第5章 Android 数据存储与访问

5.1 简单存储

5.1.1 SharedPreferences

SharedPreferences 是 Android 中最容易理解的数据存储技术，常用来存储一些轻量级的数据，采用 key-value（键值对）的方式保存数据，类似于 Web 程序的 Cookie，通常用来保存一些配置文件数据、用户名及密码等。

SharedPreferences 不仅能够保存数据，还能实现不同应用程序间的数据共享，支持三种访问模式：私有（MODE_PRIVATE）、全局读（MODE_WORLD_READABLE）、全局写（MODE_WORLD_WRITEABLE）。其中 MODE_PRIVATE 是默认模式，该模式下的配置文件只允许本程序和享有本程序 ID 的程序访问；MODE_WORLD_READABLE 模式允许其他应用程序读文件；MODE_WORLD_WRITEABLE 模式允许其他应用程序写文件。如果既要全局读又要全局写，可将访问模式设置为 MODE_WORLD_READABLE ＋MODE_WORLD_WRITEABLE。

除了定义 SharedPreferences 的访问模式，还要定义 SharedPreferences 的名称，该名称是 SharedPreferences 在 Android 文件系统中保存的文件名称，一般声明为常量字符串，以方便在代码中多次使用，如：

```
SharedPreferences sharedPreferences = getSharedPreferences("filename", MODE_PRIVATE);
```

其中，getSharedPreferences()为 Android 系统函数，通过它可获得 SharedPreferences 实例。

获取 SharedPreferences 实例后，通过 SharedPreferences.Editor 类对 SharedPreferences 实例进行修改，完成数据设置，最后调用 commit()函数保存数据。SharedPreferences 广泛支持各种基本数据类型，包括整型、布尔型、浮点型和长整形等，如：

```
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("Name", "Tom");
editor.putFloat("Height", 1.78f);
editor.commit();
```

如果需要从已保存的 SharedPreferences 中读取数据，同样调用 getSharedPreferences()函数，并在函数的第 1 个参数中指明需要访问的 SharedPreferences 名称，然后通过

get < Type >()函数获取保存在 SharedPreferences 中的键值对，如：

```
SharedPreferences mySdPferences = getSharedPreferences("filename", MODE_PRIVATE);
String name = mySdPferences.getString("Name", "Default Name");
float height = mySdPferences.getFloat("Height", 1.70f);
```

其中，get < Type >()函数的第 1 个参数是键值对的键名，第 2 个参数是无法获取键值时的默认值。

Android 系统为每个应用程序建立了与包同名的目录，用来保存应用程序产生的数据文件，包括普通文件、SharedPreferences 文件和数据库文件等。SharedPreferences 产生的文件就保存在/data/data/< package name >/shared_prefs 目录下。

5.1.2 使用 SharedPreferences 存储用户登录信息

SharedPreferences 使用方法比较简单，下面以一个例子来讲解 SharedPreferences 的用法。

【例 5-1】 演示使用 SharedPreferences 保存用户名和密码方法。

程序 SharedPreferencesDemo 演示了如何使用 SharedPreferences 保存用户名和密码的方法。用户输入用户名和密码后单击“保存”按钮，数据被保存在 SharedPreferences 文件中。以后每次程序重新启动后，会将保存的用户登录信息从 SharedPreferences 文件中读出并显示在输入框中，界面效果如图 5.1 所示。

图 5.1 SharedPreferencesDemo 程序界面

该程序的 MainActivity.java 文件内容如下：

```
package edu.cqut.sharedpreferencesdemo;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class MainActivity extends Activity
{
    SharedPreferences mySharedPreferences;
    Button saveButton;
    EditText editName,editPswrod;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editName = (EditText)findViewById(R.id.editName);
        editPswrod = (EditText)findViewById(R.id.editPassword);
        saveButton = (Button)findViewById(R.id.button1);
        saveButton.setOnClickListener(new Button.OnClickListener()
        {
            @Override
            public void onClick(View v) {
                mySharedPreferences = getSharedPreferences("userInfo", Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = mySharedPreferences.edit();
                editor.putString("usename", editName.getText().toString());
                editor.putString("password", editPswrod.getText().toString());
                editor.commit();
                Toast.makeText(MainActivity.this, "写入 Sharedpreferences 成功!",
                        Toast.LENGTH_LONG).show();
            }
        });
        mySharedPreferences = getSharedPreferences("userInfo", Context.MODE_PRIVATE);
        String usename = mySharedPreferences.getString("usename", "");
        String password = mySharedPreferences.getString("password", "");
        editName.setText(usename);
        editPswrod.setText(password);
    }
}
```

在本程序中，shared_prefs 目录中生成了一个名为 userInfo.xml 的文件。运行程序后，在开发环境中选择菜单的 Tools→Android→Android Device Monitor，弹出如图 5.2 所示的窗口，切换到 File Explorer 页面，可以看到 userInfo.xml 保存在/data/data/edu.cqut.sharepreferences demo/shared_prefs 目录下，文件大小为 144 字节，在 Linux 下的权限为 -rw-rw----。

Linux 系统中文件权限分别描述了创建者、同组用户和其他用户对文件的操作限制。x 表示可执行，r 表示可读，w 表示可写，d 表示目录，-表示普通文件，图 5.2 中的“edu.cqut.sharedpreferencesdemo”的权限为 drwxr-x--x，表示目录、可被创建者读写及执行、被同组用户读及执行、其他用户只能执行；由于设置 mySharedPreferences 实例的权限为 MODE_PRIVATE，因此 userInfo.xml 的权限为仅创建者和同组用户具有读写文件的权限。

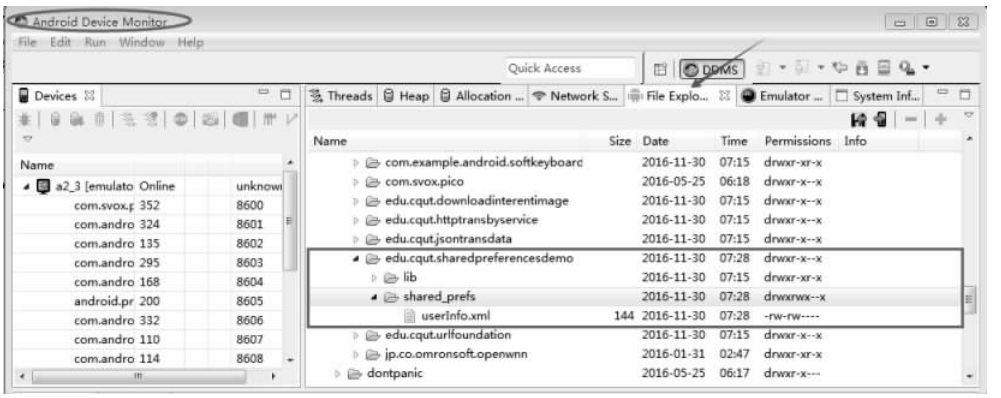


图 5.2　userInfo.xml 文件

5.2　文 件 存 储

5.2.1　内部存储

除了使用 SharedPreferences 存取少量数据外，更多的是使用文件系统进行数据的存取。Android 系统允许应用程序创建仅能够自身访问的私有文件，文件保存在设备的内部存储器上，位于系统的/data/data/< package name >/files 目录中。Android 使用 Linux 的文件系统，支持标准 Java 的 IO 类和方法，存取文件主要使用数据流方式。

流是一个可被顺序访问的数据序列，是对计算机输入数据和输出数据的抽象，有输入流和输出流之分，输入流用来读取数据，输出流则相反，用来写入数据。常用的文件字节数据流有：

(1) FileInputStream：文件字节输入流；

(2) FileOutputStream：文件字节输出流。

为了能使用字节流，可以使用 openFileOutputStream ( )、openFileOutput ( ) 和 openFileInput()等函数。openFileOutputStream()的语法格式如下：

```
public FileOutputStream openFileOutput(String name, int mode)
```

其中，第 1 个参数是文件名称，不可以包含描述路径的斜杠；第 2 个参数是操作模式。Android 系统支持 4 种文件操作模式，如表 5.1 所示。函数的返回值是 FileOutputStream 类型。

表 5.1　文件操作模式

模　　式	说　　明
MODE_PRIVATE	私有模式，默认模式，文件仅能够被创建文件的程序访问，或具有相同 UID 的程序访问
MODE_APPEND	追加模式，如果文件已存在，则在文件的结尾添加新数据
MODE_WORLD_READABLE	全局读模式，允许任何程序读取私有文件
MODE_WORLD_WRITEABLE	全局写模式，允许任何程序写入私有文件

