDMS 的 File Explorer 来查看一下。切换到 DDMS 视图，并点击 File Explorer 切换卡，在这里进入到/data/data/com.example.filepersistencetest/files/目录下，可以看到生成了一个 data 文件，如图 6.2 所示。

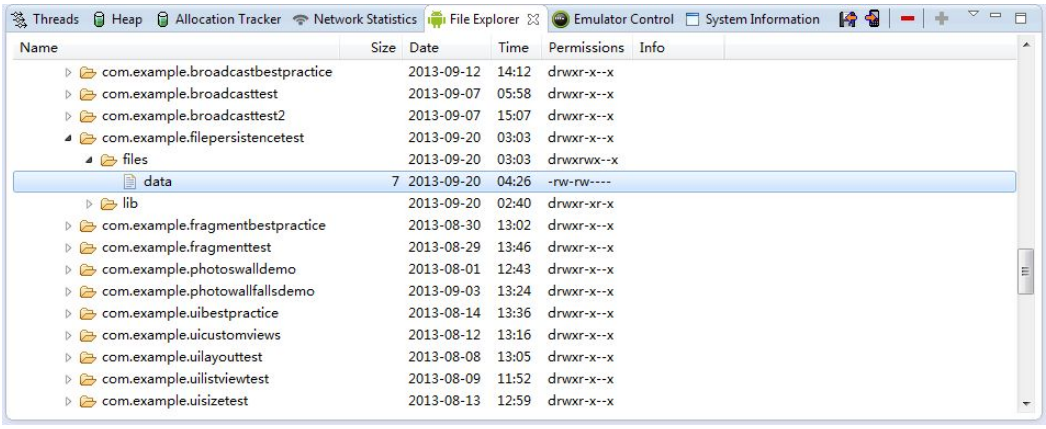

图　6.2

然后点击图 6.3 中最左边的按钮可以将这个文件导出到电脑上。

图 6.3

使用记事本打开这个文件，里面的内容如图 6.4 所示。

图 6.4

这样就证实了，在 EditText 中输入的内容确实已经成功保存到文件中了。

不过只是成功将数据保存下来还不够，我们还需要想办法在下次启动程序的时候让这些数据能够还原到 EditText 中，因此接下来我们就要学习一下，如何从文件中读取数据。

6.2.2 从文件中读取数据

类似于将数据存储到文件中，Context 类中还提供了一个 openFileInput()方法，用于从文件中读取数据。这个方法要比 openFileOutput()简单一些，它只接收一个参数，即要读取的文件名，然后系统会自动到/data/data/<package name>/files/目录下去加载这个文件，并返回一个 FileInputStream 对象，得到了这个对象之后再通过 Java 流的方式就可以将数据读取出来了。

以下是一段简单的代码示例，展示了如何从文件中读取文本数据：

```
public String load() {
    FileInputStream in = null;
    BufferedReader reader = null;
    StringBuilder content = new StringBuilder();
    try {
        in = openFileInput("data");
        reader = new BufferedReader(new InputStreamReader(in));
        String line = "";
```

```
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return content.toString();
    }
```

在这段代码中，首先通过 openFileInput()方法获取到了一个 FileInputStream 对象，然后借助它又构建出了一个 InputStreamReader 对象，接着再使用 InputStreamReader 构建出一个 BufferedReader 对象，这样我们就可以通过 BufferedReader 进行一行行地读取，把文件中所有的文本内容全部读取出来并存放在一个 StringBuilder 对象中，最后将读取到的内容返回就可以了。

了解了从文件中读取数据的方法，那么我们就来继续完善上一小节中的例子，使得重新启动程序时 EditText 中能够保留我们上次输入的内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private EditText edit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        edit = (EditText) findViewById(R.id.edit);
        String inputText = load();
        if (!TextUtils.isEmpty(inputText)) {
            edit.setText(inputText);
            edit.setSelection(inputText.length());
            Toast.makeText(this, "Restoring succeeded",
Toast.LENGTH_SHORT).show();
```

```
        }
    }
    ……
    public String load() {
        FileInputStream in = null;
        BufferedReader reader = null;
        StringBuilder content = new StringBuilder();
        try {
            in = openFileInput("data");
            reader = new BufferedReader(new InputStreamReader(in));
            String line = "";
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return content.toString();
    }

}
```

可以看到，这里的思路非常简单，在 onCreate()方法中调用 load()方法来读取文件中存储的文本内容，如果读到的内容不为空，就调用 EditText 的 setText()方法将内容填充到 EditText 里，并调用 setSelection 方法将输入光标移动到文本的末尾位置以便于继续输入，然后弹出一句还原成功的提示。load()方法中的细节我们前面已经讲过了，这里就不再赘述。

注意上述代码在对字符串进行非空判断的时候使用了 TextUtils.isEmpty()方法，这是一个非常好用的方法，它可以一次性进行两种空值的判断。当传入的字符串等于 null 或者等于空字符串的时候，这个方法都会返回 true，从而使得我们不需要单独去判断这两种空值，再使用逻辑运算符连接起来了。

现在重新运行一下程序，刚才保存的 Content 字符串肯定会被填充到 EditText 中，然后

编写一点其他的内容，比如在 EditText 中输入 Hello，接着按下 Back 键退出程序，再重新启动程序，这时刚才输入的内容并不会丢失，而是还原到了 EditText 中，如图 6.5 所示。

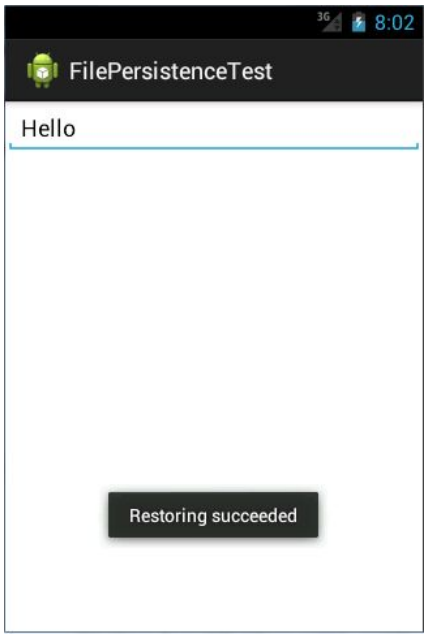

图　6.5

这样我们就已经把文件存储方面的知识学习完了，其实所用到的核心技术就是 Context 类中提供的 openFileInput()和 openFileOutput()方法，之后就是利用 Java 的各种流来进行读写操作就可以了。

不过正如我前面所说，文件存储的方式并不适合用于保存一些较为复杂的文本数据，因此，下面我们就来学习一下 Android 中另一种数据持久化的方式，它比文件存储更加简单易用，而且可以很方便地对某一指定的数据进行读写操作。

6.3　SharedPreferences 存储

不同于文件的存储方式，SharedPreferences 是使用键值对的方式来存储数据的。也就是说当保存一条数据的时候，需要给这条数据提供一个对应的键，这样在读取数据的时候就可以通过这个键把相应的值取出来。而且 SharedPreferences 还支持多种不同的数据类型存储，如果存储的数据类型是整型，那么读取出来的数据也是整型的，存储的数据是一个字符串，读取出来的数据仍然是字符串。

这样你应该就能明显地感觉到，使用 SharedPreferences 来进行数据持久化要比使用文件

方便很多，下面我们就来看一下它的具体用法吧。

6.3.1 将数据存储到 SharedPreferences 中

要想使用SharedPreferences来存储数据，首先需要获取到SharedPreferences对象。Android中主要提供了三种方法用于得到 SharedPreferences 对象。

1. Context 类中的 getSharedPreferences()方法

   此方法接收两个参数，第一个参数用于指定 SharedPreferences 文件的名称，如果指定的文件不存在则会创建一个，SharedPreferences 文件都是存放在/data/data/<package name>/shared_prefs/目录下的。第二个参数用于指定操作模式，主要有两种模式可以选择，MODE_PRIVATE 和 MODE_MULTI_PROCESS。MODE_PRIVATE 仍然是默认的操作模式，和直接传入 0 效果是相同的，表示只有当前的应用程序才可以对这个 SharedPreferences 文件进行读写。MODE_MULTI_PROCESS 则一般是用于会有多个进程中对同一个 SharedPreferences 文件进行读写的情况。类似地，MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE 这两种模式已在 Android 4.2 版本中被废弃。

2. Activity 类中的 getPreferences()方法

   这个方法和 Context 中的 getSharedPreferences()方法很相似，不过它只接收一个操作模式参数，因为使用这个方法时会自动将当前活动的类名作为 SharedPreferences 的文件名。

3. PreferenceManager 类中的 getDefaultSharedPreferences()方法

   这是一个静态方法，它接收一个 Context 参数，并自动使用当前应用程序的包名作为前缀来命名 SharedPreferences 文件。

得到了 SharedPreferences 对象之后，就可以开始向 SharedPreferences 文件中存储数据了，主要可以分为三步实现。

1. 调用 SharedPreferences 对象的 edit()方法来获取一个 SharedPreferences.Editor 对象。
2. 向 SharedPreferences.Editor 对象中添加数据，比如添加一个布尔型数据就使用 putBoolean 方法，添加一个字符串则使用 putString()方法，以此类推。
3. 调用 commit()方法将添加的数据提交，从而完成数据存储操作。

不知不觉中已经将理论知识介绍得挺多了，那我们就赶快通过一个例子来体验一下 SharedPreferences 存储的用法吧。新建一个 SharedPreferencesTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

