第 8 章 Android 的存储方式

学习目标：

- Android 在存储方面的系统知识；
- 各类存储的使用及步骤；
- SQLite 方式的存储实现 ；
- contentProvider 方式的存储实现。

8.1 存储概述

典型的桌面操作系统提供一种公共文件系统——任何应用软件可以使用它来存储和读取文件，该文件也可以被其他的应用软件所读取（会有一些权限控制设定）。Android 采用了一种不同的系统，在 Android 中，所有的应用软件数据（包括文件）为该应用软件所私有。然而，Android 同样也提供了一种标准方式供应用软件将私有数据开放给其他应用软件。这一章节描述一个应用软件存储和获取数据、开放数据给其他应用软件、从其他应用软件请求数据并且开放它们的多种方式。

在 Android 中，可供选择的存储方式有 SharedPreferences、文件存储、SQLite 数据库方式、内容提供器（content provider）和网络，我们将在本章详细介绍。

8.2 SharedPreferences 存储

首先介绍的是 SharedPreferences，其是 Android 提供用来存储一些简单的配置信息的一种机制，例如，一些默认欢迎语、登录的用户名和密码等。其以键值对的方式存储，使得我们可以很方便的读取和存入，下面看一个演示的例子。

1.第一步

在 Eclipse 中打开 ex_SharedPreferences 项目，其步骤如下所示。

（1）新建一个项目。依次单击 File→ New >→Android Project。
（2）在新建项目的对话框中，选择 Create project from existing source。
（3）单击浏览，找到 ex_SharedPreferences 项目，然后单击确定。