使用 openFileOutput()函数输出数据示例代码如下：

```
FileOutputStream fos = openFileOutput("fileDemo.txt", MODE_PRIVATE);
String text = "Some data";
fos.write(text.getBytes());
fos.flush();
fos.close();
```

由于 FileOutputStream 是字节流，因此对于字符串数据，需要先将其转换为字节数组，再使用 write()方法写入。如果写入的数据量较小，系统会把数据保存在数据缓冲区中，等数据量积累到一定程度后再一次性写入文件。因此，在调用 close()函数关闭文件前，务必使用 flush()函数将缓冲区内的所有数据写入文件，否则可能导致部分数据丢失。

openFileInput()函数语法格式为：

```
public FileInputStream openFileInput(String name)
```

参数为文件名，同样不允许包含描述路径的斜杠。使用 openFileInput()打开已有文件，并以二进制方式读取数据的示例代码如下：

```
FileInputStream fis = openFileInput("fileDemo.txt");
byte[] readBytes = new byte[fis.available()];
while (fis.read(readBytes) != -1){}
```

由于文件操作可能会遇到各种问题而导致操作失败，因此在代码中应使用 try/catch 捕获可能产生的异常。

5.2.2 外部存储

Android 外部存储设备一般指 Micro SD 卡，存储在 SD 卡上的文件称为外部文件。SD 卡使用 FAT(File Allocation Table)文件系统，不支持访问模式和权限控制。Android 模拟器支持 SD 卡的模拟，可以设置模拟器中 SD 卡的容量，模拟器启动后会自动加载 SD 卡。正确加载 SD 卡后，SD 卡中的目录和文件被映射到/mnt/sdcard 目录下。因为用户可以加载或者卸载 SD 卡，因此编程访问 SD 卡前要检测 SD 卡目录是否可用；如果不可用，说明设备中 SD 卡已经被卸载；如果可用，则直接使用标准的 java.io.File 类进行访问。

使用 SD 卡存取文件，需要在程序的 AndroidManifest.xml 中注册用户对 SD 卡的权限，分别是加载卸载文件系统权限和向外部存储器写入数据的权限，如：

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

对 SD 卡进行读写操作可以用环境变量访问类 Environment 的下面两个方法：

(1) getExternalStorageState()：获取当前存储设备的状态。

(2) getExternalStorageDirectory()：获取 SD 卡的根目录。

示例代码如下：

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
{
File path = Environment.getExternalStorageDirectory();        //获取 SD 卡目录路径
File sdfile = new File(path, "filename.txt");       //指定文件 filename.txt 在 SD 卡中的位置
//读写操作
…
}
```

上面代码中常量 Environment.MEDIA_MOUNTED 表示对 SD 卡具有读写权限。

下面以一个示例说明如何使用存储器存取文件。

【例 5-2】 演示使用输入输出流存储文件。

程序 FileStorageDemo 使用 FileOutputStream 和 FileInputStream 存取用户编写的字符串，其运行界面如图 5.3 所示。用户在编辑框中输入相应文字后单击相应按钮完成文字的保存和存储。

图 5.3 文件存储程序运行界面

程序 FileStorageDemo 的 MainActivity.java 文件内容如下：

```
package edu.cqut.filestoragedemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.content.Context;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.Toast;

public class MainActivity extends Activity
{
    EditText editText;                          //接收用户输入的字符串
    Button btnSave,btnRead, btnSaveSD, btnReadSD;
    String fileName = "test.txt";               //文件名称
    String str;                                 //要存取的字符串
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText = (EditText)findViewById(R.id.editText);
        btnSave = (Button)findViewById(R.id.btnSave);
        btnSave.setOnClickListener(new mClick());
        btnRead = (Button)findViewById(R.id.btnRead);
        btnRead.setOnClickListener(new mClick());
        btnSaveSD = (Button)findViewById(R.id.btnSaveSD);
        btnSaveSD.setOnClickListener(new mClick());
        btnReadSD = (Button)findViewById(R.id.btnReadSD);
        btnReadSD.setOnClickListener(new mClick());
    }
    class mClick implements Button.OnClickListener{
        @Override
        public void onClick(View arg0) {
            if(arg0 == btnSave)                 //存储文件到内部存储器
                savefile();
            else if(arg0 == btnRead)            //从内部存储器读取文件
                readfile(fileName);
            else if(arg0 == btnSaveSD)          //存储文件到 SD 卡中
                saveSDfile();
            else if(arg0 == btnReadSD)          //从 SD 卡中读取文件
                readSDfile(fileName);
        }
    }
    void savefile()
    {
        str = editText.getText().toString();
        try{
            FileOutputStream f_out = openFileOutput(fileName, Context.MODE_PRIVATE);
            f_out.write(str.getBytes());
            Toast.makeText(MainActivity.this,"文件保存成功", Toast.LENGTH_LONG).show();
        }
        catch(FileNotFoundException e){
            e.printStackTrace();
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
```

```
void readfile(String fileName)
{
    byte[ ] buffer = new byte[1024];
    FileInputStream in_file = null;
    try{
        in_file = openFileInput(fileName);
        int bytes = in_file.read(buffer);
        str = new String (buffer,0,bytes);
        Toast.makeText(MainActivity.this, "文件内容: " + str, Toast.LENGTH_SHORT).show();
    }
    catch (FileNotFoundException e) {
        java.lang.System.out.print("文件不存在!");
    }
    catch (IOException e) {
        java.lang.System.out.print("IO 流错误");
    }
}
void saveSDfile()
{
    str = editText.getText().toString();
    Toast.makeText(MainActivity.this, "文件内容: " + str, Toast.LENGTH_LONG).show();
    if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
    {
        File path = Environment.getExternalStorageDirectory();    //获取 SD 卡目录路径
        File sdfile = new File(path, fileName);
        try{
            FileOutputStream f_out = new FileOutputStream(sdfile);
            f_out.write(str.getBytes());
            Toast.makeText(MainActivity.this,"文件保存到 SD 卡成功",
                           Toast.LENGTH_LONG).show();
        }
        catch (FileNotFoundException e){
            e.printStackTrace();
        }
        catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }
}
void readSDfile(String fileName)
{
    if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
    {
        File path = Environment.getExternalStorageDirectory();    //获取 SD 卡目录路径
        File sdfile = new File(path, fileName);
        try{
            FileInputStream in_file = new FileInputStream(sdfile);
            byte[ ] buffer = new byte[1024];
            int bytes = in_file.read(buffer);
```

```
                str = new String(buffer,0,bytes);
                Toast.makeText(MainActivity.this,"文件内容:" + str, Toast.LENGTH_LONG).show();
            }
            catch(FileNotFoundException e){
                java.lang.System.out.print("文件不存在");
            }
            catch (Exception e) {
                java.lang.System.out.print("IO 流错误");
            }
        }
    }
}
```

为了保证程序正常运行,最后不要忘了在 AndroidManifest.xml 中注册用户对 SD 卡的权限。

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

5.2.3 编写一个文件存储访问类

文件存储是很多程序经常用到的功能,这里编写一个文件存储访问类——DataFileAccess 类,该类囊括了文件操作的常用功能,可以将它用于程序中,从而简化开发流程。

```
import android.content.Context;
import android.content.res.Resources;
import android.os.Environment;
import android.os.StatFs;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.Charset;
public class DataFileAccess
{
    private Context mContext;
    private String mSDPath;           //SD 卡路径
    DataFileAccess(Context cont)
    {
        mContext = cont;
        mSDPath = Environment.getExternalStorageDirectory().getPath() + "/";
    }
    /** 判断 SD 卡是否存在?是否可以进行读写? */
    public boolean SDCardState()
    {
        if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
```