```
    <Button
        android:id="@+id/save_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save data"
        />

</LinearLayout>
```

这里我们不做任何复杂的功能，只是简单地放置了一个按钮，用于将一些数据存储到 SharedPreferences 文件当中。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private Button saveData;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        saveData = (Button) findViewById(R.id.save_data);
        saveData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = getSharedPreferences("data",
                        MODE_PRIVATE).edit();
                editor.putString("name", "Tom");
                editor.putInt("age", 28);
                editor.putBoolean("married", false);
                editor.commit();
            }
        });
    }

}
```

可以看到，这里首先给按钮注册了一个点击事件，然后在点击事件中通过 getSharedPreferences()方法指定 SharedPreferences 的文件名为 data，并得到了 SharedPreferences.Editor 对象。接着向这个对象中添加了三条不同类型的数据，最后调用 commit()方法进行提交，从而完成了数据存储的操作。

很简单吧？现在就可以运行一下程序了，进入程序的主界面后，点击一下 Save data 按钮。这时的数据应该已经保存成功了，不过为了要证实一下，我们还是要借助 File Explorer 来进行查看。切换到 DDMS 视图，并点击 File Explorer 切换卡，然后进入到/data/data/com.example.sharedpreferencestest/shared_prefs/目录下，可以看到生成了一个 data.xml 文件，如图 6.6 所示。

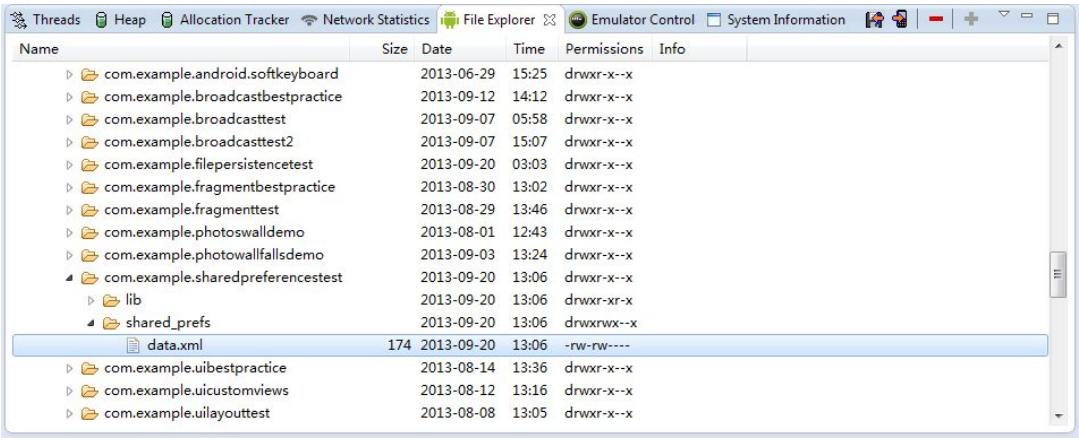

图 6.6

接下来同样是点击导出按钮将这个文件导出到电脑上，并用记事本进行查看，里面的内容如图 6.7 所示。

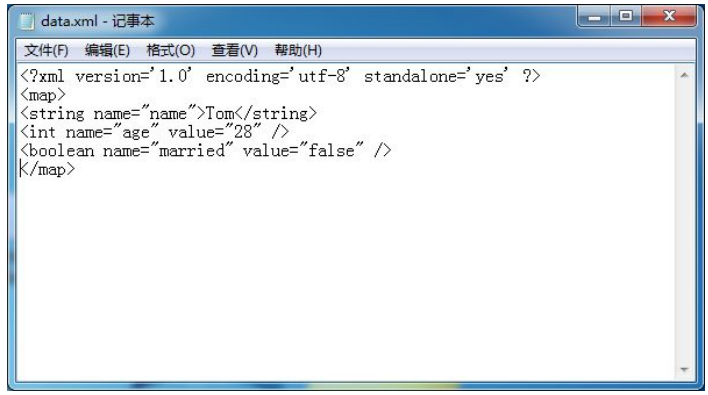

图 6.7

可以看到，我们刚刚在按钮的点击事件中添加的所有数据都已经成功保存下来了，并且 SharedPreferences 文件是使用 XML 格式来对数据进行管理的。

那么接下来我们自然要看一看，如何从 SharedPreferences 文件中去读取这些数据了。

6.3.2　从 SharedPreferences 中读取数据

你应该已经感觉到了，使用 SharedPreferences 来存储数据是非常简单的，不过下面还有更好的消息，其实从 SharedPreferences 文件中读取数据更加的简单。SharedPreferences 对象中提供了一系列的get方法用于对存储的数据进行读取，每种get方法都对应了 SharedPreferences.Editor 中的一种 put 方法，比如读取一个布尔型数据就使用 getBoolean()方法，读取一个字符串就使用 getString()方法。这些 get 方法都接收两个参数，第一个参数是键，传入存储数据时使用的键就可以得到相应的值了，第二个参数是默认值，即表示当传入的键找不到对应的值时，会以什么样的默认值进行返回。

我们还是通过例子来实际体验一下吧，仍然是在 SharedPreferencesTest 项目的基础上继续开发，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/save_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save data"
        />

    <Button
        android:id="@+id/restore_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Restore data"
        />

</LinearLayout>
```

这里增加了一个还原数据的按钮，我们希望通过点击这个按钮来从 SharedPreferences 文件中读取数据。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private Button saveData;
```

```
    private Button restoreData;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        saveData = (Button) findViewById(R.id.save_data);
        restoreData = (Button) findViewById(R.id.restore_data);
        ……
        restoreData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences pref = getSharedPreferences("data",
MODE_PRIVATE);
                String name = pref.getString("name", "");
                int age = pref.getInt("age", 0);
                boolean married = pref.getBoolean("married", false);
                Log.d("MainActivity", "name is " + name);
                Log.d("MainActivity", "age is " + age);
                Log.d("MainActivity", "married is " + married);
            }
        });
    }

}
```

可以看到，我们在还原数据按钮的点击事件中首先通过 getSharedPreferences()方法得到了 SharedPreferences 对象，然后分别调用它的 getString()、getInt()和 getBoolean()方法去获取前面所存储的姓名、年龄和是否已婚，如果没有找到相应的值就会使用方法中传入的默认值来代替，最后通过 Log 将这些值打印出来。

现在重新运行一下程序，并点击界面上的 Restore data 按钮，然后查看 LogCat 中的打印信息，如图 6.8 所示。

Tag	Text
MainActivity	name is Tom
MainActivity	age is 28
MainActivity	married is false

图 6.8

所有之前存储的数据都成功读取出来了！通过这个例子，我们就把 SharedPreferences 存储的知识也学习完了。相比之下，SharedPreferences 存储确实要比文本存储简单方便了许多，应用场景也多了不少，比如很多应用程序中的偏好设置功能其实都使用到了 SharedPreferences 技术。那么下面我们就来编写一个记住密码的功能，相信通过这个例子能够加深你对 SharedPreferences 的理解。

6.3.3　实现记住密码功能

既然是实现记住密码的功能，那么我们就不需要从头去写了，因为在上一章中的最佳实践部分已经编写过一个登录界面了，有可以重用的代码为什么不用呢？那就首先打开 BroadcastBestPractice 项目，来编辑一下登录界面的布局。修改 login.xml 中的代码，如下所示：

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1" >
   ……
    <TableRow>
        <CheckBox
            android:id="@+id/remember_pass"
            android:layout_height="wrap_content" />

