第4章

智能物流管理系统项目——数据存储功能实现

学习目标

【知识目标】

- 了解数据存储的几种方式
- 掌握 File 方式存储
- 掌握 SharePreferences 方式存储
- 掌握 SharePreferences 方式读取
- 掌握 SQLite 在 Android 里的使用
- 掌握 SQLite 的增删改查操作
- 掌握 SQLiteOpenHelper
- 掌握 ContentProvider 原理
- 掌握理解通用资源标识符(URL)
- 掌握 ContentResolver
- 掌握系统本地的 ContentProvider

【能力目标】

掌握 SharePreferences 方式存储

掌握 SharePreferences 方式读取

掌握 SQLite 在 Android 里的使用

4.1 任务描述

实现“智能物流管理系统”项目中用户个人信息等数据存储功能。

4.2 知识准备

作为一个完整的应用程序,数据存储操作是必不可少的。Android 系统一共提供了五种数据存储方式。分别是:SharePreference、SQLite、Content Provider 和 File。本章将

对这五种数据存储方式进行详细介绍。

4.2.1 数据存储简介

数据存储是 Android 平台上非常重要的功能，各个应用存储的数据或文件是私有的，在默认情况下，只有该应用本身能够访问其存储的数据资源。Android 提供的存储方式有 File，SharedPreferences，SQLite 数据库，ContentProvider 和网络存储五种。

4.2.2 File

1. 存储至默认文件夹

一般情况下，文件都默认存储在"/data/data/<包名>/files/"下，其实现代码如下：

```
button1.setOnClickListener(new OnClickListener()
        {
          public void onClick(View v){
            byte[]  buf = input.getText().toString().getBytes();
            try{
              saveFile(buf);
            }catch(IOException e)
                {
                  e.printStackTrace();
                }
            }
        });
    }
    private void saveFile(byte[] buf) throws IOException{
        FileOutputStream fos =
        openFileOutput("text.txt",Context.MODE_APPEND);
        fos.write(buf);
        fos.close();
    }
```

可以在命令行用 adb shell 命令登录，进入"/data/data/<包名>/files/"目录，用"cat test.txt"命令可以查看到刚才输入的内容。

2. 存储至指定文件夹

存储至指定文件夹的代码如下，只需要对前面的代码做一些修改：

```
private void savaFile(){
  try{
      File textFile = new File("/data/data/cn.lcdi.test/test.txt");
      FileOutputStream fos = new FileOutputStream(textFile);
```

```
            fos.write(buf);
            fos.close();
        }
    catch(Exception e){}
    }
```

需要注意的是，testFile.txt 不能创建在“/data/data/cn.lcdi.test/”之外的地方，因为 Android 的安全机制决定了在默认情况下，一个应用只能访问自己的应用资源。

3. 存储至 SD 卡

在前面的基础上我们再做一些修改，将文件存储在 SD 卡上，实现代码如下：

```
private void savaFileToSD(byte[] buf) throws IOException{
    String path = EnvironmentgetDownloadCacheDirectory().getPath();
    File textFile = new File(path + "/test.txt");
    FileOutputStream fos  = new FileOutputStream(textFile);
    fos.write(buf);
    fos.close();
}
```

存储完成后，在 SD 卡对应的目录“/sdcard”下会生成 test.txt 文件。

上面的内容介绍了如何使用文件存储数据，在存储过程中还应注意存储目录和文件编码的问题，对于中文字符应该尽量使用 UTF－8 格式存储。

如下实例，点击第一个 Button 保存 EditText 内的内容，点击第二个 Button 清空 EditText 内的内容并且保存，点击第三个 Button 读取文件内容并显示在屏幕上。

```
public class MainActivity extends Activity {
    private EditText editText;
    private TextView textView;
    private Button buttonSave,buttonReset,buttonRead;
    private String PATH = "/sdcard/testFile.txt";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        FileOutputStream outStream;
        try { //先写入一些信息
outStream = this.openFileOutput(“test.txt”,Context.MODE_PRIVATE);
            outStream.write("Hello World!".getBytes());
            outStream.close();
          } catch (FileNotFoundException e1) {
            //TODO Auto-generated catch block
```

```
                e1.printStackTrace();
            } catch (IOException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();          }
            editText = (EditText)findViewById(R.id.et);
            buttonSave = (Button)findViewById(R.id.bt1);
            buttonReset = (Button)findViewById(R.id.bt2);
            buttonRead = (Button)findViewById(R.id.bt3);
            textView = (TextView)findViewById(R.id.show);
            //保存
    buttonSave.setOnClickListener(new OnClickListener()
    {
      @Override
      public void onClick(View v) {
      // TODO Auto-generated method stub
      SaveFile(PATH,editText.getText().toString());
      }
    });
buttonReset.setOnClickListener(new OnClickListener(){
      @Override
      public void onClick (View v){
      editText.setText("");
      SaveFile(PATH,"");
     }
    });
      //读取
    buttonRead.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick (View v){
          String str = ReadFile(PATH);
          showToast(str);
          textView.setText(str);
          textView.setText(str);
        }
    });
    //读取文件的方法
    private String ReadFile(String filepath){
        //保存读取文本的字符串
```

```
    String currentStr = "";
    //创建新的 BufferedReader 对象
    BufferedReader file;
    try{
        file = new BufferedReader(new FileReader(filepath));
      }catch(Exception e)
      {
        return null;
      }
    while(true){
    try{
        //读取一行数据并保存到 currentStr 变量
        String temp = file.readLine();
        if(temp = = null)
        break;
        if(currentStr! = "")
        currentStr + = "\n";
        currentStr + = temp;
        }catch(Exception e)
        {
          return null;
        }
}
    return currentStr;
}
//保存至文件的方法
private boolean SaveFile(String filepath,String content) {
    BufferedWriter file;
    if(filepath = = null||content = = null)
    {
      return false;
    }
    try{
        file = new BufferedWriter(new FileWriter(filepath));
        file.write(content);
        file.flush();
      }catch(Exception e)
      {
```

```
            System.out.println(e.getMessage());
        }
    }
    //显示 Toast 的方法
    private void showToast(String str){
        Toast.makeText(this,str,Toast.LENGTH_SHORT).show();
    }
}
```

4.2.3 SharedPreferences

1. SharedPreferences 简介

SharedPreferencess 提供了一种基于“名一值”对形式的键值二元组的轻量级数据存储方式，通过 SharedPreferences，运行于统一 Context 下的应用程序可以共享其中的数据。

SharedPreferences 支持的数据类型有 boolean，String，float，long 和 integer，非常适合用于存储默认值，实例变量，UI 状态以及用户设置。此外，SharedPreferences 也常用于应用程序各个控件之间共享设置的情况。