```
            return true;
        else
            return false;
    }
    /** 获取 SD 卡文件路径 */
    public String SDCardPath()
    {
        if(SDCardState()){                    //如果 SD 卡存在并且可以读写
            mSDPath = Environment.getExternalStorageDirectory().getPath();
            return mSDPath;
        }
        else{
            return null;
        }
    }
    /** 获取 SD 卡可用容量大小(MB) */
    public long SDCardFree()
    {
        if(null!= SDCardPath()){
            StatFs statfs = new StatFs(SDCardPath());
            //获取 SD 卡的 Block 可用数
            long availaBlocks = statfs.getAvailableBlocks();
            //获取每个 block 的大小
            long blockSize = statfs.getBlockSize();
            //计算 SD 卡可用容量大小(MB)
            long SDFreeSize = availaBlocks * blockSize/1024/1024;
            return SDFreeSize;
        }
        else{
            return 0;
        }
    }
    /**
     * 在 SD 卡上创建目录
     * @param dirName,要创建的目录名
     * @return 创建得到的目录
     */
    public boolean createSDDir(String dirName)
    {
            String [] strSubDir = dirName.split("/");
            String strCurrentPath = mSDPath;
            for (int i = 0; i < strSubDir.length; i++) {
                strCurrentPath += "/" + strSubDir[i];
                File curDir = new File(strCurrentPath);
                if (!curDir.exists()) {    //当前目录不存在
                    //创建目录
                    boolean isCreated = curDir.mkdir();
                    if (!isCreated) {      //目录创建失败
                        System.out.println(strCurrentPath + " 创建失败!");
                        return false;
```

```
                }
            }
        }
        return true;
    }
    /**
     * 删除SD卡上的目录
     * @param dirName
     */
    public boolean delSDDir(String dirName)
    {
        File dir = new File(mSDPath + "/" + dirName);
        return delDir(dir);
    }
    /**
     * 删除一个目录(可以是非空目录)
     * @param dir
     */
    public boolean delDir(File dir)
    {
        if (dir == null || !dir.exists() || dir.isFile())
            return false;
        for (File file : dir.listFiles())
        {
            if (file.isFile()) {
                file.delete();
            } else if (file.isDirectory()) {
                delDir(file);              // 递归
            }
        }
        dir.delete();
        return true;
    }
    /**
     * 在SD卡上创建文件
     * @throws IOException
     */
    public File createSDFile(String fileName) throws IOException
    {
        File file = new File(mSDPath + "/" + fileName);
        System.out.println(mSDPath + "/" + fileName);
        file.createNewFile();
        return file;
    }
    /**
     * 判断文件是否已经存在
     * @param fileName, 要检查的文件名
     * @return boolean, true表示存在,false表示不存在
     */
    public boolean isFileExist(String fileName)
```

```
        {
            File file = new File(mSDPath + "/" + fileName);
            boolean isExisted = file.exists();
            return isExisted;
        }
        /**
         * 删除 SD 卡上的文件
         * @param fileName
         */
        public boolean delSDFile(String fileName)
        {
            File file = new File(mSDPath + fileName);
            if (file == null || !file.exists() || file.isDirectory())
                return false;
            file.delete();
            return true;
        }
        /**
         * 拷贝一个文件,srcFile 源文件,destFile 目标文件
         * @param path
         * @throws IOException
         */
        public boolean copyFileTo(File srcFile, File destFile) throws IOException
        {
            if (srcFile.isDirectory() || destFile.isDirectory())
                return false;                    // 判断是否是文件
            FileInputStream fis = new FileInputStream(srcFile);
            FileOutputStream fos = new FileOutputStream(destFile);
            int readLen = 0;
            byte[] buf = new byte[1024];
            while ((readLen = fis.read(buf)) != -1) {
                fos.write(buf, 0, readLen);
            }
            fos.flush();
            fos.close();
            fis.close();
            return true;
        }
        //该函数将文件存储到内部存储器的文件夹
        public void SaveFile(String fileName, byte[] fileData)
        {
                try {
                    FileOutputStream fos = mContext.openFileOutput(fileName, Context.MODE_PRIVATE);
                    fos.write(fileData);         //将 fileData 里的数据写入到输出流中
                    fos.flush();                 //将输出流中所有数据写入文件
                    fos.close();                 //关闭输出流
                }
                catch (Exception e) {
                }
        }
    }
```

5.2.4 “移动点餐系统”中的文件操作

现在利用 Android 系统的文件存储向“移动点餐系统”添加如下功能：

(1) 用 SharedPreferences 存取用户名；

(2) 用内部存储器存取已登录用户的个人信息(用户名、密码、电话号码、地址)；

(3) 将原来存储在项目 res/raw 目录中的菜品图片存储在 SD 卡上。

1. 使用 SharedPreferences 存取用户名

在 MainActivity 类中添加代码如下：

```
public class MainActivity extends Activity
{   …
    private static String mUserFileName = "UserInfo"; //定义 SharedPreferences 数据文件名称
    …
    //使用 SharedPreferences 读取用户名
    private String LoadUserPreferencesName()
    {
            int mode = Activity.MODE_PRIVATE;
            //获取 SharedPreferences 对象
            SharedPreferences usersetting = getSharedPreferences(mUserFileName, mode);
            String username = usersetting.getString("username", "");
            return username;
    }
    …
}
```

修改 MainActivity 类中的 myImageButtonListener 监听器，当用户单击“登录”按钮时，程序先从 SharedPreferences 中读取用户登录名，将其显示在登录对话框的“用户名”编辑框中。在登录过程中如果用户勾选“记住用户名”，则将用户名保存在 SharedPreferences 文件中，否则清除 SharedPreferences 中原有的用户名，即用空字符代替原有用户名。

```
public class myImageButtonListener implements View.OnClickListener
{
    @Override
    public void onClick(View v) {
        switch (v.getId())
        {
        …
        case R.id.imgBtnLogin:                    //用户未登录时该按钮才会出现
            //用户未登录，显示登录对话框让用户登录
            final LoginDialog loginDlg = new LoginDialog(MainActivity.this);
            //从 SharedPreferences 中载入用户名
            String holdName = LoadUserPreferencesName();
            loginDlg.DisplayUserName(holdName);
            loginDlg.show();
            //对话框销毁时的响应事件
            loginDlg.setOnDismissListener(new DialogInterface.OnDismissListener() {
                @Override
```

```
                public void onDismiss(DialogInterface dialog) {
                    switch (loginDlg.mBtnClicked)
                    {
                    case BUTTON_OK:                 //用户单击了"确定"按钮
                        MyApplication appInstance = (MyApplication)getApplication();
                        if (appInstance.g_user.mUserid.equals(loginDlg.mUserId) &&
                         appInstance.g_user.mPassword.equals(loginDlg.mPsword)) {
                            //用户登录成功
                            …
                            //使用 SharedPreferences 保存用户名
                            int mode = Activity.MODE_PRIVATE;   //定义权限为私有
                            //(1)获取 SharedPreferences 对象
                            SharedPreferences usersetting =
                                                getSharedPreferences(mUserFileName, mode);
                            //(2)获得 Editor 类
                            SharedPreferences.Editor ed = usersetting.edit();
                            if (loginDlg.mIsHoldUserId){  //用户勾选"记住用户名"选项
                                //(3)添加用户名数据
                                ed.putString("username", appInstance.g_user.mUserid);
                            }
                            else {
                                //保存空的用户名(即清除用户名)
                                ed.putString("username", "");
                            }
                            ed.commit();                    //保存键值对
                            Toast.makeText(MainActivity.this, "登录成功!",
                                                Toast.LENGTH_LONG).show();
                        }
                        …
                        break;
                    case BUTTON_REGISTER:                   //用户单击了"注册"按钮
                        …
                    }
                }
            });
            return;
        …
        }
    }
}
```

2. 使用内部存储器存取已登录用户的个人信息

将 DataFileAccess 类添加进项目，然后在 DataFileAccess 类中添加两个函数用来保存和读取用户信息，包括用户名、密码、电话和地址，用户信息以字节流的形式保存。

```
public class DataFileAccess
{
    //该函数将用户信息保存到内部存储器的文件
    public void SaveUserInfotoFile(String fileName, MyUser user)
```