        <TextView
            android:layout_height="wrap_content"
            android:text="Remember password" />
    </TableRow>

    <TableRow>
        <Button
            android:id="@+id/login"
            android:layout_height="wrap_content"
            android:layout_span="2"
            android:text="Login" />
    </TableRow>
</TableLayout>
```

这里使用到了一个新控件，CheckBox。这是一个复选框控件，用户可以通过点击的方式来进行选中和取消，我们就使用这个控件来表示用户是否需要记住密码。

然后修改 LoginActivity 中的代码，如下所示：

```
public class LoginActivity extends BaseActivity {

    private SharedPreferences pref;

    private SharedPreferences.Editor editor;

    private EditText accountEdit;

    private EditText passwordEdit;

    private Button login;

    private CheckBox rememberPass;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        pref = PreferenceManager.getDefaultSharedPreferences(this);
        accountEdit = (EditText) findViewById(R.id.account);
        passwordEdit = (EditText) findViewById(R.id.password);
        rememberPass = (CheckBox) findViewById(R.id.remember_pass);
        login = (Button) findViewById(R.id.login);
        boolean isRemember = pref.getBoolean("remember_password", false);
        if (isRemember) {
            // 将账号和密码都设置到文本框中
            String account = pref.getString("account", "");
            String password = pref.getString("password", "");
            accountEdit.setText(account);
            passwordEdit.setText(password);
            rememberPass.setChecked(true);
        }
        login.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String account = accountEdit.getText().toString();
                String password = passwordEdit.getText().toString();
                if (account.equals("admin") && password.equals("123456")) {
                    editor = pref.edit();
                    if (rememberPass.isChecked()) { // 检查复选框是否被选中
```

```
                    editor.putBoolean("remember_password", true);
                    editor.putString("account", account);
                    editor.putString("password", password);
                } else {
                    editor.clear();
                }
                editor.commit();
                Intent intent = new Intent(LoginActivity.this,
MainActivity.class);
                startActivity(intent);
                finish();
            } else {
                Toast.makeText(LoginActivity.this, "account or password
is invalid", Toast.LENGTH_SHORT).show();
            }
        }
    });
}

}
```

可以看到，这里首先在 onCreate()方法中获取到了 SharedPreferences 对象，然后调用它的 getBoolean()方法去获取 remember_password 这个键对应的值，一开始当然不存在对应的值了，所以会使用默认值 false，这样就什么都不会发生。接着在登录成功之后，会调用 CheckBox 的 isChecked()方法来检查复选框是否被选中，如果被选中了表示用户想要记住密码，这时将 remember_password 设置为 true，然后把 account 和 password 对应的值都存入到 SharedPreferences 文件当中并提交。如果没有被选中，就简单地调用一下 clear()方法，将 SharedPreferences 文件中的数据全部清除掉。

当用户选中了记住密码复选框，并成功登录一次之后，remember_password 键对应的值就是 true 了，这个时候如果再重新启动登录界面，就会从 SharedPreferences 文件中将保存的账号和密码都读取出来，并填充到文本输入框中，然后把记住密码复选框选中，这样就完成记住密码的功能了。

现在重新运行一下程序，可以看到界面上多出了一个记住密码复选框，如图 6.9 所示。

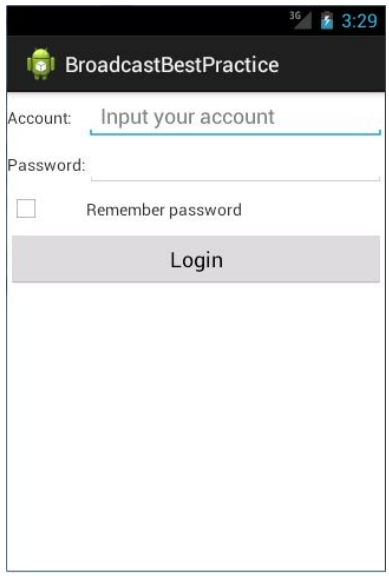

图 6.9

然后账号输入 admin，密码输入 123456，并选中记住密码复选框，点击登录，就会跳转到 MainActivity。接着在 MainActivity 中发出一条强制下线广播会让程序重新回到登录界面，此时你会发现，账号密码都已经自动填充到界面上了，如图 6.10 所示。

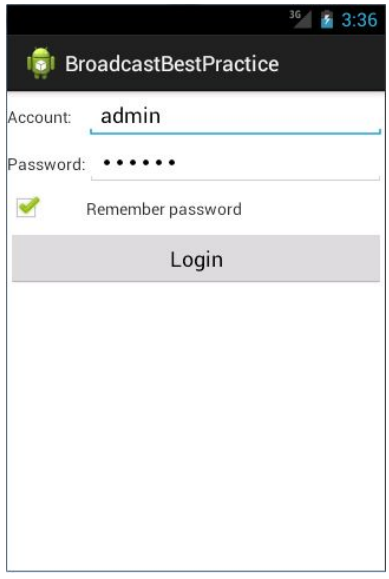

图 6.10

这样我们就使用 SharedPreferences 技术将记住密码功能成功实现了，你是不是对 SharedPreferences 理解得更加深刻了呢？

不过需要注意，这里实现的记住密码功能仍然还只是个简单的示例，并不能在实际的项目中直接使用。因为将密码以明文的形式存储在 SharedPreferences 文件中是非常不安全的，很容易就会被别人盗取，因此在正式的项目里还需要结合一定的加密算法来对密码进行保护才行。

好了，关于 SharedPreferences 的内容就讲到这里，接下来我们要学习一下本章的重头戏，Android 中的数据库技术。

经验值：+3000　　目前经验值：26905

级别：资深鸟

赢得宝物：战胜初级存储王。拾取初级存储王掉落的宝物，全新 100TB 固态硬盘一个、冥狼皮 Android 战袍一套、强力记忆提升剂一瓶。初级存储王名叫闻见，是一位武学大师，擅长轻功和存储掌，步态轻盈、敏捷，吃了一颗鄙视丸后我瞪了他两眼，发现他文科不行，换句话说智力不够。所以我要求与他比文的，我考了他两道脑筋急转弯，“一头公牛加一头母牛，猜三个字？”、“一本书放在什么地方你跨不过去？”题目很简单，但初级存储王都没有答出来，只得甘拜下风。在我告诉他答案后，他对这两道的解题思路非常震惊，我不禁再次为他的智力和前途担忧。拜别了初级存储王，我继续前进。

6.4　SQLite 数据库存储

在刚开始接触 Android 的时候，我甚至都不敢相信，Android 系统竟然是内置了数据库的！好吧，是我太孤陋寡闻了。SQLite 是一款轻量级的关系型数据库，它的运算速度非常快，占用资源很少，通常只需要几百 K 的内存就足够了，因而特别适合在移动设备上使用。SQLite 不仅支持标准的 SQL 语法，还遵循了数据库的 ACID 事务，所以只要你以前使用过其他的关系型数据库，就可以很快地上手 SQLite。而 SQLite 又比一般的数据库要简单得多，它甚至不用设置用户名和密码就可以使用。Android 正是把这个功能极为强大的数据库嵌入到了系统当中，使得本地持久化的功能有了一次质的飞跃。

前面我们所学的文件存储和 SharedPreferences 存储毕竟只适用于去保存一些简单的数据和键值对，当需要存储大量复杂的关系型数据的时候，你就会发现以上两种存储方式很难应付得了。比如我们手机的短信程序中可能会有很多个会话，每个会话中又包含了很多条信息内容，并且大部分会话还可能各自对应了电话簿中的某个联系人。很难想象如何用文件或者 SharedPreferences 来存储这些数据量大、结构性复杂的数据吧？但是使用数据库就可以做得到。那么我们就赶快来看一看，Android 中的 SQLite 数据库到底是如何使用的。

6.4.1 创建数据库

Android 为了让我们能够更加方便地管理数据库，专门提供了一个 SQLiteOpenHelper 帮助类，借助这个类就可以非常简单地对数据库进行创建和升级。既然有好东西可以直接使用，那我们自然要尝试一下了，下面我就将对 SQLiteOpenHelper 的基本用法进行介绍。

首先你要知道 SQLiteOpenHelper 是一个抽象类，这意味着如果我们想要使用它的话，就需要创建一个自己的帮助类去继承它。SQLiteOpenHelper 中有两个抽象方法，分别是 onCreate()和 onUpgrade()，我们必须在自己的帮助类里面重写这两个方法，然后分别在这两个方法中去实现创建、升级数据库的逻辑。

SQLiteOpenHelper 中还有两个非常重要的实例方法，getReadableDatabase() 和 getWritableDatabase()。这两个方法都可以创建或打开一个现有的数据库（如果数据库已存在则直接打开，否则创建一个新的数据库），并返回一个可对数据库进行读写操作的对象。不同的是，当数据库不可写入的时候（如磁盘空间已满）getReadableDatabase()方法返回的对象将以只读的方式去打开数据库，而 getWritableDatabase()方法则将出现异常。

SQLiteOpenHelper 中有两个构造方法可供重写，一般使用参数少一点的那个构造方法即可。这个构造方法中接收四个参数，第一个参数是 Context，这个没什么好说的，必须要有它才能对数据库进行操作。第二个参数是数据库名，创建数据库时使用的就是这里指定的名称。第三个参数允许我们在查询数据的时候返回一个自定义的 Cursor，一般都是传入 null。第四个参数表示当前数据库的版本号，可用于对数据库进行升级操作。构建出 SQLiteOpenHelper 的实例之后，再调用它的 getReadableDatabase()或 getWritableDatabase()方法就能够创建数据库了，数据库文件会存放在/data/data/<package name>/databases/目录下。此时，重写的 onCreate()方法也会得到执行，所以通常会在这里去处理一些创建表的逻辑。

接下来还是让我们通过例子的方式来更加直观地体会 SQLiteOpenHelper 的用法吧，首先新建一个 DatabaseTest 项目。

这里我们希望创建一个名为 BookStore.db 的数据库，然后在这个数据库中新建一张 Book 表，表中有 id（主键）、作者、价格、页数和书名等列。创建数据库表当然还是需要用建表语句的，这里也是要考验一下你的 SQL 基本功了，Book 表的建表语句如下所示：