2. SharedPreferences 读写操作

每个应用程序都有一个 SharedPreferences 对象，可以通过 getSharePreferences()方法来获取该对象，然后对其中的数据进行读写：

public abstract SharedPreferences getSharedPreferences (String name,int mode)

其权限可以通过参数 mode 来设置，可取如下 3 个值：

(1)MODE_PRIVATE：应用程序私有，只有当前程序可访问；

(2)MODE_WORLD_READABLE：其他程序可读；

(3)MODE_WORLD_WRITEABLE：其他程序可写。

在获取 SharedPreferences 对象后，对其内容的修改还要依靠 Editor 类的方法，如 putBoolean，putFloat，putInt 等，其参数分别对应于 name/value 对的名称(name)和值(value)。需要注意的是，在赋值完成后，必须用 Editor 对象的 commit()方法执行写入操作. 否则修改无效。

sharedPreferences 对象的创建和修改的代码如下：

```
public static final String SETTING = "LOGIN";//定义 SharePreferencess 数据 ID 名
public void save(){
        int mode = Activity.MODE_PRIVATE;//设定权限为私有
        //获取 SharePreferencess 对象
        SharedPreferences settings = getSharedPreferences(SETTING,mode);
        SharedPreferences.Editor editor = settingsedit();//调用 Editor 类
```

```
editor.putString("username","cool boy");//添加 username 数据
editor.putInt("age",21);//添加 age 数据
editor.putBoolean("male",true);//添加 male 数据
editor.commit();//完成写入
}
```

SharedPreferences 类提供了读写数据的方法，如 getBoolean，getFloat 及 getInt 等，他们均有两个参数，第一个参数为 name/value 对的名称(name)，第二个参数为如果指定名称的值对不存在时方法返回的默认取值. 当指定名称存在时，方法的返回值是 name/value 对的值(value)。

SharedPreferences 的数据读取代码如下：

```
public void load(){
    int mode = Activity.MODE_PRIVATE;
    SharedPreferences settings = getSharedPreferences(SETTING,mode);
    editor.putString("username","nobody");//获取 username 对应值
    editor.putInt("age", 0);//获取 age 对应值
    editor.putBoolean("male",true);//获取 male 对应值
    }
```

如下例所示：

Ex06_2 类创建了 SharedPreferences 对象，并存储了 name，password 两个 String 型的数据和一个 Boolean 型的 isvip 数据。需要注意的是存储操作必须获取 Editor 对象，然后通过 commit 方法提交改变。

```
public class Ex06_2 extends Activity{
public Button button;
public EditText editText_name;//用户名
public EditText editText_password;//用户密码
public CheckBox checkBox;//判断是否 VIP
public boolean isVIP;
/ * * Called when the activity is frist created. * /
@Override
public void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstancestate);
setContentview(R.layout.main);
Button = (Button)findViewById(R.id.bt);
editText_name = (EditText)findViewById(R.id.name);
editText_password = (EditText)findViewById(R.id.password);
checkBox = (CheckBox)findViewById(R.id.VIP);
button.setOnClickListener(new Button.OnClickListener){
```

```
public void noClick(View v){
//创建 SharedPreferences 对象,并限定 MODE_PRIVATE
SharedPreferences sp = getSharedPerferences("LOGIN",Activity.MODE_PRIVATE);
//获取 Editor 对象
SharedPreferences.Editor editor = sp.edit();
editor.PutString("name"editText_name.getText().toString());
editor.PutSrtring("password",editText_password.getText().toSrring());
editor.putBoolean("isvip",isvip);
//提交
Editor.commit();
//跳转到 Result 类
Intent i = new Intent();
class<Result> a = Result.class;
i.setClass(Ex06_2.this,a);
startActivity(i);
//清空输入框
editText_name.setText("");
editText_password.setText("");
checkBox.setChecked(flase);
}}};
}
```

Result 类实现了读取 SharedPreferences 对象里的数据,和存储时不一样的是,读取的时候不需要获取 Editor 对象,也不需要执行 commit 操作。

```
public class Result extends Activity{
public TextView textView1,textView2,textView3;
public Button button;
/ * * Called when the activity is frist created. * /
@Override
public void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.result);
textView1 = (TextView)findViewById(R.id.tv1);
textView2 = (TextView)findViewById(R.id.tv2);
textView3 = (TextView)findViewById(R.id.tv3);
Button = (Button)findViewById(R.id.bt);
Button.setOnClickListener(new Button.OnClickListener(){
@Override
```

```
ublic void onClick(View v){
SharedPreferences sp = getSharedPreferences("LOGIN",Activity.MODE_PRIVATE);
textView1.setText("你的用户名:" + sp.getString("name","null"));
textView2.setText("你的密码:" + sp.getString("name","000"));
textView3.setText("是否 VIP:" + sp.getString("isvip",false));
}}});
}
}
```

布局文件 activity_main.xml：

```
<? xml version = "1.0" encoding = "utf - 8"? >
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical">
    <LinearLayout
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:orientation = "horizontal">
        <TextView
          android:layout_width = "wrap_content"
          android:layout_height = "wrap_content"
          android:text = "登录名:"/>
        <EditText
          android:id = "@ + id/name"
          android:layout_width = "200px"
          android:layout_height = "wrap_content"/>
    </LinearLayout>
    <LinearLayout
        android:orientation = "horizontal"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content">
        <TextView
          android:layout_width = "wrap_content"
          android:layout_height = "wrap_content"
          android:text = "密 码:"/>
        <EditText
          android:id = "@ + id/password"
          android:layout_width = "200px"
```

```
            android:layout_height = "wrap_content"
            android:password = "true"/>
        <CheckBox
            android:id = "@ + id/vip"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "是否 vip 用户"/>
        <Button
            android:id = "@ + id/bt"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "提交"/>
    </LinearLayout>
</LinearLayout>
```

布局文件 Result. xml：

```
</LinearLayout
xmlns:android = "http://schemas.android.com/apk/res/android"
Android:orientation = "vertical"
Android:layout_width = "wrap_content"
Android:layout_heightt = "wrap_content">
<Textview Android:id = "@ + id/tv1"
Android:layout_width = "wrap_content"
Android:layout_height = "wrap_content"
Android:text = "你的用户名:"/>
<Textview Android:id = "@ + id/tv2"
Android:layout_width = "wrap_content"
Android:layout_height = "wrap_content"
Android:text = "你的密码:"/>
<Textview Android:id = "@ + id/tv3" Android:layout_width = "wrap_content"
Android:layout_height = "wrap_content"
Android:text = "是否 vip:"/>
<Button Android:id = "@ + id/bt"
Android:layout_width = "wrap_content"
Android:layout_height = "wrap_content"
Android:text = "获取信息"/>
</LinearLayout>
```