```
{
        try {
            FileOutputStream fos = mContext.openFileOutput(fileName, Context.MODE_PRIVATE);
            //将用户名编码为 UTF-8 格式的字节数组
            byte[] idbuf = user.mUserid.getBytes(Charset.forName("UTF-8"));
            byte bufsize = (byte)idbuf.length;
            fos.write(bufsize);      //写入用户名字节长度
            fos.write(idbuf);        //将 mUserid 值,即用户名写入到输出流中

            byte[] psdbuf = user.mPassword.getBytes();
            bufsize = (byte)psdbuf.length;
            fos.write(bufsize);      //写入用户密码字节长度
            fos.write(psdbuf);       //将 mPassword 值,即用户密码写入到输出流中

            byte[] phonebuf = user.mUserphone.getBytes();
            bufsize = (byte)phonebuf.length;
            fos.write(bufsize);      //写入用户电话号码字节长度
            fos.write(phonebuf);     //将 mUserphone 值,即用户电话号码写入到输出流中

            byte[] addbuf = user.mUseraddress.getBytes(Charset.forName("UTF-8"));
            bufsize = (byte)addbuf.length;
            fos.write(bufsize);      //写入用户地址字节长度
            fos.write(addbuf);       //将 mUseraddress 值,即用户地址写入到输出流中

            fos.flush();             //将输出流中所有数据写入文件
            fos.close();             //关闭输出流
        }catch (Exception e) {}
}
//该函数将保存在内部存储器上的用户信息文件读出
public MyUser ReadUserInfofromFile(String fileName)
{
        MyUser userinfo = null;
        try {
            FileInputStream fis = mContext.openFileInput(fileName);
            int fileLen = fis.available();
            if (fileLen == 0)return null;
            userinfo = new MyUser();
            //读入用户名信息
            byte bufsize = (byte)fis.read();    //读入用户名长度
            byte[] idbuf = new byte[bufsize];
            fis.read(idbuf);         //读入用户名字节流
            userinfo.mUserid = new String(idbuf, "UTF-8");  //将字节数组解码为 UTF-8
                                                            //格式的字符串
            //读入用户密码
            bufsize = (byte)fis.read();
            byte[] psdbuf = new byte[bufsize];
            fis.read(psdbuf);        //读入用户密码字节流
            userinfo.mPassword = new String(psdbuf);
            //读入用户电话
            bufsize = (byte)fis.read();
```

```
                byte[] phonebuf = new byte[bufsize];
                fis.read(phonebuf);       //读入用户电话字节流
                userinfo.mUserphone = new String(phonebuf);
                //读入用户地址
                bufsize = (byte)fis.read();
                byte[] addbuf = new byte[bufsize];
                fis.read(addbuf);         //读入用户地址字节流
                userinfo.mUseraddress = new String(addbuf, "UTF-8");
            } catch (Exception e) {}
            return userinfo;
    }
}
```

在 MainActivity 类中添加文件访问对象。

```
public class MainActivity extends Activity
{   …
    //文件访问对象
    private DataFileAccess mDFA = new DataFileAccess(MainActivity.this);
    …
}
```

在 MainActivity 类的 onCreate()函数中添加用户信息读入代码，这样每次程序启动时首先读入用户信息，用它来初始化用户全局变量 g_user。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    …
    mAppInstance.g_user = mDFA.ReadUserInfofromFile("userinfo.txt");
     if (mAppInstance.g_user == null)
        mAppInstance.g_user = new MyUser();         //读入失败则创建新用户
    …
}
```

当用户注册用户名，填写了注册信息后要将它们保存到内部文件中，为此修改 MainActivity 类中的 onActivityResult()函数如下。

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode){
    case REGISTERACTIVITY:
        if (resultCode == Activity.RESULT_OK){
            //获得 RegisterActivity 封装在 intent 中的数据
            MyUser userInfo = new MyUser();
            userInfo.mUserid = data.getStringExtra("user");
```

```
            userInfo.mPassword = data.getStringExtra("password");
            userInfo.mUserphone = data.getStringExtra("phone");
            userInfo.mUseraddress = data.getStringExtra("address");
            //将用户信息保存到默认文件夹中
            String filename = "userinfo.txt";
            mDFA.SaveUserInfotoFile(filename, userInfo);
            //从保存的用户信息文件中读入用户信息到全局变量 g_user
            mAppInstance.g_user = mDFA.ReadUserInfofromFile(filename);
            Toast.makeText(MainActivity.this, "已保存并读取!", Toast.LENGTH_LONG).show();
        }
        break;
    }
}
```

3. 将存储在项目 res/raw 目录中的菜品图片存储到 SD 卡上

首先,在 DataFileAccess 类中添加将资源文件复制到 SD 卡指定目录中的函数。

```
/**
 * 将 raw 文件夹中的资源文件复制到 SD 卡中的指定文件夹中
 * @param resFileId: raw 文件夹中的文件 id 号
 * @param strSDFileName:SD 卡中的文件路径,这里为相对路径
 */
public boolean CopyRawFilestoSD(int resFileId, String strSDFileName)
{
        Resources resources = mContext.getResources();          //获得资源对象
        InputStream inputStream = null;                         //二进制输入流
        try {
            File sdFile = new File(mSDPath + "/" + strSDFileName);
            sdFile.createNewFile();                             //创建新文件
            //判断 SD 文件是否存在、可写,且不是目录
            if (!(sdFile.exists() && sdFile.canWrite()) || sdFile.isDirectory())
                return false;
            //创建文件输出流
            FileOutputStream fos = new FileOutputStream(sdFile);
            //打开资源文件,获得二进制输入流
            inputStream = resources.openRawResource(resFileId);
            byte [] readerbuf = new byte[1024];                 //资源缓冲区
            int readLen = 0;
            while ((readLen = inputStream.read(readerbuf)) != -1) {
                fos.write(readerbuf, 0, readLen);
            }
            fos.flush();                                        //由缓冲区写入 SD 卡
            fos.close();
            inputStream.close();
        }catch (Exception e) {
            return false;
        }
        return true;
}
```

然后，在 MyApplication 类中添加一个全局变量用于指定菜品图片要保存在 SD 卡中的位置。

```
public class MyApplication extends Application                    //该类用于保存全局变量
{
    …
    String g_imgDishImgPath = "Android/data/edu.cqut.mobileorderfood/img";   //菜品图片路径
}
```

最后，在 MainActivity 类中添加将菜品图片从 res/raw 目录复制到 SD 卡指定位置的函数。