```
create table Book (
    id integer primary key autoincrement,
    author text,
    price real,
    pages integer,
    name text)
```

只要你对 SQL 方面的知识稍微有一些了解，上面的建表语句对你来说应该都不难吧。SQLite 不像其他的数据库拥有众多繁杂的数据类型，它的数据类型很简单，integer 表示整型，

real 表示浮点型，text 表示文本类型，blob 表示二进制类型。另外，上述建表语句中我们还使用了 primary key 将 id 列设为主键，并用 autoincrement 关键字表示 id 列是自增长的。

然后需要在代码中去执行这条 SQL 语句，才能完成创建表的操作。新建 MyDatabaseHelper 类继承自 SQLiteOpenHelper，代码如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name, CursorFactory
factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        Toast.makeText(mContext, "Create succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

}
```

可以看到，我们把建表语句定义成了一个字符串常量，然后在 onCreate()方法中又调用了 SQLiteDatabase 的 execSQL()方法去执行这条建表语句，并弹出一个 Toast 提示创建成功，这样就可以保证在数据库创建完成的同时还能成功创建 Book 表。

现在修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/create_database"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Create database"
         />

</LinearLayout>
```

布局文件很简单，就是加入了一个按钮，用于创建数据库。最后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 1);
        Button createDatabase = (Button) findViewById(R.id.create_database);
        createDatabase.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                dbHelper.getWritableDatabase();
            }
        });
    }

}
```

这里我们在 onCreate()方法中构建了一个 MyDatabaseHelper 对象，并且通过构造函数的参数将数据库名指定为 BookStore.db，版本号指定为 1，然后在 Create database 按钮的点击事件里调用了 getWritableDatabase()方法。这样当第一次点击 Create database 按钮时，就会检测到当前程序中并没有 BookStore.db 这个数据库，于是会创建该数据库并调用 MyDatabaseHelper

中的 onCreate()方法，这样 Book 表也就得到了创建，然后会弹出一个 Toast 提示创建成功。再次点击 Create database 按钮时，会发现此时已经存在 BookStore.db 数据库了，因此不会再创建一次。

现在就可以运行一下代码了，在程序主界面点击 Create database 按钮，结果如图 6.11 所示。

图　6.11

此时 BookStore.db 数据库和 Book 表应该都已经创建成功了，因为当你再次点击 Create database 按钮时不会再有 Toast 弹出。可是又回到了之前的那个老问题，怎样才能证实它们的确是创建成功了？如果还是使用 File Explorer，那么最多你只能看到 databases 目录下出现了一个 BookStore.db 文件，Book 表是无法通过 File Explorer 看到的。因此这次我们准备换一种查看方式，使用 adb shell 来对数据库和表的创建情况进行检查。

adb 是 Android SDK 中自带的一个调试工具，使用这个工具可以直接对连接在电脑上的手机或模拟器进行调试操作。它存放在 sdk 的 platform-tools 目录下，如果想要在命令行中使用这个工具，就需要先把它的路径配置到环境变量里。

如果你使用的是 Windows 系统，可以右击我的电脑→属性→高级→环境变量，然后在系统变量里找到 Path 并点击编辑，将 platform-tools 目录配置进去，如图 6.12 所示。

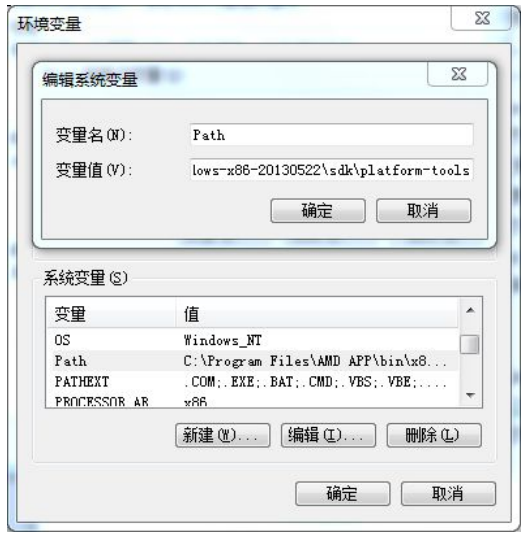

图 6.12

如果你使用的是 Linux 系统，可以在 home 路径下编辑.bash_profile 文件，将 platform-tools 目录配置进去即可，如图 6.13 所示：

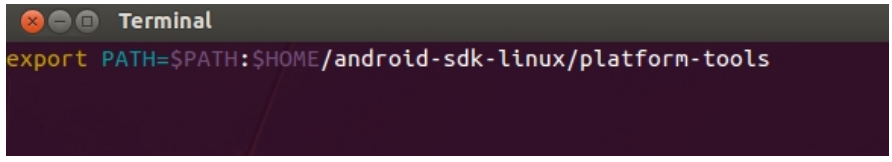

图 6.13

配置好了环境变量之后，就可以使用 adb 工具了。打开命令行界面，输入 adb shell，就会进入到设备的控制台，如图 6.14 所示。

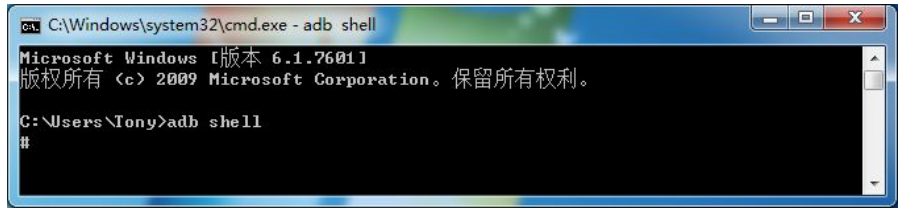

图 6.14

然后使用 cd 命令进行到/data/data/com.example.databasetest/databases/目录下，并使用 ls 命令查看到该目录里的文件，如图 6.15 所示。

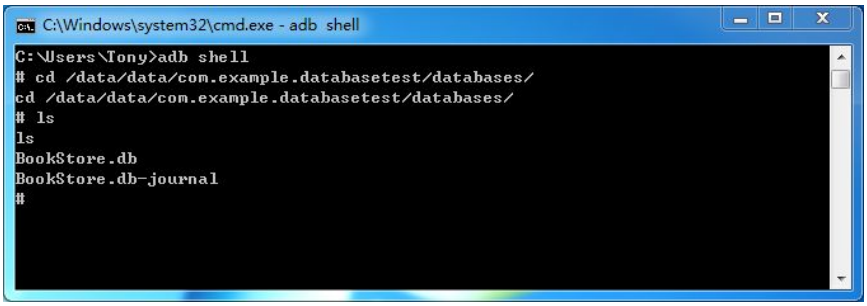

图　6.15

这个目录下出现了两个数据库文件，一个正是我们创建的 BookStore.db，而另一个 BookStore.db-journal 则是为了让数据库能够支持事务而产生的临时日志文件，通常情况下这个文件的大小都是 0 字节。