4.2.4　SQLite 数据库

1. SQLite 数据库简介

自从出现商业应用程序以来，数据库就成为软件应用程序的主要组成部分。与数据库管理系统一样，它们也变得非常庞大，并占用相当多的系统资源，增加管理的复杂性。随着软件应用逐渐模块化，一种新型数据库与大型复杂的传统数据库系统相比，更适应这种变化。嵌入式数据库直接在应用程序中运行，资源占用非常少。

SQLite 是一种在嵌入式系统中很常见的数据库，而且所有的数据都存储在一个文件中。SQLite 是一种轻量级的数据库，具有简洁的 SQL 访问界面和相当快的访问速度，而且相对于其他数据库来说，占用的内容空间较少。在 Android 平台上，SQLite 数据库可以用来存储应用程序中使用到的数据。在默认情况下，每个应用所创建的数据库都是私有的，其名字也是唯一的，每个应用空间的数据库无法相互访问。Android 平台提供了一个完整的 SQLite 数据库接口，各应用生成的数据库在“/data/data/<包名>/database”目录下。需要注意的是，为保证数据库的检索效率，并保持较小的体积，数据库中不应该保存较大的文件。

在 SQLite 中，我们可以使用 SQL 语句来查询(SELECT)，插入(INSERT)，修改(UPDATE)，删除(DELETE)，定义数据格式(CREATE TABLE)等。要了解更多的 SQL 数据库查询语言的详细内容，请参考专门 SQL 语言书籍。

2. 查看模拟目录

在模拟器打开的情况下，打开命令行，找到安装 Android SDK 的文件夹，并进入“tools”目录。在“tools”目录中可以找到“adb”这个命令行工具。输入“adb shell”命令后，进入“/data/data”目录。

进入“/data/data”目录后，可以使用“ls”命令来查看模拟器中所有程序数据列表。找到应用程序的包名称后，使用“cd”命令来进入具体某个项目的数据目录。如下所示：

```
E:\android\tools>adb shell
Error: device not found
E:\android\tools>adb shell
# cd data/data/
cd data/data/
# ls
ls
com.android.alarmclock
com.android.calculator2
com.android.customlocale
com.android.providers.contacts
com.android.providers.downloads
```

```
com.android.email
```

3. 手动创建数据库

SQLite 数据库需要放在目录中的“databases”文件夹里，我们可以使用“adb shell”中的“mkdir”命令来创建这个文件夹。

```
#ls
Lib
#mkdir databases
#ls
Databases
lib
```

现在有了用来存放数据库的文件夹，可以开始创建数据库了。我们使用“sqlite3”命令来创建数据库。进入“databases”目录，使用“sqlite3”命令创建一个名为“notes.db”的文件。

```
# cd databases
# sqlite3 notes.db
SQLite version 3.5.9
Enter".help"for instructions
sqlite>
```

现在 notes.db 存储了所有数据，但是这个文件的内容还是空的，里面还没有存储任何数据表或记录。

当我们执行“sqlite3”命令后，屏幕上会显示当前的“SQLite”版本号与一些提示消息。同时，命令提示符号也从“#”换成“sqlite>”，表示已进入“SQLite”的互动模式中。我们可以在此模式下对数据表及其中的条目(数据字段)做添加，删除，修改，查询等操作。

4. 创建数据表

在 SQLite 数据库中，数据是以“数据表”的形式保存的。“数据表”的概念如同表格软件一样(比如 microsoft Excel)以“列”(Column)，“行”(Row)来构成一个个表格。数据表中保存了数据记录及数据记录之间的关系。如通讯录以“姓名”，“电话”，“地址”等表示数据记录之间的关系，每笔输入都要记录这三种关系的数据。

在“sqlite>”提示符号下，我们通过输入以下命令来创建一个数据表。

```
sqlite>CREATE TABLE notes
sqlite >(_id INTEGER PRIMARY KEY,
sqlite >note TEXT NOT NULL,
sqlite >created INTEGER);
sqlite>
```

通过上面的执行，我们创建了一个 notes 表，表里面有_id，note，created 三个字段。

SQLite 命令和 Java 代码一样，大小写代表不同的符号，而且要以分号(;)结尾。在 SQLite 的互动模式中，如果没有输入“;”号就按下“enter”键，在互动模式的下一行就会出

现“...>”提示符号，表示这行语句(命令)还没有结束。

我们用“CREATE TABLE”命令创建数据表，内容以括号包起来，最后要以分号(;)结尾，这样才算完成一个命令语句。

```
_ id INTEGER PRIMARY KEY
```

当属性设为“INTEGER PRIMARY KEY”时，“_id”成为一个自动计数的整数字段。每当新添加一笔数据时，SQLite 会自动以流水编号填写“_id”这个字段。

在 SQLite 上，可以将各字段定义成“TEXT”(文字)，“INTEGER”(整数)等属性。属性后可以加上“NOT NULL ”之类的约束，以表示此字段一定要填入内容。但是在存储时，SQLite 一律以“字符串”(string)类型来存储数据。所以要是不够细心的话，可能会将文字(TEXT)数据存储到标注为整数(INTEGER)的字段内。为了避免这种情况发生，使用 SQLite 时需要注意数据格式的验证。为了验证之前的数据表是否创建成功，可以用“.databases”命令列出目录下的 SQLite 数据库列表。用“.table”命令列出所有数据表。“.schema”命令是显示出数据表的建表语句的内容。如果要退出互动模式，可以使用“.exit”命令。另外，如果想知道 SQLite 的其他命令，可以在互动模式下输入“.help”命令。

重新打开数据库的方法，只需要在“sqlite3”命令后添加数据库名称即可。需要注意的是必须用完整的数据库名，如：“#sqlite3 notes.db”。

5. 数据的增删改查

我们可以用 SELECT 命令来查询数据表中的内容：

```
sqlite>SELECT * FROM system;
```

SQLite 语言会识别大小写，但 SQLite 数据库中，其命令也做了大写，小写两套完全相同的命令。因此 SQL 语句可以接受大写“SELECT”等关键字，也能接受小写“select”等关键字。

SELECT 语句的格式如下：

SELECT 字段 FROM 数据表；

可以通过 INSERT 命令来向数据库添加一组数据。

```
sqlite> INSERT INTO system VALUES(10,'rob','176cm');
```

INNSERT 语句的格式如下：

INSERT INTO 数据表 VALUES (字段内容值)；

可以通过 UPDATE 语句更新数据。

```
sqlite>UPDATE telphone SET value = "010 - 64436873" WHERE_id = 1;
```