```
private boolean CopyDishImagesFromRawToSD()
{
    if (mDFA.SDCardState())            //检查 SD 卡是否可用
    {
        //在 SD 卡中创建存放菜品图像的指定文件夹
        if (!mDFA.isFileExist(mAppInstance.g_imgDishImgPath)) {
            //文件夹不存在，创建文件夹
            mDFA.createSDDir(mAppInstance.g_imgDishImgPath);
        }
        //依次将 raw 文件夹中的菜品图像复制到 SD 卡的指定文件夹中
        String strDishImgName = mAppInstance.g_imgDishImgPath + "/" + "food01gongbaojiding.
jpg";
        if (!(mDFA.isFileExist(strDishImgName)))
            //将 raw 文件夹中的 food01gongbaojiding.jpg 文件复制至 SD 卡指定文件夹中
            mDFA.CopyRawFilestoSD(R.raw.food01gongbaojiding, strDishImgName);
        strDishImgName = mAppInstance.g_imgDishImgPath + "/" + "food02jiaoyanyumi.jpg";
        if (!(mDFA.isFileExist(strDishImgName)))
            //将 raw 文件夹中的 food02jiaoyanyumi.jpg 文件复制至 SD 卡指定文件夹中
            mDFA.CopyRawFilestoSD(R.raw.food02jiaoyanyumi, strDishImgName);
        strDishImgName = mAppInstance.g_imgDishImgPath + "/" + "food03qingzhengwuchangyu.
jpg";
        if (!(mDFA.isFileExist(strDishImgName)))
            //将 raw 文件夹中的 food03qingzhengwuchangyu.jpg 文件复制至 SD 卡指定文件夹中
            mDFA.CopyRawFilestoSD(R.raw.food03qingzhengwuchangyu, strDishImgName);
        strDishImgName = mAppInstance.g_imgDishImgPath + "/" + "food04yuxiangrousi.jpg";
        if (!(mDFA.isFileExist(strDishImgName)))
            //将 raw 文件夹中的 food04yuxiangrousi.jpg 文件复制至 SD 卡指定文件夹中
            mDFA.CopyRawFilestoSD(R.raw.food04yuxiangrousi, strDishImgName);
        return true;
    }
    return false;
}
```

最后，在 MainActivity 类的 onCreate()函数中调用 CopyDishImagesFromRawToSD()函数完成图片复制任务。

5.3 数据库存储

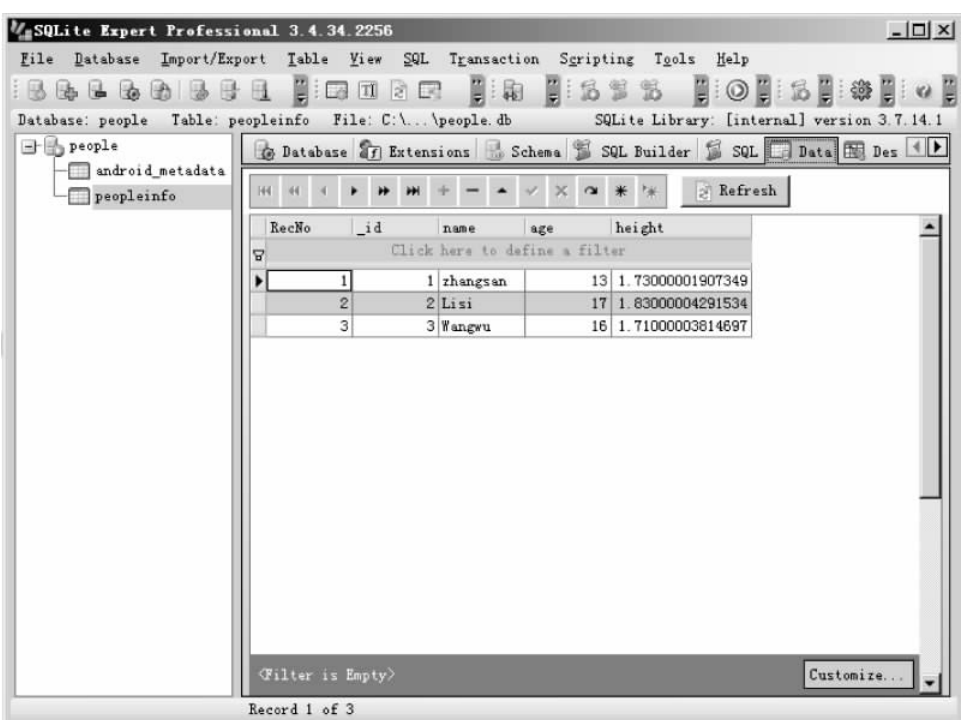
5.3.1 SQLite 简介

SQLite 是一款轻型的关系数据库，是由 D. RichardHipp 发布的开源嵌入式数据库，支持跨平台，最大支持 2048GB 数据，可被所有主流编程语言支持，目前已经在很多嵌入式产品中使用。它占用资源非常低，在嵌入式设备中，可能只需要几百 KB 的内存就够了。

SQLite 数据库管理工具很多，比较常用的有 SQLite Expert Professional，其强大的功能几乎可以在可视化环境下完成所有的数据库操作。另外，Mozilla Firefox——火狐浏览器的免费插件 SQLite Manager 也支持 SQLite 的可视化操作，这两个软件的运行界面如图 5.4 所示。

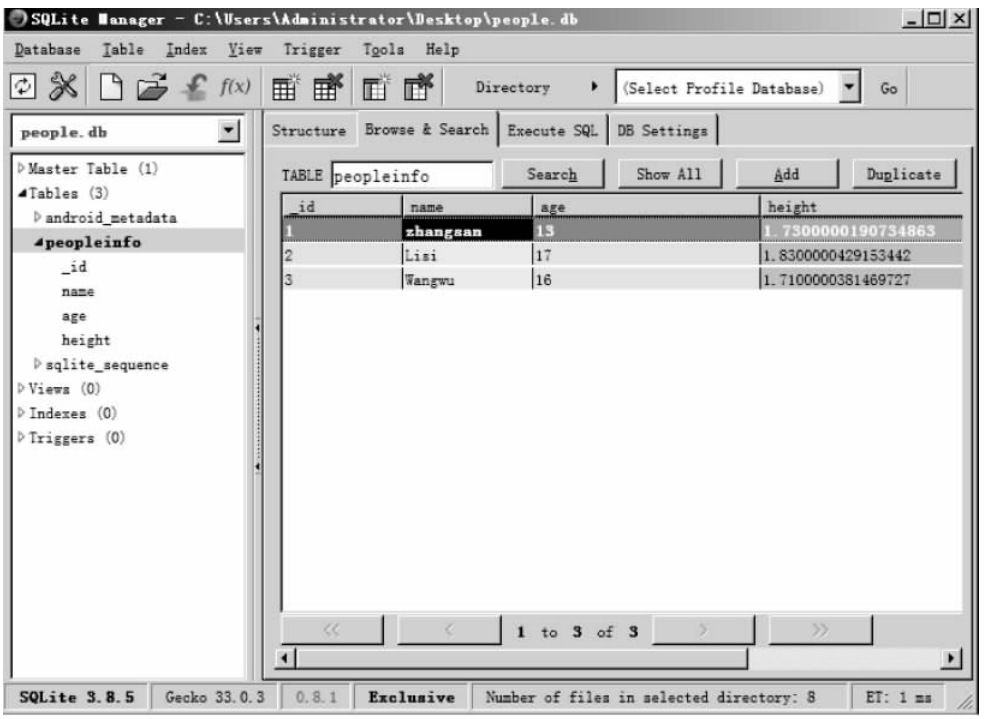
SQLite 的核心大约有 3 万行标准 C 代码，模块化的设计使这些代码非常易于理解。Android 集成了 SQLite 数据库，每个 Android 应用程序都可以使用该数据库。对于熟悉 SQL 语言的开发人员来说，在 Android 中使用 SQLite 也很简单。

Android 系统中，每个应用程序的 SQLite 数据库保存在各自的/data/data/< package name >/databases 目录中，默认情况下，所有数据库都是私有的，仅允许创建数据库的应用程序访问。



(a) SQLite Expert Professional运行界面

图 5.4 两款 SQLite 数据库可视化管理工具



(b) SQLite Manager运行界面

图 5.4 （续）

5.3.2 管理和操作 SQLite 数据库的对象

Android 提供了一个名为 SQLiteDatebase 的类，该类封装了一些数据库的 API，使用它可以对数据库进行添加（Create）、查询（Retrieve）、更新（Update）和删除（Delete）。表 5.2 列出了 SQLiteDatebase 类的常用方法。

表 5.2 SQLiteDatebase 类常用方法

方　法	说　明
openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)	打开或创建数据库
openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)	打开指定的数据库
delete(String table, String whereClause, String[] whereArgs)	删除一条记录
insert(String table,String nullColumnHack,ContentValues values)	插入一条记录
query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	查询一条记录
update(String table, ContentValues values, String whereClause, String[] whereArgs)	修改记录
execSQL(String sql)	执行一条 SQL 语句
close()	关闭数据库

除了 SQLiteDatebase，还有一个类 SQLiteOpenHelper，是 SQLiteDatebase 的辅助类，主要用于创建数据库，并对数据库的版本进行管理。该类是一个抽象类，使用时一般是定义

一个类继承 SQLiteOpenHelper，并实现两个回调方法：OnCreate(SQLiteDatabase db)和 onUpgrade(SQLiteDatabse, int oldVersion, int newVersion)来创建和更新数据库。SQLiteOpenHelper 的方法见表 5.3。

表 5.3 SQLiteOpenHelper 类的常用方法

方 法	说 明
onCreate(SQLiteDatabase db)	首次生成数据库时调用该方法
onOpen(SQLiteDatabase db)	调用已经打开的数据库
onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	升级数据库时调用
getWritableDatabase()	得到可写的数据库，返回 SQLiteDatabase 对象，然后通过对象进行数据库读取操作
getReadableDatabase()	得到可读的数据库，返回 SQLiteDatabase 对象，然后通过对象进行数据库读取操作
close()	关闭数据库，需要强调的是，在每次打开数据库后停止使用时调用，否则会造成数据泄露

5.3.3 数据操作

数据操作包括 3 个层次，依次为：

- 数据库操作：建立或删除数据库。
- 数据表操作：建立、修改及删除数据库中的数据表。
- 数据记录操作：对数据表中的数据记录添加、删除、修改和查询。

为了便于理解，我通过一个例子来说明 SQLite 的数据操作。

【例 5-3】 编写程序演示 SQLite 数据库操作。

建立 SQLiteDemo 的 Android 程序。在该程序中建立一个 people 数据库，该数据库中有一个 peopleinfo 的数据表，该表拥有 4 个字段，分别是 id(整型、主键)、姓名(字符串型)、年龄(整型)和身高(浮点型)。用户在程序中可以完成对数据库的常用操作，如图 5.5 所示。

图 5.5 SQLiteDemo 运行效果

为了实现以上功能，需要编写一个类 DBAdapter 完成数据库及表的建立、更新、删除操作，以及对表中记录的插入、更新、删除、查询操作。

首先定义一个 People 类如下：