接下来我们就要借助 sqlite 命令来打开数据库了，只需要键入 sqlite3，后面加上数据库名即可，如图 6.16 所示。

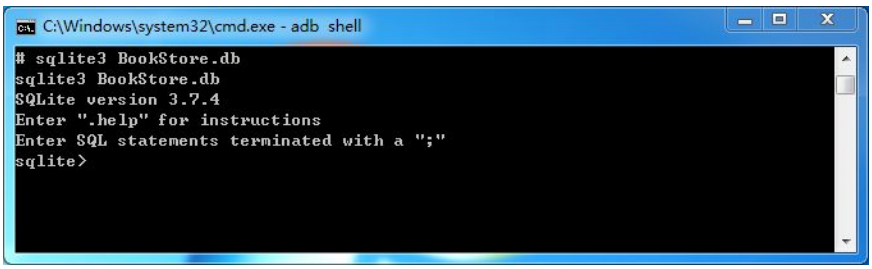

图　6.16

这时就已经打开了 BookStore.db 数据库，现在就可以对这个数据库中的表进行管理了。首先来看一下目前数据库中有哪些表，键入.table 命令，如图 6.17 所示。

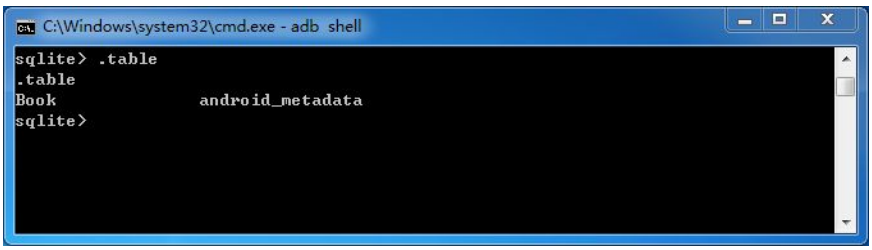

图　6.17

可以看到，此时数据库中有两张表，android_metadata 表是每个数据库中都会自动生成

的，不用管它，而另外一张 Book 表就是我们在 MyDatabaseHelper 中创建的了。这里还可以通过.schema 命令来查看它们的建表语句，如图 6.18 所示。

图 6.18

由此证明，BookStore.db 数据库和 Book 表确实已经是创建成功了。之后键入.exit 或.quit 命令可以退出数据库的编辑，再键入 exit 命令就可以退出设备控制台了。

6.4.2 升级数据库

如果你足够细心，一定会发现 MyDatabaseHelper 中还有一个空方法呢！没错，onUpgrade() 方法是用于对数据库进行升级的，它在整个数据库的管理工作当中起着非常重要的作用，可千万不能忽视它哟。

目前 DatabaseTest 项目中已经有一张 Book 表用于存放书的各种详细数据，如果我们想再添加一张 Category 表用于记录书籍的分类该怎么做呢？

比如 Category 表中有 id（主键）、分类名和分类代码这几个列，那么建表语句就可以写成：

```
create table Category (
    id integer primary key autoincrement,
    category_name text,
    category_code integer)
```

接下来我们将这条建表语句添加到 MyDatabaseHelper 中，代码如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";
```

```
    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
        Toast.makeText(mContext, "Create succeeded", Toast.LENGTH_SHORT).
show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

}
```

看上去好像都挺对的吧，现在我们重新运行一下程序，并点击 Create database 按钮，咦？竟然没有弹出创建成功的提示。当然，你也可以通过 adb 工具到数据库中再去检查一下，这样你会更加地确认，Category 表没有创建成功！

其实没有创建成功的原因不难思考，因为此时 BookStore.db 数据库已经存在了，之后不管我们怎样点击 Create database 按钮，MyDatabaseHelper 中的 onCreate()方法都不会再次执行，因此新添加的表也就无法得到创建了。

解决这个问题的办法也相当简单，只需要先将程序卸载掉，然后重新运行，这时 BookStore.db 数据库已经不存在了，如果再点击 Create database 按钮，MyDatabaseHelper 中的 onCreate()方法就会执行，这时 Category 表就可以创建成功了。

不过通过卸载程序的方式来新增一张表毫无疑问是很极端的做法，其实我们只需要巧妙地运用 SQLiteOpenHelper 的升级功能就可以很轻松地解决这个问题。修改 MyDatabaseHelper

中的代码，如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
    ……
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists Book");
        db.execSQL("drop table if exists Category");
        onCreate(db);
    }
}
```

可以看到，我们在 onUpgrade()方法中执行了两条 DROP 语句，如果发现数据库中已经存在 Book 表或 Category 表了，就将这两张表删除掉，然后再调用 onCreate()方法去重新创建。这里先将已经存在的表删除掉，是因为如果在创建表时发现这张表已经存在了，就会直接报错。

接下来的问题就是如何让 onUpgrade()方法能够执行了，还记得 SQLiteOpenHelper 的构造方法里接收的第四个参数吗？它表示当前数据库的版本号，之前我们传入的是 1，现在只要传入一个比 1 大的数，就可以让 onUpgrade()方法得到执行了。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        Button createDatabase = (Button) findViewById(R.id.create_database);
        createDatabase.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                dbHelper.getWritableDatabase();
            }
        });
    }

}
```

这里将数据库版本号指定为 2，表示我们对数据库进行升级了。现在重新运行程序，并点击 Create database 按钮，这时就会再次弹出创建成功的提示。为了验证一下 Category 表是不是已经创建成功了，我们在 adb shell 中打开 BookStore.db 数据库，然后键入.table 命令，结果如图 6.19 所示。

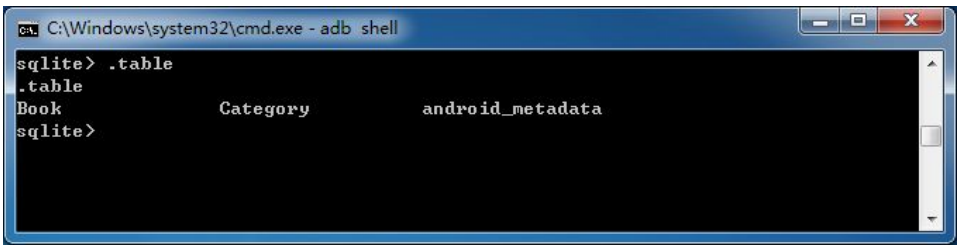

图　6.19

接着键入.schema 命令查看一下建表语句，结果如图 6.20 所示。

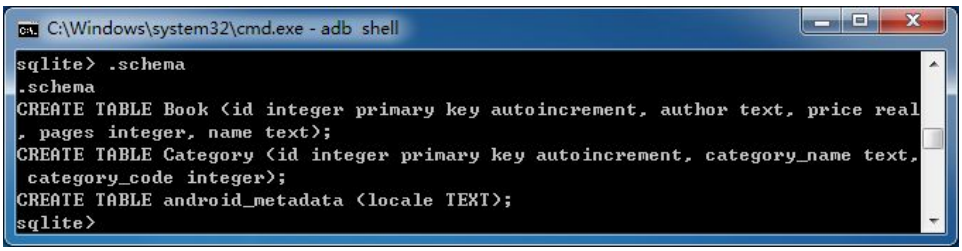

图　6.20

由此可以看出，Category 表已经创建成功了，同时也说明我们的升级功能的确起到了作用。

6.4.3　添加数据

现在你已经掌握了创建和升级数据库的方法，接下来就该学习一下如何对表中的数据进行操作了。其实我们可以对数据进行的操作也就无非四种，即 CRUD。其中 C 代表添加（Create），R 代表查询（Retrieve），U 代表更新（Update），D 代表删除（Delete）。每一种操作又各自对应了一种 SQL 命令，如果你比较熟悉 SQL 语言的话，一定会知道添加数据时使用 insert，查询数据时使用 select，更新数据时使用 update，删除数据时使用 delete。但是开发者的水平总会是参差不齐的，未必每一个人都能非常熟悉地使用 SQL 语言，因此 Android 也是提供了一系列的辅助性方法，使得在 Android 中即使不去编写 SQL 语句，也能轻松完成所有的 CRUD 操作。

前面我们已经知道，调用 SQLiteOpenHelper 的 getReadableDatabase()或 getWritableDatabase()

方法是可以用于创建和升级数据库的，不仅如此，这两个方法还都会返回一个SQLiteDatabase对象，借助这个对象就可以对数据进行CRUD操作了。

那么我们一个一个功能地看，首先学习一下如何向数据库的表中添加数据吧。SQLiteDatabase中提供了一个insert()方法，这个方法就是专门用于添加数据的。它接收三个参数，第一个参数是表名，我们希望向哪张表里添加数据，这里就传入该表的名字。第二个参数用于在未指定添加数据的情况下给某些可为空的列自动赋值NULL，一般我们用不到这个功能，直接传入null即可。第三个参数是一个ContentValues对象，它提供了一系列的put()方法重载，用于向ContentValues中添加数据，只需要将表中的每个列名以及相应的待添加数据传入即可。

介绍完了基本用法，接下来还是让我们通过例子的方式来亲身体验一下如何添加数据吧。修改activity_main.xml中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    ……

    <Button
        android:id="@+id/add_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add data"
        />
</LinearLayout>
```