UPDATE 语句的格式如下：

UPDATE 数据表 SET 字段＝“内容值” WHERE 条件

可以通过 DELETE 语句删除数据。

sqlite>DELETE FROM telephone WHERE_id＝1；

上面的命令表示：从 telephone 数据表中，删除符合“_id”字段的内容值等于 1 这条件

的项目。

DELETE 语句的格式如下：

DELETE from 数据字段＝内容值

6. 在 Android 中使用 SQL

Context 类的 SQLiteDatabase openOrCeateDatabase(String name, int mode, CursorFactory)方法可以用来建立一个新的数据库或打开一个已有的数据库，并返回一个 SQLiteDatabase 对象。如果不能成功打开已有的数据库，该方法会抛出 SQLiteException 异常。该方法的第一个参数表示数据库的名字，第二个参数表示数据库创建的模式，第三个参数用于查询构造 Cursor 子类的对象，在通常情况下填“nill”。和文件存储类似，在默认情况下，数据库的创建方式也是 MODE_PRIVATE。通常不建议用 MODE_WORLD_WRITABLE 方式创建数据库，对于跨应用的访问数据库需求，应该使用 Content Provider 机制，我们会在后续的章节介绍。

SQLiteDatabase 对象用来操作 SQLite 数据库，可以使用 SQLiteDatabase 对象来执行标准的 SQL 语句，对数据库进行初始化操作。

下面的例子创建了一个名为 SQLiteDemo. db 的数据库，并在其中建立了名为 DataSheet 的数据表，其各个条目的初始值分别是“1”，和“Male”。实现代码如下：

```
import android.app.Activity;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
Import android.os.Bundle;
public class SQLiteActivity extends Activity{
    private static final String DATABASE_NAME = "SQLiteDemo.db";
    //数据库名字
    private static final string TABLE_NAME = "Datasheet";
    //数据表名字
    private static string_id;
    private static string_name;
    private static string_gender;
    private static string_count;
    SQLiteDatabase db = null;
    public void oncreate (Bundle sevedinstancestate){
        super.oncreate (savedinstancestate);
        Setcontentview(R.layout.main);
        createDatabase();
  }
  public void createDatabase(){
    try{
```

```
        Boolean flag = false;
        //判断数据库是否存在
        String[] list = databaselist();
        for (int i = 0;i<list.length - 1;i + +){
            if (DATABASE_NAME.equals (list[1])){
              flag = true;
              Break;
        }
      }
    //创建数据库
    db = openorcreateDatabase(DATABASE_NAME,MODE_PRIVATE,null);
    If (! flag) {//建立数据表
          db.execSQL ("CREATA TABLE " + TABKE_NAME + "(" + _id + " INTERGER
    NOT NULL PRIMARY KEY," + _name + " TEXT," + _GENDER + " TEXT) ;");
    }
    db.execsSQL ("INSERT INTO" + TABLE_NAME + " VALUES('1','Roger','Male')");
    } catch (SQLException e){
        e.printstackTrace();
    }
    }
}
```

7. SQLiteOpenHelper

Android 提供了 SQLiteOpenHelper 抽象类来创建数据库，并通过该类的 getWritableDatabase()方法来获得一个可写的 SQLiteDatabase 对象，这也是创建数据库最常用的方法。SQLiteOpenHelper 类不会重复执行数据库初始化操作，从而避免了前面例子里的查询操作，提高了效率。我们大家都知道 Android 平台提供给我们一个数据库辅助类来创建或打开数据库，这个辅助类继承自 SQLiteOpenHelper 类，在该类的构造器中，调用 Context 中的方法创建并打开一个指定名称的数据库对象。继承和扩展 SQLiteOpenHelper 类主要做的工作就是重写以下两个方法。

(1)onCreate(SQLiteDatabase db)：当数据库被首次创建时执行该方法，一般将创建表等初始化操作在该方法中执行。

(2)onUpgrade(SQLiteDatabse dv, int oldVersion,int new Version)：当打开数据库时传入的版本号与当前的版本号不同时会调用该方法。

除了上述两个必须要实现的方法外，还可以选择性地实现 onOpen 方法，该方法会在每次打开数据库时被调用。

SQLiteOpenHelper 类的基本用法是：当需要创建或打开一个数据库并获得数据库对象时，首先根据指定的文件名创建一个辅助对象，然后调用该对象的

getWritableDatabase 或 getReadableDatabase 方法获得 SQLiteDatabase 对象。调用 getReadableDatabase 方法返回的并不总是只读数据库对象，一般来说该方法和 getWriteableDatabase 方法的返回情况相同，只有在数据库仅开放只读权限或磁盘已满时才会返回一个只读的数据库对象。

下面通过一个简单的小例子说明 SQLiteOpenDatabase 的用法，其中包括创建数据库、插入数据、更新、查询等等，我们将查询后获取到的数据显示到 TextView 上，看一下运行后的效果。

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
public class MySQLiteHelper extends SQLiteOpenHelper{//调用父类构造器
public MySQLiteHelper(Context context, String name, CursorFactory factory,
int version) {super(context, name, factory, version);}
/**
 * 当数据库首次创建时执行该方法，一般将创建表等初始化操作放在该方法中执行.
 * 重写 onCreate 方法，调用 execSQL 方法创建表
 * */
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("create table if not exists hero_info("
+ "id integer primary key,"
+ "name varchar,"
+ "level integer)");
}//当打开数据库时传入的版本号与当前的版本号不同时会调用该方法
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}
}
//不对数据库进行升级，所以 onUpgrade 方法里没有代码
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
public class MySQLiteHelper extends SQLiteOpenHelper{//调用父类构造器
public MySQLiteHelper(Context context, String name, CursorFactory factory,
int version) {
super(context, name, factory, version);
}
```

```
/**
 * 当数据库首次创建时执行该方法,一般将创建表等初始化操作放在该方法中执行.
 * 重写 onCreate 方法,调用 execSQL 方法创建表
 * */
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("create table if not exists hero_info("
 + "id integer primary key,"
 + "name varchar,"
 + "level integer)");}
//当打开数据库时传入的版本号与当前的版本号不同时会调用该方法
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
}
//执行 select 语句
public Cursor query(String sql, String[] args)
{
  SQLiteDatabase db = this.getReadableDatabase();
  Cursor cursor = db.rawQuery(sql, args);
  return cursor;
}
}
```

DBService 类创建了一个 test.db 数据库文件,并在该文件中创建了 t_test 表。在该表中包含了两个字段:_id 和 name。其中_id 是自增字段,并且是索引。

下面是 Main 类。Main 类是 ListActivity 的子类。在该类的 onCreate 方法中创建了 DBService 对象,然后通过 DBService 类的 query 方法查询出 t_test 表中的所有记录,并返回 Cursor 对象。