```
public class People {
    public int ID =  -1;
```

```
    public String Name;
    public int Age;
    public float Height;
}
```

然后,定义一个数据库类如下:

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
public class DBAdapter
{   private static final String DB_NAME = "people.db";          //数据库名称
    private static final String DB_TABLE = "peopleinfo";        //数据表名称
    private static final int DB_VERSION = 1;                    //数据库版本号
    public static final String KEY_ID = "_id";                  //ID字段名称
    public static final String NAME = "name";                   //姓名字段名称
    public static final String AGE = "age";                     //年龄字段名称
    public static final String HEIGHT = "height";               //身高字段名称
    private SQLiteDatabase db;                                  //people数据库
    private final Context context;
}
```

1. 创建及删除数据库、数据表

创建数据库及其数据表有多种方法,可以应用 SQLiteDatabase 对象 openDatabase()方法及 openOrCreateDatabase()方法,也可以使用 SQLiteHelper 的子类创建数据库,该方法示例如下。

```
/** 静态 Helper 类,用于建立、更新和打开数据库 */
//DBOpenHelper 作为访问 SQLite 的助手类,提供两方面功能:
//(1)通过 getReadableDatabase()和 getWritableDatabase()可以获得 SQLiteDatabase 对象
//(2)提供 onCreate()和 onUpgrade()两个回调函数,允许在创建和升级数据库时,进行自己的操作
  public static class DBOpenHelper extends SQLiteOpenHelper
  {
      //在 SQLiteOpenHelper 的子类中必须有该构造函数
       public DBOpenHelper(Context context, String db_name, CursorFactory factory, int
version)
      { super(context, db_name, factory, version);}
      //创建数据表的 SQL 语句
      private static final String DB_CREATE =
            "create table " + TABLE_NAME
            + " (" + KEY_ID + " integer primary key autoincrement, "//ID号:整型主键字段
            + NAME + " text not null, "                              //姓名:字符串字段
            + AGE + " integer,"                                      //年龄:整型字段
            + HEIGHT + " float);";                                   //身高:浮点型字段
```

```
        @Override
        //该函数在第一次创建数据库时候执行,在第一次得到 SQLiteDatabase 对象的时候,才会调用
        public void onCreate(SQLiteDatabase _db) {
          _db.execSQL(DB_CREATE);
        }
        @Override
        public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
          _db.execSQL("DROP TABLE IF EXISTS " + DB_TABLE);
          onCreate(_db);
      }
  }
```

在 DBAdapter 类中添加 DBOpenHelper 类的成员变量及数据库创建、打开、关闭操作的函数。

```
  public class DBAdapter
  {
      …
      private DBOpenHelper dbOpenHelper;
      public DBAdapter(Context _context) {
          context = _context;
      }
      /** 关闭数据库 */
      public void close() {
          if (db != null){
           db.close();
           db = null;
           }
      }
      /** 创建及打开数据库 */
      public void open() throws SQLiteException {
          //创建一个 DatabaseHelper 对象
          dbOpenHelper = new DBOpenHelper(context, DB_NAME, null, DB_VERSION);
          //只有调用了 DatabaseHelper 对象的 getReadableDatabase()或者 getWritableDatabase()
          //方法才会调用
           //DBOpenHelper 的 onCreate()方法
          try {
              db = dbOpenHelper.getWritableDatabase();
          }catch (SQLiteException ex) {
              db = dbOpenHelper.getReadableDatabase();
          }
      }
      /** 删除数据库 */
      public void delete() throws SQLiteException {
          context.deleteDatabase(DB_NAME);
      }
      /** 创建数据表 */
      public void create_table(String createTableSql) throws SQLiteException {
          db.execSQL(createTableSql);
```

```
    }
    /** 删除数据表 */
    public void create_table(String tableName) throws SQLiteException {
        db.execSQL("DROP TABLE IF EXISTS " + tableName);
    }
}
```

2. 数据记录操作

数据表中的列称为字段，每一行称为记录。对数据表中的数据进行操作处理，主要是对其记录进行操作处理。

对数据记录处理有两种方法，一种是编写一条对记录进行增、删、改、查的 SQL 语句，通过 execSQL()方法来执行。另一种是使用 Android 系统的 SQLiteDatabase 对象的相应方法进行操作。前者容易掌握，下面只介绍使用 SQLiteDatabase 对象操作数据记录的方法。

1）增加记录

新增记录使用 SQLiteDatabase 对象的 insert()方法实现，方法原型为：

```
long insert(String table, String nullColumnHack, ContentValues values)
```

其参数含义如下。

- table：增加记录的数据表。
- nullColumnHack：空列的默认值，通常为 null。
- values：为 ContentValues 对象，即键值对的字段名称，键名为表中字段名，键值为要增加的记录数据值。通过 ContentValues 对象的 put()方法将数据存放到 ContentValues 对象中。
- 返回值：返回插入记录所在行的行号，如果插入失败返回－1。

下面是 DBAdapter 类中的插入记录的方法：

```
public long insert(People people)
{//生成 ContentValues 对象
 ContentValues newValues = new ContentValues();
 //向该对象当中插入键值对，其中键是列名，值是希望插入到这一列的值，值必须和键的类型匹配
 newValues.put(NAME, people.Name);
 newValues.put(AGE, people.Age);
 newValues.put(HEIGHT, people.Height);
 return db.insert(DB_TABLE, null, newValues);
}
```

2）修改记录

修改记录使用 SQLiteDatabase 对象的 update()方法，方法原型为：

```
int update(String table, ContentValues values, String whereClause, String[] whereArgs)
```

其参数含义如下：

- table：修改记录的数据表。

- values：ContentValues 对象，存放已修改数据的对象。
- whereClause：修改数据的条件，相当 SQL 语句的 where 子句，null 表示更新所有记录。
- whereArgs：修改数据值的数组，null 表示更新整行。
- 返回值：返回修改记录个数。

下面是使用 update()函数修改记录的方法：

```
//相当于执行 SQL 语句中的 update 语句: update table_name SET XXCOL = XXX WHERE XXCOL = XX...
public long updateOneData(long id , People people)
{    ContentValues updateValues = new ContentValues();
     updateValues.put(NAME, people.Name);
     updateValues.put(AGE, people.Age);
     updateValues.put(HEIGHT, people.Height);
     return db.update(DB_TABLE, updateValues, KEY_ID + " = " + id, null);
}
```

3）删除记录

删除记录使用 SQLiteDatabase 对象的 delete()方法，方法原型为：

```
int delete(String table, String whereClause, String[] whereArgs)
```

其参数含义如下：

- table：删除记录的数据表。
- whereClause：删除数据的条件，相当 SQL 语句的 where 子句，null 表示删除所有记录。
- whereArgs：删除条件的数组。
- 返回值：返回删除记录的个数。

下面是 DBAdapter 类中的删除记录的方法：

```
public long deleteAllData()
{    return db.delete(DB_TABLE, null, null);
}
public long deleteOneData(long id)
{    return db.delete(DB_TABLE, KEY_ID + " = " + id, null);
}
```

4）查询记录

查询记录使用 SQLiteDatabase 对象的 query()方法，方法原型为：

```
Cursor query(String table, String[] columns, String selection, String[] selectionArgs,
            String groupBy, String having, String orderBy)
```

其参数含义如下：

- table：查询记录的数据表。
- columns：查询的字段，如为 null 表示所有字段。

- selection：查询条件，可以使用通配符“?”。
- selectionArgs：参数数组，用于替换查询条件中的“?”。
- groupBy：查询结果，按指定字段分组。
- having：限定分组的条件。
- orderBy：查询结果的排序条件。
- 返回值：返回查询结果。

用 query()方法查询的数据均封装在查询结果 Cursor 对象中，Cursor 相当于 SQL 语句中的 resultSet 结果集上的一个游标，可以在结果集中向前、向后移动，并能够获取结果集的属性名称和序号，具体的方法见表 5.4。