可以看到，我们在布局文件中又新增了一个按钮，稍后就会在这个按钮的点击事件里编写添加数据的逻辑。接着修改MainActivity中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ……
```

```
        Button addData = (Button) findViewById(R.id.add_data);
        addData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                ContentValues values = new ContentValues();
                // 开始组装第一条数据
                values.put("name", "The Da Vinci Code");
                values.put("author", "Dan Brown");
                values.put("pages", 454);
                values.put("price", 16.96);
                db.insert("Book", null, values); // 插入第一条数据
                values.clear();
                // 开始组装第二条数据
                values.put("name", "The Lost Symbol");
                values.put("author", "Dan Brown");
                values.put("pages", 510);
                values.put("price", 19.95);
                db.insert("Book", null, values); // 插入第二条数据
            }
        });
    }

}
```

在添加数据按钮的点击事件里面，我们先获取到了 SQLiteDatabase 对象，然后使用 ContentValues 来对要添加的数据进行组装。如果你比较细心的话应该会发现，这里只对 Book 表里其中四列的数据进行了组装，id 那一列没并没给它赋值。这是因为在前面创建表的时候我们就将 id 列设置为自增长了，它的值会在入库的时候自动生成，所以不需要手动给它赋值了。接下来调用了 insert()方法将数据添加到表当中，注意这里我们实际上添加了两条数据，上述代码中使用 ContentValues 分别组装了两次不同的内容，并调用了两次 insert()方法。

好了，现在可以重新运行一下程序了，界面如图 6.21 所示。

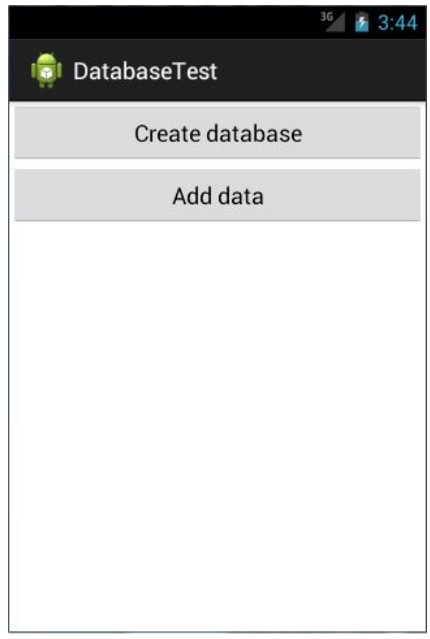

图 6.21

点击一下 Add data 按钮，此时两条数据应该都已经添加成功了，不过为了证实一下，我们还是打开 BookStore.db 数据库瞧一瞧。输入 SQL 查询语句 select * from Book，结果如图 6.22 所示。

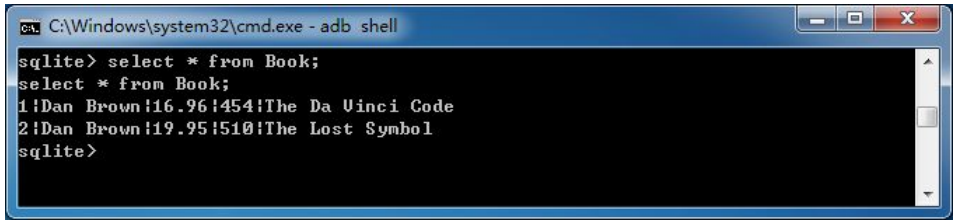

图 6.22

由此可以看出，我们刚刚组装的两条数据，都已经准确无误地添加到 Book 表中了。

6.4.4 更新数据

学习完了如何向表中添加数据，接下来我们看看怎样才能修改表中已有的数据。SQLiteDatabase 中也是提供了一个非常好用的 update()方法用于对数据进行更新，这个方法接收四个参数，第一个参数和 insert()方法一样，也是表名，在这里指定去更新哪张表里的数

据。第二个参数是 ContentValues 对象，要把更新数据在这里组装进去。第三、第四个参数用于去约束更新某一行或某几行中的数据，不指定的话默认就是更新所有行。

那么接下来我们仍然是在 DatabaseTest 项目的基础上修改，看一下更新数据的具体用法。比如说刚才添加到数据库里的第一本书，由于过了畅销季，卖得不是很火了，现在需要通过降低价格的方式来吸引更多的顾客，我们应该怎么操作呢？首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    ......

    <Button
        android:id="@+id/update_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Update data"
        />
</LinearLayout>
```

布局文件中的代码就已经非常简单了，就是添加了一个用于更新数据的按钮。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ......
        Button updateData = (Button) findViewById(R.id.update_data);
        updateData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
                ContentValues values = new ContentValues();
                values.put("price", 10.99);
                db.update("Book", values, "name = ?", new String[] { "The Da
Vinci Code" });
            }
        });
    }

}
```

这里在更新数据按钮的点击事件里面构建了一个 ContentValues 对象，并且只给它指定了一组数据，说明我们只是想把价格这一列的数据更新成 10.99。然后调用了 SQLiteDatabase 的 update()方法去执行具体的更新操作，可以看到，这里使用了第三、第四个参数来指定具体更新哪几行。第三个参数对应的是 SQL 语句的 where 部分，表示去更新所有 name 等于?的行，而?是一个占位符，可以通过第四个参数提供的一个字符串数组为第三个参数中的每个占位符指定相应的内容。因此上述代码想表达的意图就是，将名字是 The Da Vinci Code 的这本书的价格改成 10.99。

现在重新运行一下程序，界面如图 6.23 所示。

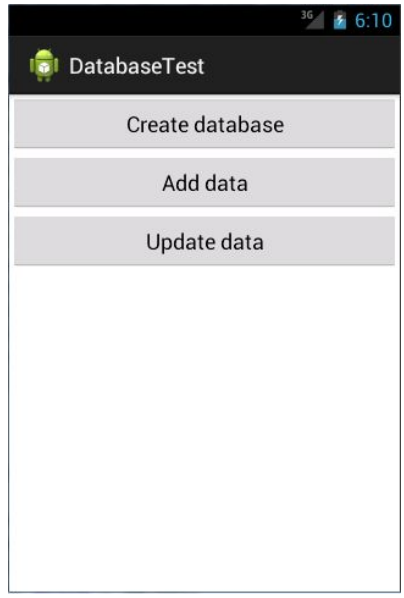

图 6.23

点击一下 Update data 按钮后，再次输入查询语句查看表中的数据情况，结果如图 6.24 所示。

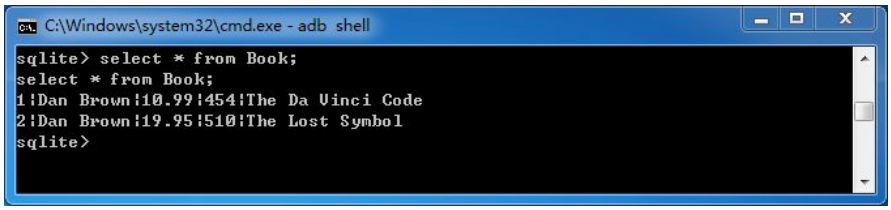

图　6.24

可以看到，The Da Vinci Code 这本书的价格已经被成功改为 10.99 了。

6.4.5　删除数据

怎么样？添加和更新数据的功能都还挺简单的吧，代码也不多，理解起来又容易，那么我们要马不停蹄地开始学习下一种操作了，即如何从表中删除数据。

删除数据对你来说应该就更简单了，因为它所需要用到的知识点你全部已经学过了。SQLiteDatabase 中提供了一个 delete()方法专门用于删除数据，这个方法接收三个参数，第一个参数仍然是表名，这个已经没什么好说的了，第二、第三个参数又是用于去约束删除某一行或某几行的数据，不指定的话默认就是删除所有行。

是不是理解起来很轻松了？那我们就继续动手实践吧，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    ......

    <Button
        android:id="@+id/delete_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Delete data"
        />
</LinearLayout>
```