```
public class Main extends ListActivity
{
  @Override
  public void onCreate(Bundle savedInstnceState)
  { super.onCreate(savedInstandeState);
    DBService dvService = new DBService(this);
    Cursor cursor = dbService =
    dbService.query("select * from t_test",null);
    SimpleCursorAdapter simpleCursorAdapter =
    new SimpleCursorAdapter(this,
      android.R.layout.simple_expandable_list_item_1, cursor,
```

```
            new String[]
            {"name"},new int[]
            {android.R.id.text1});
        setListAdapter(simpleCursorAdapter);
    }
}
```

SimpleCursorAdapter 类构造方法的第 4 个参数表示返回的 Cursor 对象中的字段名，第 5 个参数表示要将该字段的值赋给哪个组件。该组件在第 2 个参数指定的布局文件中定义。

最后需要注意的是，在绑定数据时，Cursor 对象返回的记录集中必须有一个名为“_id”的字段。否则将无法完成数据绑定。

4.2.5 ContentProvider

1. ContentProvider

前面我们介绍的数据存储方法都只针对一个应用本身，如果想实现多个应用之间的数据共享，则应该使用 Android 平台上处理存储数据操作的核心模块 ContentProvider。

ContentProvider 把需要的数据封装起来，并提供了一系列标准的方法接口，应用通过 ContentResolver 来使用这些方法操作 ContentProvider 中的数据。在 Android 平台中，如果是特定数据，如联系人信息会被多个应用使用，则应该使用 ContentProvider 提供的方法来读写，如果数据只被唯一的应用使用则可以用 4.2.4 节介绍的方法来读写。Android 自身提供了几个系统本地实现的 ContentProvider；Contacts，Browser，CallLog，Settings，及 MediaStore。

选择特定的存储媒质来存储数据，如本地普通文件，XML 文件等，通常情况下，选用 Android 自带的 SQLite 数据库作为存储媒质。每个 ContentProvider 都有一个通用资源标识符(URL)作为身份标识，它类似一个指示路径的字符串。如同使用网址去访问一个网站一样，Android 应用程序也使用 URL 去访问一个 ContentProvider。

URL 组成为：

<standard_prefix>://<authority>/<data_path>/<id>，standard_prefix

在 Android 里一般内容都是“content”。表现形式一般分为指定全部数据和指定某一个数据两种，如下：

```
Content://com.ldci.android/member
Content://com.ldci.android/member/2
```

看看下面的代码，注意 Jock 类的一个内部类 Data 实现了 BaseColumns 接口：

```
import android.provider.BaseColuns;
public class Joke{
  //主机名
```

```
public static final string AUTHORITY = "com.ldci.android";
//定义基本字段
public static final class Data implements BaseColumns{
    private Data(){
}
//要共享的数据表 URL
public static final uri CONTENT_URI = uri.parse("content;//"
 + AUTHORITY + "/member");
……}
```

还可以通过 conteturis 类的 conteturis . withappendedid(uri contenturi,long id)方法和 uri 类的 onteturis . withappendedpath(uri baseuri,string pathsegment)方法来创建新的 uri,示例如下:

```
uri new uri = (contenturis.withappendid("com.ldci.android/member",2));
uri new uri = uris.withappendpath("conten;//com.ldci.android/member","2");
```

这两种方法生成的 uri 都是“content;//com. ldci. android/member","2")。需要注意的是,参数 contenturis 和 baseuri 的结尾部分都没有“/”。

为了方便使用,在 android 平台上提供了一些已经定义好的 uri,如 mediastrore. images. media. interinal－content－uri 代表内部储存,contact. people. content－uri 代表全体联系人等。

以 mediaprovider 为例,在定义 mediaprocider 的项目中 androidmanifest. xml 声明文件中包含如下语句:

```
＜procider android:name = "media
procider" android:authoirties = "media" android:multiprocess = "false"＞
```

以上语句中,android:name 代表了 content 类的名字,android:authoritie 代表了 uri 中的＜authority＞字段内容。

2. 创建 ContentProvider

ContentProvider 类是一个抽象的类,其抽象方法如下:

```
// 用于初始化数据,uri 匹配等
abstract uri insert(uri uri,contentvalues valuse);
abstract int delete(uri uri,string selection,string[]selectionargs);
abstract cursor query (uri uri, string[ ] projection, string[ ] selectin args, string
sortorder);
//获取数据类型
abstract string gettype (uri uri);
```

创建一个自定义的 ContentProvider 需要实现这些抽象的方法,以完成数据源的初始配置,提供对数据源操作的接口以及数据类型获取接口。下面我们就从这三个方面介绍这六个抽象方法的实现。

(1)数据和 uri 的初始化。

在 onCreate 方法中实现对底层数据进行打开或初始化操作。同时,需要定义一个 content－uri 变量(public statiic 类型)来记录该 ContentProvider 的 uri。由于 ContentProvider 的 uri 是唯一的,所以 uri 通常会基于包的名称,形式如下:

Content://<authroity>/<data－path>

具体地,例如:Content://com.ldci.android/member

前面章节提到,uri 有两种类型,一种用于返回某一类型的所有值(如上面的文字),另一种用于在上面的路径基础上增加行号的指定,以返回一条指定的记录,例如:Content://com.ldci.android/member/5

所以,支持两种类型的 uri,需要在 ContentProvider 中支持两种访问方式。

配置 urimatcher 的代码如下:

```
private static final int member = 1;
privatestatic final int member－id = 2;
privatestatic final urimatcher urimatcher = new urimactcher();
public class my pro vider extends content provider{
private staticfinal uri matcher urimather;
  static{
Urimatcher = new urimatcher urimatcher(uri matcher.no－match);
Urimatcher.uri("com.ldci.anfdroid".,"nember",allrows);
Urimatcher.adduri ("com.ldci.android",.,"member/＃,single－row");
}
}
```

通过 Urimatcher 这样的机制,可以对 uri 进行有效的区分,将不同的 uri 指向不同的数据集,比如,数据库中不同的表。另外,如果数据来源是数据库,在 ContentProvider 提供列的名称和搜索引擎的访问将简化对数据库返回结果的操作。

(2)数据操作接口实现。

可以通过提供 delete,isert,update 以及 query 方法来实现对 ContentProvider 提供内容的各项操作。这些方法是访问下层数据库的通用接口,各个应用程序可以通过这些接口来共享数据,如数据库,文件及变量等。例如,由于 sqllite 数据库是应用程序私有的,我们可以为某程序 sqllite 数据实现一个 ContentProvide,从而使其他应用程序也可以访问其数据库。

ContentProvide 中对数据库操作的具体实现代码如下:

```
pubilc Cursor query(Uri uri,String[] projection,String selection,String[]
selectionArgs, String sort){
switch(uriMatcher.match(uri)){
case SINGLE_ROW:
```

```
int rowNum = uri.getPathSegments().get(1);
//TODO:更具行号选择要返回的结果
break;
}
return null;
}
pubilc Uir insert (Uir uir,ContentValues initialValues){
long rowID;
//rowID = ...用 initialValues 创建新的一行...
f(rowID>0){
return ContentUris.withAppendedId(CONTENT_URI,rowID);
}
throw new SQLException("Failure in insert a row into " + uri);
}
public int delete(Uri uri,String where,String[] whereArgs){
switch(uriMatcher.match(uri)){
case ALLROWS:
//...
break;
default : throw new IllegalArgumentException("error here");
}
}
public int update(Uri uri, ContentValues values,String where, String where, String[]
whereArgs){
switch(uriMatcher.match(uri)){
case ALLROWS:
//...
break;
default : throw new IllegalArgumentException("error here");
}
}
```

(3)数据类型查询接口实现。

数据类型的查询通过 getType 方法来实现,其返回值描述了目标数据的类型。与 URI 相对应的。返回的数据类型包含两种形式:单条记录和多条记录。实现的方法代码如下:

```
public String getType(Uri uri){
switch(uriMatcher.match(uri)){
case ALLROWS:
```

```
//...
return "com.ldci.android.dir/providercontent";
default: throw new IllegalArgumentException("error here");
}
```

在通过以上 3 步完成对 ContentProvider 的创建后，还需要在应用程序的配置文件中添声明以注册新的 ContentProvider。例如：

```
<Provider
android:name = "member"
android:authorities = "com.ldci.android"/>
```

3. ContentResolver

前面介绍的 ContentProvider 是分享内容的关键，那接下来的 ContentResolver 则充当被分享数据的使用者。通常情况下，ContentProvider 提供了分享的内容以及可以访问这些内容的方式，更多的时候，我们使用 ContentResolver 去修改这些被分享的内容。每个应用程序的 Context 对象均有一个 ContentResolver 对象。并可以通过方法 getContentResolver()得到：

ContentResolver contentResolver=getContentResolver();

getContentResolver 提供了访问 ContentProvider 的方法，可以通过提供一个 URI 来指定要访问的 ContentProvider。

(1)数据读取。

ContentResolver 提供了 query 方法对 ContentProvider 进行读取或查询。如同对数据库的查询，对 ContentProvider 的查询将返回一个 Cursor 对象，应用程序可以通过 Cursor 对象获取读取的内容。

ContentResolver 的 query 方法需要 5 个参数：

①Uri uri，ContentProvied 的 URI，指定了数据来源；

②String[] projection，指定返回的列，传入 null 将返回所有列；

③String selection，指定返回的行，格式与 SQL 语句的 WHERE 字句相同，传入与 null 将返回上次那个 URI 的所有行；

④String selectionArgs，对 selection 语句中出现的 1 个或多个"?"进行替换，将其替换为字符串；

⑤String sortOrder，指定返回结果的排序方式，格式与 SQL 语句的 ORDER BY 从句相同，传入 null 将使用默认的顺序。

对 ContentProvider 提供的数据进行查询的数据代码如下：

```
//返回所有执行
CursoraiiRows = GetcontentResolver().query(MyProvider.CONTENT_URI,null,null,null,null);
//返回第 3 列满足等价条件的所有记录，记录的所有行均返回，
//并且，返回结果按第五列的值进行排序
```

```
String where = KEY_COL3 + " = " + requiredValue;
String order = KEY_COL5;
Cursor someRows = getContentResolvesolver( ). querry(MyProvider. CONTENT _ URI, NULL, WHERE,
NULL,ORDER);
```

(2)数据增加。

数据的增加即插入数据,通过 ContentResolver 的 insert 和 bulkinsert 方法实现。这两种方法均有两个参数,其中,第一个参数均为 ContentProvider 的 URI,第二个略有不同,前者需要的是一个 VontentValuse 的对象,后者需要一个 ContentValues 的数组。

数据插入方法的使用方式代码如下:

```
//创建一行新数据
ContentValuse newValuse = new ContentValues();
//为新的行赋值,这里的 COLUMN_NAME 代表了数据库里表的某个属性
//newValue 表示对这个属性的赋值
newValues.put(COLUMN_NAME,newValue);
//.....为其他每列赋值
Uri myRowUri = getContentResolver(). insert(MyProvider. CONTENT_URI,newValues);
//创建一个 ContentValues Array
ContentValues[] valueArray = new ContentValues[5];
//.....用 bulkinsert 为每行赋值
Int count = getContentResolver(). bulkinsert(MyProvider. CONTENT_URl,valueArray0;
```

(3)数据删除。

数据删除通过 ContentResolver 的 delete 方法实现。方法 delete 有三个数据,参数的作用与 query 中的参数类似。三个参数分别是 ContentProvider 的 URI,WHERE 从句以及用于替换 WHERE 从句中若干“?”的字符串数组。通过对不同的参数进行制定,可以实现删除 1 条记录有条件的删除多条记录。

使用方法如下:

```
//删除一行数据
getContentResolverResolver(). delete(myRowUri,null,null);
//删除前十行数据
String where"_id<10";
getContentResolvesolver(). delete(MyProvider. CONTENT_URI,where,null);
```

(4)数据更新。

数据修改即更新通过 ContentResolvesolver 的 update 方法实现,update 有四个参数:ContentProvider 的 URI,ContentValues 对象,WHERE 从句以及用于替换 WHERE 从句中若干“?”的字符串数组。通过 update 方法,可以有条件的对 ContentProvider 的内容进行更新。

```
//创建新的一行
```

```
ContentValuse newValuse = new newValuseContentValues();
//制定查询条件
String where"_id<10";
getContentResolvesolver().update(MyProvider.CONTENT_URI,where,null,null);
```

5. Andriod 系统 ContentProvider

Andriod 系统本身实现了多个 ContentProvider 供应用程序使用，此类 ContentProvider 常被称为本地 ContentProvider。这些本地 ContentProvider 可以在 Andriod.Provider 包中找到。具体如下：

Browser 可用于读取和修改网页浏览器，浏览历史以及网页搜索等。

CallLog 用于查看和更新呼叫历史，包括呼入列表，呼出列表，未接来电列表和通话细节(如接听时间，呼叫人及持续时间等)。

Contacts 用于读取，修改和保存通讯录的信息。

MediaStore 针对设备上的多媒体文件，MediaStore 提供了集中的托管式访问方式，第三方的应用程序可以通过 MediaStore 来保存音频，视频和图片等文件，并可以设置文件访问权限为全局可见以便共享。

Settings 用于访问和修改设备的设置，如蓝牙设置，手机铃声等。

在设置相关或类似功能时，应用程序应该尽量使用本地 ContentProvider 而避免自己编写，因为这样会带来更好的性能，兼容性以及继承性。

例如，很多应用程序需要获取通讯录的信息，Andriod 会通过本地的 ContentProvider 的形式将通讯录数据库共享给任何有读取权限的应用程序。例如，应用程序可以通过 Contacts.People 类来查询数据库中 People 这张表的信息。People.CONTENT_URl 是该表的 URI 表示，People.NAME 为人名列表的名称，People.NUMBER 是号码列的名称。Contacts 提供的属性和接口大大简化了针对通讯录的复杂度。读取人名和相应的号码信息代码如下：