表 5.4　Cursor 类的公有方法

方　　法	说　　明
moveToFirst()	将游标移动到结果集的第一行记录
moveToLast()	将游标移动到结果集的最后一行记录
moveToNext()	将游标移动到结果集的下一行记录
moveToPrevious()	将游标移动到结果集的上一行记录
moveToPosition(int position)	将游标移动到结果集的指定位置
int getCount()	获得结果集的记录数
int getPosition()	获得游标的当前位置
int getColumnIndexOrThrow(String columnName)	返回指定属性名称的序号，如果属性不存在，则产生异常
String getColumnName(int columnIndex)	返回指定序号的属性名称
String[] getColumnNames()	返回属性名称的字符串数组
int getColumnIndex(String columnName)	返回指定属性名称的序号

下面是 DBAdapter 类中的查询记录的方法：

```
public People[ ] queryAllData()
{   Cursor results = db.query(DB_TABLE, new String[ ] { KEY_ID, NAME, AGE, HEIGHT}, null,
                null, null, null, null);
    return ConvertToPeople(results);
}

public People[ ] queryOneData(long id)
{   Cursor results = db.query(DB_TABLE, new String[ ] { KEY_ID, NAME, AGE, HEIGHT},
                 KEY_ID + "=" + id, null, null, null, null);
    return ConvertToPeople(results);
}

private People[ ] ConvertToPeople(Cursor cursor)
{   int resultCounts = cursor.getCount();
    if (resultCounts == 0 || !cursor.moveToFirst())
        return null;
    People[ ] peoples = new People[resultCounts];
    for (int i = 0 ; i<resultCounts; i++)
```

```
        {   peoples[i] = new People();
            peoples[i].ID = cursor.getInt(0);
            peoples[i].Name = cursor.getString(cursor.getColumnIndex(KEY_NAME));
            peoples[i].Age = cursor.getInt(cursor.getColumnIndex(KEY_AGE));
            peoples[i].Height = cursor.getFloat(cursor.getColumnIndex(KEY_HEIGHT));
            cursor.moveToNext();          //将游标向下移动一位
        }
        return peoples;
    }
```

最后，给出 SQLiteDemo 的 Activity 页面的代码——SQLiteDemoActivity.java 文件的内容，在该 Activity 中通过调用上面 DBAdapter 类的方法完成数据库的创建、更新及各项数据操作。

```
public class SQLiteDemoActivity extends Activity {
    private DBAdapter dbAdepter ;
    private EditText nameText;
    private EditText ageText;
    private EditText heightText;
    private EditText idEntry;
    private TextView labelView;
    private TextView displayView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        nameText = (EditText)findViewById(R.id.name);
        ageText = (EditText)findViewById(R.id.age);
        heightText = (EditText)findViewById(R.id.height);
        idEntry = (EditText)findViewById(R.id.id_entry);
        labelView = (TextView)findViewById(R.id.label);
        displayView = (TextView)findViewById(R.id.display);
        Button addButton = (Button)findViewById(R.id.add);
        Button queryAllButton = (Button)findViewById(R.id.query_all);
        Button clearButton = (Button)findViewById(R.id.clear);
        Button deleteAllButton = (Button)findViewById(R.id.delete_all);
        Button queryButton = (Button)findViewById(R.id.query);
        Button deleteButton = (Button)findViewById(R.id.delete);
        Button updateButton = (Button)findViewById(R.id.update);
        addButton.setOnClickListener(addButtonListener);
        queryAllButton.setOnClickListener(queryAllButtonListener);
        clearButton.setOnClickListener(clearButtonListener);
        deleteAllButton.setOnClickListener(deleteAllButtonListener);
        queryButton.setOnClickListener(queryButtonListener);
        deleteButton.setOnClickListener(deleteButtonListener);
        updateButton.setOnClickListener(updateButtonListener);
        dbAdepter = new DBAdapter(this);
        dbAdepter.open();
    }
```

```
        OnClickListener addButtonListener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                People people = new People();
                people.Name = nameText.getText().toString();
                people.Age = Integer.parseInt(ageText.getText().toString());
                people.Height = Float.parseFloat(heightText.getText().toString());
                long columm = dbAdepter.insert(people);
                if (columm == -1 ){
                    labelView.setText("添加过程错误!");
                } else {
                    labelView.setText("成功添加数据,ID: " + String.valueOf(columm));

                }
            }
        };
        OnClickListener queryAllButtonListener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                People[] peoples = dbAdepter.queryAllData();
                if (peoples == null){
                    labelView.setText("数据库中没有数据");
                    return;
                }
                labelView.setText("数据库: ");
                String msg = "";
                for (int i = 0 ; i < peoples.length; i++){
                    msg += peoples[i].toString() + "\n";
                }
                displayView.setText(msg);
            }
        };
        OnClickListener clearButtonListener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                displayView.setText("");
            }
        };
        OnClickListener deleteAllButtonListener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                dbAdepter.deleteAllData();
                String msg = "数据全部删除";
                labelView.setText(msg);
            }
        };
        OnClickListener queryButtonListener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                int id = Integer.parseInt(idEntry.getText().toString());
```

```
            People[] peoples = dbAdepter.queryOneData(id);
            if (peoples == null){
                labelView.setText("数据库中没有 ID 为" + String.valueOf(id) + "的数据");
                return;
            }
            labelView.setText("数据库: ");
            displayView.setText(peoples[0].toString());
        }
    };
    OnClickListener deleteButtonListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            long id = Integer.parseInt(idEntry.getText().toString());
            long result = dbAdepter.deleteOneData(id);
            String msg = "删除 ID 为" + idEntry.getText().toString() + "的数据" + (result>
0?"成功":"失败");
            labelView.setText(msg);
        }
    };
    OnClickListener updateButtonListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            People people = new People();
            people.Name = nameText.getText().toString();
            people.Age = Integer.parseInt(ageText.getText().toString());
            people.Height = Float.parseFloat(heightText.getText().toString());
            long id = Integer.parseInt(idEntry.getText().toString());
            long count = dbAdepter.updateOneData(id, people);
            if (count == -1 ){
                labelView.setText("更新错误!");
            } else {
                labelView.setText("更新成功,更新数据" + String.valueOf(count) + "条");

            }
        }
    };
}
```

5.3.4 用数据库管理“移动点餐系统”中的菜单

“移动点餐系统”中的 SQLite 菜单数据库名称为 dishes.db,其中包含一个数据表 dishinfo 存储菜品信息,该表拥有 4 个字段,分别是_id(菜品 ID,整型,主键)、name(菜名,字符串型)、imgname(菜品图片名,字符串型)、price(价格,浮点型)。注意,数据库中保存的不是菜品图片,而是菜品图片的文件名,图片仍然保存在 SD 卡中,访问图片时根据程序中设置的图片目录及数据库中的文件名进行查找。

为了在“移动点餐系统”中使用 SQLite 数据库存储菜单,采用以下步骤修改程序,粗体部分为增加或者修改内容。

（1）在 Dish 类中增加一个成员变量保存图片文件的名称。

```
public class Dish
{
    public int mId = -1;                    //菜品 ID
    public String mName;                    //菜名
    public int mImage;                      //菜品图像
    public String mImageName;               //菜品图像的文件名
    public float mPrice;                    //价格
}
```

（2）修改主页面 MainActivity 类中的 FillDishesList()方法，在输出的 ArrayList < Dish >列表的各 Dish 元素中增加存储菜品图像文件名的部分。

```
private ArrayList<Dish> FillDishesList()
{
    String imgPath = mDFA.SDCardPath() + "/" + mAppInstance.g_imgDishImgPath + "/";
    ArrayList<Dish> theDishesList = new ArrayList<Dish>();
    Dish theDish = new Dish();
    //添加菜品
    theDish.mId = 1001;
    theDish.mName = "宫保鸡丁";
    theDish.mPrice = (float) 20.0;
    theDish.mImage = (R.raw.food01gongbaojiding);
    theDish.mImageName = imgPath + "food01gongbaojiding.jpg";
    theDishesList.add(theDish);

    theDish = new Dish();
    theDish.mId = 1002;
    theDish.mName = "椒盐玉米";
    theDish.mPrice = (float) 24.0;
    theDish.mImage = (R.raw.food02jiaoyanyumi);
    theDish.mImageName = imgPath + "food02jiaoyanyumi.jpg";
    theDishesList.add(theDish);

    theDish = new Dish();
    theDish.mId = 1003;
    theDish.mName = "清蒸武昌鱼";
    theDish.mPrice = (float) 48.0;
    theDish.mImage = (R.raw.food03qingzhengwuchangyu);
    theDish.mImageName = imgPath + "food03qingzhengwuchangyu.jpg";
    theDishesList.add(theDish);

    theDish = new Dish();
    theDish.mId = 1004;
    theDish.mName = "鱼香肉丝";
    theDish.mPrice = (float) 20.0;
    theDish.mImage = (R.raw.food04yuxiangrousi);
    theDish.mImageName = imgPath + "food04yuxiangrousi.jpg";
```

```
        theDishesList.add(theDish);
        return theDishesList;
    }
```