仍然是在布局文件中添加了一个按钮，用于删除数据。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {
```

```
    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ……
        Button deleteButton = (Button) findViewById(R.id.delete_data);
        deleteButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                db.delete("Book", "pages > ?", new String[] { "500" });
            }
        });
    }

}
```

可以看到，我们在删除按钮的点击事件里指明去删除 Book 表中的数据，并且通过第二、第三个参数来指定仅删除那些页数超过 500 页的书籍。当然这个需求很奇怪，这里也仅仅是为了做个测试。你可以先查看一下当前 Book 表里的数据，其中 The Lost Symbol 这本书的页数超过了 500 页，也就是说当我们点击删除按钮时，这条记录应该会被删除掉。

现在重新运行一下程序，界面如图 6.25 所示。

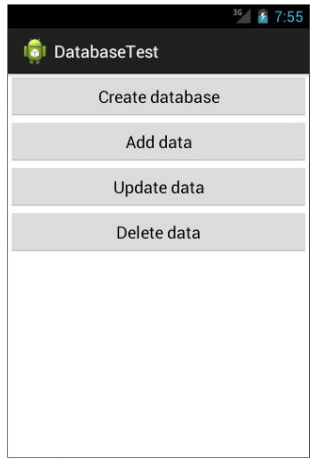

图 6.25

点击一下 Delete data 按钮后，再次输入查询语句查看表中的数据情况，结果如图 6.26 所示。

图　6.26

这样就可以明显地看出，The Lost Symbol 这本书的数据已经被删除了。

6.4.6　查询数据

终于到了最后一种操作了，掌握了查询数据的方法之后，你也就将数据库的 CRUD 操作全部学完了。不过千万不要因此而放松，因为查询数据也是在 CRUD 中最复杂的一种操作。

我们都知道 SQL 的全称是 Structured Query Language，翻译成中文就是结构化查询语言。它的大部功能都是体现在“查”这个字上的，而“增删改”只是其中的一小部分功能。由于 SQL 查询涉及的内容实在是太多了，因此在这里我不准备对它展开来讲解，而是只会介绍 Android 上的查询功能。如果你对 SQL 语言非常感兴趣，可以找一本专门介绍 SQL 的书进行学习。

相信你已经猜到了，SQLiteDatabase 中还提供了一个 query()方法用于对数据进行查询。这个方法的参数非常复杂，最短的一个方法重载也需要传入七个参数。那我们就先来看一下这七个参数各自的含义吧，第一个参数不用说，当然还是表名，表示我们希望从哪张表中查询数据。第二个参数用于指定去查询哪几列，如果不指定则默认查询所有列。第三、第四个参数用于去约束查询某一行或某几行的数据，不指定则默认是查询所有行的数据。第五个参数用于指定需要去 group by 的列，不指定则表示不对查询结果进行 group by 操作。第六个参数用于对 group by 之后的数据进行进一步的过滤，不指定则表示不进行过滤。第七个参数用于指定查询结果的排序方式，不指定则表示使用默认的排序方式。更多详细的内容可以参考下表。其他几个 query()方法的重载其实也大同小异，你可以自己去研究一下，这里就不再进行介绍了。

query()方法参数	对应 SQL 部分	描述
table	from table_name	指定查询的表名
columns	select column1, column2	指定查询的列名
selection	where column = value	指定 where 的约束条件
selectionArgs	-	为 where 中的占位符提供具体的值
groupBy	group by column	指定需要 group by 的列
having	having column = value	对 group by 后的结果进一步约束
orderBy	order by column1, column2	指定查询结果的排序方式

虽然 query()方法的参数非常多，但是不要对它产生畏惧，因为我们不必为每条查询语句都指定上所有的参数，多数情况下只需要传入少数几个参数就可以完成查询操作了。调用 query()方法后会返回一个 Cursor 对象，查询到的所有数据都将从这个对象中取出。

下面还是让我们通过例子的方式来体验一下查询数据的具体用法，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    ......

    <Button
        android:id="@+id/query_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Query data"
        />
</LinearLayout>
```

这个已经没什么好说的了，添加了一个按钮用于查询数据。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ……
        Button queryButton = (Button) findViewById(R.id.query_data);
        queryButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                // 查询Book表中所有的数据
                Cursor cursor = db.query("Book", null, null, null, null, null, null);
                if (cursor.moveToFirst()) {
                    do {
                        // 遍历Cursor对象，取出数据并打印
                        String name = cursor.getString(cursor.
getColumnIndex("name"));
                        String author = cursor.getString(cursor.
getColumnIndex("author"));
                        int pages = cursor.getInt(cursor.getColumnIndex
("pages"));
                        double price = cursor.getDouble(cursor.
getColumnIndex("price"));
                        Log.d("MainActivity", "book name is " + name);
                        Log.d("MainActivity", "book author is " + author);
                        Log.d("MainActivity", "book pages is " + pages);
                        Log.d("MainActivity", "book price is " + price);
                    } while (cursor.moveToNext());
                }
                cursor.close();
            }
        });
    }

}
```

可以看到，我们首先在查询按钮的点击事件里面调用了 SQLiteDatabase 的 query()方法去查询数据。这里的 query()方法非常简单，只是使用了第一个参数指明去查询 Book 表，后面的参数全部为 null。这就表示希望查询这张表中的所有数据，虽然这张表中目前只剩下一条数据了。查询完之后就得到了一个 Cursor 对象，接着我们调用它的 moveToFirst()方法将数

据的指针移动到第一行的位置，然后进入了一个循环当中，去遍历查询到的每一行数据。在这个循环中可以通过 Cursor 的 getColumnIndex()方法获取到某一列在表中对应的位置索引，然后将这个索引传入到相应的取值方法中，就可以得到从数据库中读取到的数据了。接着我们使用 Log 的方式将取出的数据打印出来，借此来检查一下读取工作有没有成功完成。最后别忘了调用 close()方法来关闭 Cursor。

好了，现在再次重新运行程序，界面如图 6.27 所示。

图 6.27

点击一下 Query data 按钮后，查看 LogCat 的打印内容，结果如图 6.28 所示。

Tag	Text
MainActivity	book name is The Da Vinci Code
MainActivity	book author is Dan Brown
MainActivity	book pages is 454
MainActivity	book price is 10.99

图 6.28

可以看到，这里已经将 Book 表中唯一的一条数据成功地读取出来了。

当然这个例子只是对查询数据的用法进行了最简单的示范，在真正的项目中你可能会遇到比这要复杂得多的查询功能，更多高级的用法还需要你自己去慢慢摸索，毕竟 query()方法中还有那么多的参数我们都还没用到呢。

6.4.7　使用 SQL 操作数据库

虽然 Android 已经给我们提供了很多非常方便的 API 用于操作数据库，不过总会有一些人不习惯去使用这些辅助性的方法，而是更加青睐于直接使用 SQL 来操作数据库。这种人一般都是属于 SQL 大牛，如果你也是其中之一的话，那么恭喜，Android 充分考虑到了你们的编程习惯，同样提供了一系列的方法，使得可以直接通过 SQL 来操作数据库。

下面我就来简略演示一下，如何直接使用 SQL 来完成前面几小节中学过的 CRUD 操作。

添加数据的方法如下：

```
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Da Vinci Code", "Dan Brown", "454", "16.96" });
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Lost Symbol", "Dan Brown", "510", "19.95" });
```

更新数据的方法如下：

```
db.execSQL("update Book set price = ? where name = ?", new String[] { "10.99",
"The Da Vinci Code" });
```

删除数据的方法如下：

```
db.execSQL("delete from Book where pages > ?", new String[] { "500" });
```

查询数据的方法如下：