```
//获取查询结果的 Cursor 对象
Cursor.getContentResolvesolver().
query(people.CONTENT_URI,null,null,null,null);
//让 Activity 管理 Cursor 对象的生命周期
staitManagingCursor(Cursor);
//使用本地 ContentProvider 的属性来获取列的索引值
int nameIdx = Cursor.getColumnIndexOrThrow(people.NAME);
int phoneIdx = Cursor.getColumnIndexOrThrow(people.NUMBER);
String[]result = new String[cursor.getCount()];
if (cursor.moveToFirst())
do{
    //获取人名
```

```
        String name = cursor.getString();
        Result[cursor.getPosition()] = name + "(" + phone + ")";
    }while(cursor.moveToNext());
```

4.2.6　网络存储

网络存储是 Android 中文数据存储的另一种方式，可以将手机上的文件上传到服务器端实现保存。下面的例子实现了一个上传文件的功能：

```
    //192.168.17.156 是 PC 的 IP 地址
      String uploadurl = "http://192.168.17.156:8080/upload/UploadServlet";
      String end = "\r\n";
      String twoHyphens = "--"; //两个字符
      String boundary = "******"; //分界符的字符串
      try
      {
        URL url = new URL(uploadUrl);
        HttpURLConnection httppURLConnection = (HttpURlConnection) url.openConnection();
    //要使用 InputStream 和 OutputStream，必须使用下面两行代码
    httpURLConnection.setDoInput(true);
    httpURLConnection.setDoOutput(true);
    httpURLConnection.setUseCaches(false);
    //设置 HTTP 请求方法，方法名必须大写，如：GET，POST
    httpURLConnection,setRequestMethod("POST");
    httpURLConnection.setRequestProperty("Connection","Keep-Alive");
    httpURLConnection.setRequestProperty("Charset","UTF-8");
    //必须在 Content-Type 请求头中指定分界符中的任意字符串
    httpURLConnection.setRequestProperty("Content-Type","multipart/form-data;boundary="
+ boundary);
    //获得 OutputStream 对象，准备上传文件
    DataOutputStream dos =
                    new DataOutputStream(httpURLConection.getOutputStreaam());
    //设置分界符，加 end 表示为单独一行
    dos.writeBytes(twoHyphens + boundary + end);
    //设置与上传文件相关的信息
    dos.writeBytes("Content-Disposition: form-data; name=\"file\";filename=\"" +
filename.substring(filename.lastIndexOf("/") + 1) + "\"" + end);
    }//在上传文件信息与文件内容之间必须有一个空行
    dos.writeBytes(end);
    //开始上传文件
```

```
FileInputStream fis = new FileInputStream(filename);
Byte[] buffer = new byte[8192];//8k
int count = 0;
//读取文件内容,并写入 OutputStream 对象
while ((count = fis.read(buffer)) ! = -1)
{dos.write(buffer,0,count);
}
Fis.close();//新起一行
dos.writeBytes(end);//设置结束符号(再分界符后面加两个连字符)
dos.writeBytes(twoHyphens + boundary + twoHyphens + end);
Dos.flush();//开始读取从服务器端传过来的信息
InputStream is = httpURLConnection.getInoutStream();
InputStreamReader isr = new BufferedReader(isr);
String result = br.readLine();
Toast.makeText(this,result,Tost.LENGTH_LONG).show();
dos.close();
is.close();
}catch (Exception e){}
```

网络存储方式,需要用到 Android 网络数据包,关于 Android 网络数据包的详细说明,请查阅 Android SDK。

4.3 任务实施

(1)定义类 DistributionDBHHelper 继承 SQLiteOpenHelper,子类构造方法中调用父类的构造方法。

```
public class DistributionDBHHelper extends SQLiteOpenHelper {
    public DistributionDBHHelper(Context context) {
      super(context, DATABASENAME, null, DATABASEVESION);
      // TODO Auto - generated constructor stub
    }
}
```

(2)在 onCreate()方法中开启 SQLiteDatabase 的对象 db 的事务。

```
db.beginTransaction();
```

(3)编写创建表结构 sql 语句,创建数据表结构。

```
String sqlReturngoods = "CREATE TABLE returngoods (id INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL, returngoodsId varchar,state INTEGER,text varchar)";
```

(4)执行创建表结构 sql 语句。

```
db.execSQL(sqlReturngoods);
```

(5)创建类 DistributionService,实现将接收到的货物信息存储到 SQLite 数据库。

```
public class Distribution {};
```

(6)定义 DistributionDBHHelper 类对象 dbhelper,并且构造方法中实例化 dbhelper 对象,dbhelper 调用 getWritableDatabase()方法,并编写初始化运单号等方法。

```
private DistributionDBHHelper dbhelper;
public DistributionService(Context context) {
    this.dbhelper = new DistributionDBHHelper(context);
    SQLiteDatabase db = this.dbhelper.getWritableDatabase();
    db.close();
}
// 初始化运单号
public void initialDistributionWaybill(String waybillId) {
    // 获取一个数据库操作实例
    SQLiteDatabase db = dbhelper.getWritableDatabase();
    db.execSQL("insert into waybill(waybillId,state) values(?,?)",
    new String[] { waybillId, "-1" });
    db.close();
}
```

(7) 在 DistributionService 类中解析从输入流中解析出整个运单的所有信息:运单信息,商品种类信息,商品信息。

```
public Waybill getDistributionWaybillFromInputStream(InputStream inStream)
  throws Throwable {
  XmlPullParser parser = Xml.newPullParser();
  parser.setInput(inStream, "UTF-8");
  int eventType = parser.getEventType();// 产生第一个事件
  Waybill waybill = new Waybill(true);
  ProductVariety productVariety = null;
while (eventType != XmlPullParser.END_DOCUMENT) {// 只要不是文档结束事件
  switch (eventType) {
    case XmlPullParser.START_DOCUMENT:
      break;
    case XmlPullParser.START_TAG:
  String name = parser.getName();// 获取解析器当前指向的元素的名称
    if ("Waybill".equals(name)) {
        waybill.setWaybillInfo(
          new Integer(parser.getAttributeValue(0)),
          parser.getAttributeValue(1),
          parser.getAttributeValue(2),
```