（3）将前面的数据库管理类 DBAdapter 添加到本程序中，根据 dishes.db 内容对该类进行修改，使之适合菜品数据库。

```
public class DBAdapter
{
    private static final String DB_NAME = "dishes.db";
    private static final String DB_TABLE = "dishinfo";
    private static final int DB_VERSION = 1;
    public static final String KEY_ID = "_id";
    public static final String KEY_NAME = "name";
    public static final String KEY_IMGNAME = "imgname";
    public static final String KEY_PRICE = "price";
    private SQLiteDatabase db;
    private final Context context;
    private DBOpenHelper dbOpenHelper;
    public DBAdapter(Context _context)
    { context = _context;
    }
    /** Close the database */
    public void close()
    {   if (db != null)
        {   db.close();
            db = null;
        }
    }
    /** Open the database */
    public void open() throws SQLiteException
    {   dbOpenHelper = new DBOpenHelper(context, DB_NAME, null, DB_VERSION);
        try {
            db = dbOpenHelper.getWritableDatabase();
        }catch (SQLiteException ex) {
            db = dbOpenHelper.getReadableDatabase();
        }
    }
    public long insert(Dish dish)
    { ContentValues newValues = new ContentValues();
        newValues.put(KEY_ID, dish.mId);
        newValues.put(KEY_NAME, dish.mName);
        newValues.put(KEY_IMGNAME, dish.mImageName);
        newValues.put(KEY_PRICE, dish.mPrice);
        return db.insert(DB_TABLE, null, newValues);
    }
    public ArrayList<Dish> queryAllData()
    {   Cursor results = db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME,
                        KEY_IMGNAME, KEY_PRICE}, null, null, null, null, null);
```

```
        return ConvertToDishes(results);
    }
    public Dish queryOneData(long id)
    {   Cursor results = db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME,
              KEY_IMGNAME, KEY_PRICE}, KEY_ID + " = " + id, null, null, null, null);
        return ConertToDish(results);
    }
    private Dish ConertToDish(Cursor cursor)
    {
        int resultCounts = cursor.getCount();
        if (resultCounts == 0 || !cursor.moveToFirst()){
            return null;
        }
        Dish theDish = new Dish();
        theDish.mId = cursor.getInt(0);
        theDish.mName = cursor.getString(cursor.getColumnIndex(KEY_NAME));
        theDish.mImageName = cursor.getString(cursor.getColumnIndex(KEY_IMGNAME));
        theDish.mPrice = cursor.getFloat(cursor.getColumnIndex(KEY_PRICE));
        return theDish;
    }
    private ArrayList<Dish> ConvertToDishes(Cursor cursor)
    {   int resultCounts = cursor.getCount();
        if (resultCounts == 0 || !cursor.moveToFirst()){
            return null;
        }
        ArrayList<Dish> dishes = new ArrayList<Dish>();
        for (int i = 0 ; i<resultCounts; i++){
            Dish theDish = new Dish();
            theDish.mId = cursor.getInt(0);
            theDish.mName = cursor.getString(cursor.getColumnIndex(KEY_NAME));
            theDish.mImageName = cursor.getString(cursor.getColumnIndex(KEY_IMGNAME));
            theDish.mPrice = cursor.getFloat(cursor.getColumnIndex(KEY_PRICE));
            dishes.add(theDish);
            cursor.moveToNext();
        }
        return dishes;
    }
    public long deleteAllData()
    {   return db.delete(DB_TABLE, null, null);
    }
    public long deleteOneData(long id)
    {   return db.delete(DB_TABLE, KEY_ID + " = " + id, null);
    }
    public long updateOneData(long id , Dish dish)
    {   ContentValues updateValues = new ContentValues();
        updateValues.put(KEY_NAME, dish.mName);
        updateValues.put(KEY_IMGNAME, dish.mImageName);
        updateValues.put(KEY_PRICE, dish.mPrice);
        return db.update(DB_TABLE, updateValues, KEY_ID + " = " + id, null);
    }
```

```
    /** 静态 Helper 类,用于建立、更新和打开数据库 */
    private static class DBOpenHelper extends SQLiteOpenHelper
    {
        public DBOpenHelper(Context context, String name, CursorFactory factory, int version)
{
          super(context, name, factory, version);
        }
        private static final String DB_CREATE = "create table " +
          DB_TABLE + " (" + KEY_ID + " integer primary key, " +
          KEY_NAME + " text not null, " + KEY_IMGNAME + " text," + KEY_PRICE + " float);";
          @Override
        public void onCreate(SQLiteDatabase _db)
        { _db.execSQL(DB_CREATE);
        }
        @Override
        public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion)
        { _db.execSQL("DROP TABLE IF EXISTS " + DB_TABLE);
            onCreate(_db);
        }
    }
    /** 创建菜品数据库 */
    //将保存在内存 dishes 数组中的菜单保存在菜品数据库中
    public boolean FillDishTable(ArrayList<Dish> dishes)
    {
            int s = dishes.size();       //得到列表元素个数
            for (int i = 0; i < s; i++)
            {   Dish theDish = dishes.get(i);
                if (insert(theDish) == -1)
                    return false;
            }
            return true;
    }
    /** 取出菜品数据库中的数据 */
    //将保存在菜品数据库中的数据输出成 ArrayList<Map<String, Object>>格式的数据
    public ArrayList<Map<String, Object>> getDishData()
    {
        //将菜品从数据库中填充进 ArrayList 列表
        ArrayList<Dish> dishes = queryAllData();
        ArrayList<Map<String, Object>> foodddata = new ArrayList<Map<String,Object>>();
        //再将菜品从 ArrayList 中填充进 foodinfo 列表
        int s = dishes.size();           //得到菜品数量
        for (int i = 0; i < s; i++)
        {   Dish theDish = dishes.get(i);      //得到当前菜品
            Map<String, Object> map = new HashMap<String, Object>();
            map.put("dishid", theDish.mId);
            map.put("image", theDish.mImageName);
            map.put("title", theDish.mName);
            map.put("price", theDish.mPrice);
            foodddata.add(map);
        }
```

```
            return fooddata;
        }
    }
```

然后在 MyApplication 类中添加一个数据库管理的成员变量。

```
public class MyApplication extends Application          //该类用于保存全局变量
{
    …
    public DBAdapter g_dbAdepter = null;                //数据库辅助对象
}
```

(4) 在 MainActivity 类的 onCreate()方法中添加将菜单保存进 dishes.db 数据库中的代码。为了便于和前面的代码衔接,这里没有采用直接将菜品信息填充进数据库的方法,而是仍沿袭以前代码将菜品保存在 ArrayList<Dish>列表中,然后再由 ArrayList<Dish>列表填充进数据库中。

```
@Override
protected void onCreate(Bundle savedInstanceState)
{ …
    mAppInstance.g_orders = new ArrayList<Order>();   //创建订单列表
    CopyDishImagesFromRawToSD();                      //将 RAW 文件夹中的菜品图像复制到
                                                      //SD 卡的指定文件夹中
    mAppInstance.g_dbAdepter = new DBAdapter(this);
    mAppInstance.g_dbAdepter.open();
    mAppInstance.g_dbAdepter.deleteAllData();         //清除原有菜品数据
    ArrayList<Dish> dishes = FillDishesList();        //将菜品列表保存在内存 dishes 表中
    //将菜品从 dishes 表中填充进数据库
    mAppInstance.g_dbAdepter.FillDishTable(dishes);
}
```

(5) 修改程序的 CaipinActivity 类的 onCreate()方法,让菜单列表从菜品数据库中加载。

```
@Override
protected void onCreate(Bundle savedInstanceState)
{   …
    final MyApplication appInstance = (MyApplication)getApplication();
    mfoodinfo = appInstance.g_dbAdepter.getDishData();
}
```