```
db.rawQuery("select * from Book", null);
```

可以看到，除了查询数据的时候调用的是 SQLiteDatabase 的 rawQuery()方法，其他的操作都是调用的 execSQL()方法。以上演示的几种方式，执行结果会和前面几小节中我们学习的 CRUD 操作的结果完全相同，选择使用哪一种方式就看你个人的喜好了。

6.5　SQLite 数据库的最佳实践

在上一节里我们只能算是学习了 SQLite 数据库的基本用法，如果你想继续深入钻研，SQLite 数据库中可拓展的知识就太多了。既然还有那么多的高级技巧在等着我们，自然又要进入到本章的最佳实践环节了。

6.5.1 使用事务

前面我们已经知道，SQLite 数据库是支持事务的，事务的特性可以保证让某一系列的操作要么全部完成，要么一个都不会完成。那么在什么情况下才需要使用事务呢？想象以下场景，比如你正在进行一次转账操作，银行会将转账的金额先从你的账户中扣除，然后再向收款方的账户中添加等量的金额。看上去好像没什么问题吧？可是，如果当你账户中的金额刚刚被扣除，这时由于一些异常原因导致对方收款失败，这一部分钱就凭空消失了！当然银行肯定已经充分考虑到了这种情况，它会保证扣钱和收款的操作要么一起成功，要么都不会成功，而使用的技术当然就是事务了。

接下来我们看一看如何在 Android 中使用事务吧，仍然是在 DatabaseTest 项目的基础上进行修改。比如 Book 表中的数据都已经很老了，现在准备全部废弃掉替换成新数据，可以先使用 delete()方法将 Book 表中的数据删除，然后再使用 insert()方法将新的数据添加到表中。我们要保证的是，删除旧数据和添加新数据的操作必须一起完成，否则就还要继续保留原来的旧数据。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    ......

    <Button
        android:id="@+id/replace_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Replace data"
        />
</LinearLayout>
```

可以看到，这里又添加了一个按钮，用于进行数据替换操作。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ……
        Button replaceData = (Button) findViewById(R.id.replace_data);
        replaceData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                db.beginTransaction(); // 开启事务
                try {
                    db.delete("Book", null, null);
                    if (true) {
                        // 在这里手动抛出一个异常，让事务失败
                        throw new NullPointerException();
                    }
                    ContentValues values = new ContentValues();
                    values.put("name", "Game of Thrones");
                    values.put("author", "George Martin");
                    values.put("pages", 720);
                    values.put("price", 20.85);
                    db.insert("Book", null, values);
                    db.setTransactionSuccessful(); // 事务已经执行成功
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    db.endTransaction(); // 结束事务
                }
            }
        });
    }

}
```

上述代码就是 Android 中事务的标准用法，首先调用 SQLiteDatabase 的 beginTransaction()方法来开启一个事务，然后在一个异常捕获的代码块中去执行具体的数据库操作，当所有的操作都完成之后，调用 setTransactionSuccessful()表示事务已经执行成功了，最后在 finally 代码块中调用 endTransaction()来结束事务。注意观察，我们在删除旧数据的操作完成后手动抛出了一个 NullPointerException，这样添加新数据的代码就执行不到了。不过由于事务的存在，中途出现异常会导致事务的失败，此时旧数据应该是删除不掉的。

现在可以运行一下程序并点击 Replace data 按钮，你会发现，Book 表中存在的还是之前的旧数据。然后将手动抛出异常的那行代码去除，再重新运行一下程序，此时点击一下 Replace data 按钮就会将 Book 表中的数据替换成新数据了。

6.5.2 升级数据库的最佳写法

在 6.4.2 节中我们学习的升级数据库的方式是非常粗暴的，为了保证数据库中的表是最新的，我们只是简单地在 onUpgrade()方法中删除掉了当前所有的表，然后强制重新执行了一遍 onCreate()方法。这种方式在产品的开发阶段确实可以用，但是当产品真正上线了之后就绝对不行了。想象以下场景，比如你编写的某个应用已经成功上线，并且还拥有了不错的下载量。现在由于添加新功能的原因，使得数据库也需要一起升级，然后用户更新了这个版本之后发现以前程序中存储的本地数据全部丢失了！那么很遗憾，你的用户群体可能已经流失一大半了。

听起来好像挺恐怖的样子，难道说在产品发布出去之后还不能升级数据库了？当然不是，其实只需要进行一些合理的控制，就可以保证在升级数据库的时候数据并不会丢失了。

下面我们就来学习一下如何实现这样的功能，你已经知道，每一个数据库版本都会对应一个版本号，当指定的数据库版本号大于当前数据库版本号的时候，就会进入到 onUpgrade()方法中去执行更新操作。这里需要为每一个版本号赋予它各自改变的内容，然后在 onUpgrade()方法中对当前数据库的版本号进行判断，再执行相应的改变就可以了。

接着就让我们来模拟一个数据库升级的案例，还是由 MyDatabaseHelper 类来对数据库进行管理。第一版的程序要求非常简单，只需要创建一张 Book 表，MyDatabaseHelper 中的代码如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    public MyDatabaseHelper(Context context, String name, CursorFactory
factory, int version) {
        super(context, name, factory, version);
    }

    @Override
```

```
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

}
```

不过，几星期之后又有了新需求，这次需要向数据库中再添加一张 Category 表。于是，修改 MyDatabaseHelper 中的代码，如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    public MyDatabaseHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        switch (oldVersion) {
```

```
            case 1:
                db.execSQL(CREATE_CATEGORY);
            default:
            }
        }

    }
```

可以看到，在 onCreate()方法里我们新增了一条建表语句，然后又在 onUpgrade()方法中添加了一个 switch 判断，如果用户当前数据库的版本号是 1，就只会创建一张 Category 表。这样当用户是直接安装的第二版的程序时，就会将两张表一起创建。而当用户是使用第二版的程序覆盖安装第一版的程序时，就会进入到升级数据库的操作中，此时由于 Book 表已经存在了，因此只需要创建一张 Category 表即可。

但是没过多久，新的需求又来了，这次要给 Book 表和 Category 表之间建立关联，需要在 Book 表中添加一个 category_id 的字段。再次修改 MyDatabaseHelper 中的代码，如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text, "
            + "category_id integer)";

    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    public MyDatabaseHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
```

```
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            switch (oldVersion) {
            case 1:
                db.execSQL(CREATE_CATEGORY);
            case 2:
                db.execSQL("alter table Book add column category_id integer");
            default:
            }
        }

    }
```

可以看到，首先我们在 Book 表的建表语句中添加了一个 category_id 列，这样当用户直接安装第三版的程序时，这个新增的列就已经自动添加成功了。然而，如果用户之前已经安装了某一版本的程序，现在需要覆盖安装，就会进入到升级数据库的操作中。在 onUpgrade() 方法里，我们添加了一个新的 case，如果当前数据库的版本号是 2，就会执行 alter 命令来为 Book 表新增一个 category_id 列。

这里请注意一个非常重要的细节，switch 中每一个 case 的最后都是没有使用 break 的，为什么要这么做呢？这是为了保证在跨版本升级的时候，每一次的数据库修改都能被全部执行到。比如用户当前是从第二版程序升级到第三版程序的，那么 case 2 中的逻辑就会执行。而如果用户是直接从第一版程序升级到第三版程序的，那么 case 1 和 case 2 中的逻辑都会执行。使用这种方式来维护数据库的升级，不管版本怎样更新，都可以保证数据库的表结构是最新的，而且表中的数据也完全不会丢失了。

6.6　小结与点评

经过了一章漫长地学习，我们终于可以缓解一下疲劳，对本章所学的知识进行梳理和总结了。本章主要是对 Android 常用的数据持久化方式进行了详细的讲解，包括文件存储、SharedPreferences 存储以及数据库存储。其中文件适用于存储一些简单的文本数据或者二进制数据，SharedPreferences 适用于存储一些键值对，而数据库则适用于存储那些复杂的关系型数据。虽然目前你已经掌握了这三种数据持久化方式的用法，但是能够根据项目的实际需求来选择最合适的方式也是你未来需要继续探索的。

那么正如上一章小结里提到的，既然现在我们已经掌握了 Android 中的数据持久化技术，接下来就应该继续学习 Android 中剩余的四大组件了。放松一下自己，然后一起踏上内容提

供器的学习之旅。

经验值：+10000　　目前经验值：36905

级别：资深鸟

赢得宝物：战胜高级存储王。拾取高级存储王掉落的宝物，超高速 500PB 硬盘一块。剑齿虎皮 Android 战袍一套、强力大脑永久扩容丸一颗。高级存储王名叫术巨酷，是一位哲学大湿，神界为数很少的长期定居的人族，同时也是一个在神界很难见到的十分虚伪的家伙，在神界非常不招人待见。他有两个大脑，其中一个擅长面利都学派，另一个则信奉牛克思什么主义哲学。我平生最受不了就是这种装逼的家伙，我一个箭步冲上去，一拳将他打翻在地，然后坐在他身上足足 K 了他十分钟。我相信我已经成功地在他的两个脑子中植入了某种我还没来得及起名的新学派。好吧，也许可以叫“你大爷学派”。我整了整战袍。继续前进。