```
                parser.getAttributeValue(3),
                parser.getAttributeValue(4),
                parser.getAttributeValue(5),
            Integer.parseInt(parser.getAttributeValue(6)),
                null, parser.getAttributeValue(7));
              }
              break;
            case XmlPullParser.END_TAG: // 结束元素标签
              if ("ProductVariety".equals(parser.getName())) {
                waybill.getProductVariety().add(productVariety);
                productVariety = null;
              }
              break;
            }
            eventType = parser.next();
          }
          return waybill;
        }
```

(8)将货物信息插入到数据库中。

```
        SQLiteDatabase db = dbhelper.getWritableDatabase();
            db.beginTransaction();// 开始事务
            try {
            db.execSQL("update waybill set name = '" + waybill.getName() + "',companyName = '" +
waybill.getCompanyName() + "', telephone = '" + waybill.getTelephone() + "',adress = '" +
waybill.getAddress() + "', texte = '" + waybill.getText() + "' ,payType = '" + waybill.
getPayType() + "', state = '" + waybill.getState() + "' where waybillId = " + waybill.
getWaybillID());
            List<ProductVariety> proVarietys = waybill.getProductVariety();
            for (ProductVariety proVariety : proVarietys) {
              db.execSQL(
                  "insert into productvariety (waybillIdref,number,productvarietyId,name,isbn,
color,size,weight,state)values('"
                    + waybill.getWaybillID()
                    + "','"
                    + proVariety.getNumber()
                    + "','"
                    + proVariety.getId()
                    + "',?,?,?,?,?,'"
```

```
            + proVariety.getWeight()
            + "','"
            + proVariety.getState() + "')", new String[] {
    proVariety.getName(), proVariety.getIsbn(),
    proVariety.getColor(), proVariety.getSize() });
    }
    db.setTransactionSuccessful();
  } finally {
}
```

(9)数据存储之后关闭 db 事务,关闭 db 对象。

```
db.endTransaction();
db.close();
```

(10)货物信息实体类的定义。

```
public class Product {
    private Long id;
    private String batchNumber;
    private String text;
    private int state;
    public int getState() {
      return state;
    }
    public void setProductInfo(Long id, String batchNumber, String text,
      int state) {
      this.id = id;
      this.batchNumber = batchNumber;
      this.text = text;
      this.state = state;
    }
  public void setState(int state) {
      this.state = state;
  }
  public Product() {
  }
  public Product(Long id, String batchNumber, String text) {
      super();
      this.id = id;
      this.batchNumber = batchNumber;
      this.text = text;
```

```
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getBatchNumber() {
        return batchNumber;
    }
    public void setBatchNumber(String batchNumber) {
        this.batchNumber = batchNumber;
    }
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
    @Override
      public String toString() {
        return "Product [id = " + id + ", batchNumber = " + batchNumber + ", text = " + text
+ "]";
    }
}
```

(11)创建存储货物信息 SQLite 数据库的类。

```
/**
 * 该类主要作用:为送货创建服务数据库
 */
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;
public class DMSDBOpenHelper extends SQLiteOpenHelper {
  private static final String DATABASENAME = "DMS";// 数据库名称
  private static final int DATABASEVESION = 1;// 数据库版本
  public DMSDBOpenHelper(Context context) {
    super(context, DATABASENAME, null, DATABASEVESION);
```

```
        // TODO Auto-generated constructor stub
    }
    @Override
    public void onCreate(SQLiteDatabase db) {// 该方法仅仅执行一次,在数据库第一次被创建时执行
        // TODO Auto-generated method stub
        // execSQL()方法,执行有更新数据操作时用
        // db.openDatabase(, factory, flags)
        db.beginTransaction();
        try {
          //创建表
          String sqlgprs = "CREATE TABLE gps (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
GPSx varchar,GPSy varchar,time varchar,user varchar)";
    String sqlwaybill = "CREATE TABLE waybill (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
waybillId varchar,name varchar,companyName varchar,telephone varchar,adress varchar,payType
INTEGER,payPrice REAL,payNumber TEXT,state INTEGER,text varchar,datetime varchar)";
    String sqlProductVariety = " CREATE TABLE productvariety (id NTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL,waybillIdref archar,productvarietyId INTEGER,name varchar,isbn varchar,
color varchar,size varchar,weight REAL,state INTEGER)";
    String sqlproduct = "CREATE TABLE product (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
waybillIdref varchar,productvarietyIdref INTEGER ,productId INTEGER,batchNumber varchar,state
INTEGER,text varchar)";
    //在数据库中添加触发器
    String triggerdeletewaybill = "CREATE TRIGGER deletewaybill after DELETE ON [waybill]BEGIN
DELETE FROM [productvariety] WHERE waybillIdref = old.waybillId;END";
    String triggerdeleteproductvariety = "CREATE TRIGGER deleteproductvariety after DELETE ON
[productvariety] BEGIN DELETE FROM [product] WHERE waybillIdref = old.waybillIdref and
productvarietyIdref = old.productvarietyId;END";
    String triggerdeletewaybill_gprs = "CREATE TRIGGER deletewaybill_gprs after DELETE ON
[waybill]BEGIN DELETE FROM [gprs] WHERE waybillIdref = old.waybillId;END";
    String triggerdeletewaybill_payment = "CREATE TRIGGER deletewaybill_payment after DELETE ON
[waybill]BEGIN DELETE FROM [payment] WHERE waybillIdref = old.waybillId;END";
    //执行 sql 语句
            db.execSQL(sqlwaybill);
            db.execSQL(sqlProductVariety);
            db.execSQL(sqlproduct);
            db.execSQL(sqlgprs);
    // 增添级联删除触发器
```

```
            db.execSQL(triggerdeletewaybill);
            db.execSQL(triggerdeleteproductvariety);
            db.execSQL(triggerdeletewaybill_gprs);
            db.execSQL(triggerdeletewaybill_payment);
    // 设置事务处理成功标志
            db.setTransactionSuccessful();
        } finally {
        }
        db.endTransaction();
        // db.close();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // 该方法在数据库结构改变时执行,可通过更改版本号,来调用;
    }
}
```

4.4 小 结

本部分主要通过任务“智能物流管理系统”项目中用户个人信息等数据存储功能的实现,讲解了 Android 中的常用数据存储方式,主要讲解了 File 存储方式、SharePreferences 存储方式、SQLite 存储方式、ContentProvider 存储等内容。