其程序的目录结构如图 8-2-1 所示：

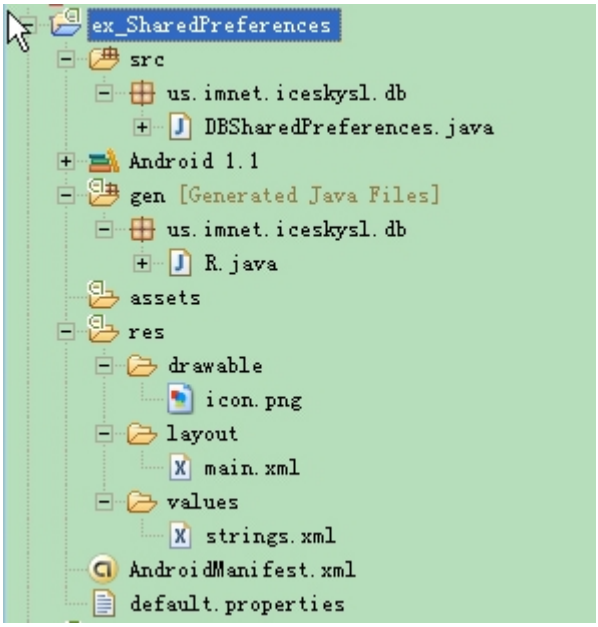


图 8-2-1 程序目录结构

2.第二步

单击运行项目，可以看到主界面如图 8-2-2 所示，这个界面的布局信息都在 main.xml 文件当中，在一个 LinearLayout 当中放了 3 个 TextView 和两个 EditView，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
   Android:orientation="vertical"
   Android:layout_width="fill_parent"
   Android:layout_height="fill_parent"
   >
<TextView
   Android:layout_width="fill_parent"
   Android:layout_height="wrap_content"
   Android:text="SharedPreferences demo"
   />
<TextView Android:layout_width="fill_parent"
Android:layout_height="wrap_content"
Android:text="Name:" />
<EditText Android:id="@+id/name"
Android:layout_width="fill_parent"
Android:layout_height="wrap_content"
Android:text="" />
<TextView Android:layout_width="fill_parent"
Android:layout_height="wrap_content"
Android:text="Password:" />
<EditText Android:id="@+id/password"
Android:layout_width="fill_parent"
Android:layout_height="wrap_content"
Android:password="true"
Android:text="" />
</LinearLayout>
```

如上代码表示：使用 LinearLayout 布局，其中放置三个用来做界面提示的文本框(TextView)组件和两个用于输入 Name 和 Password（注意这里使用了 Android:password="true"）的编辑框(EditText)组件，运行这个应用，可以看到其界面如图 8-1 所示。

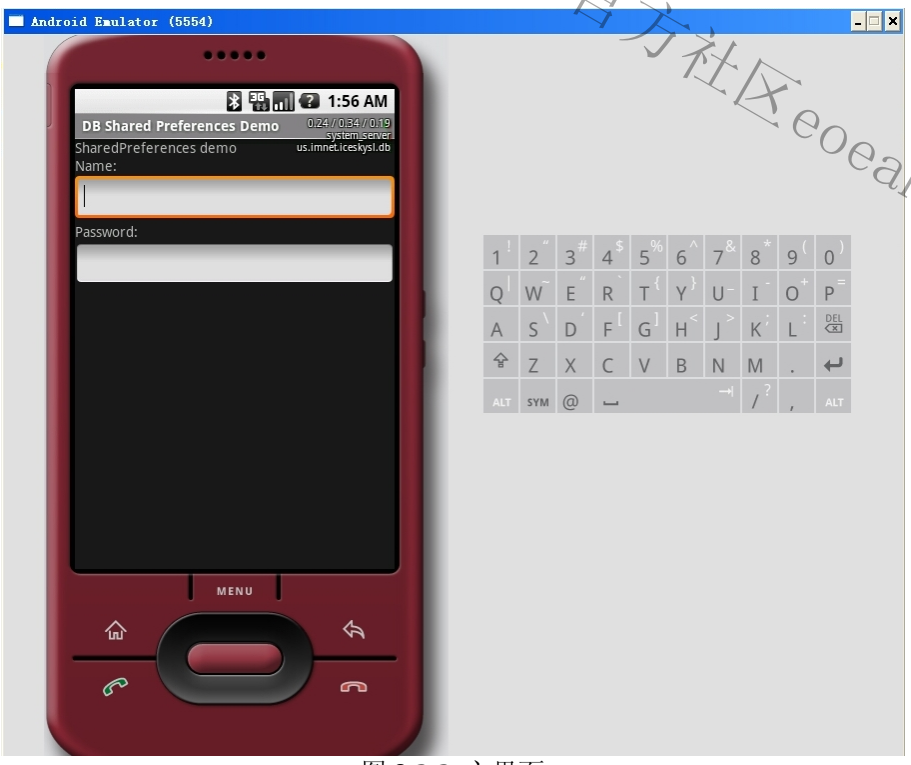


图 8-2-2 主界面

在图 8-1 中，我们可以看到，初始状态下两个 EditView 都是空的，现在输入一些字符，如图 8-2-3 所示。

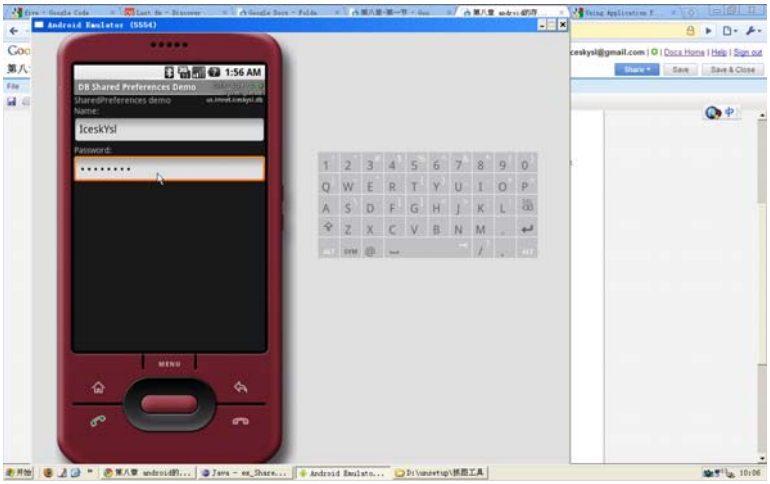


图 8-2-3 输入 Name 和 Password

如图 8-2 所示，我们在 Name 文本框中中输入 IceskYsl，在 Password 文本框中输入 Password，然后退出这个应用。我们在应用程序列表中找到这个应用，重新启动，可以看到其使用了前面输入的 Name 和 Password，如图 8-2-4 所示。

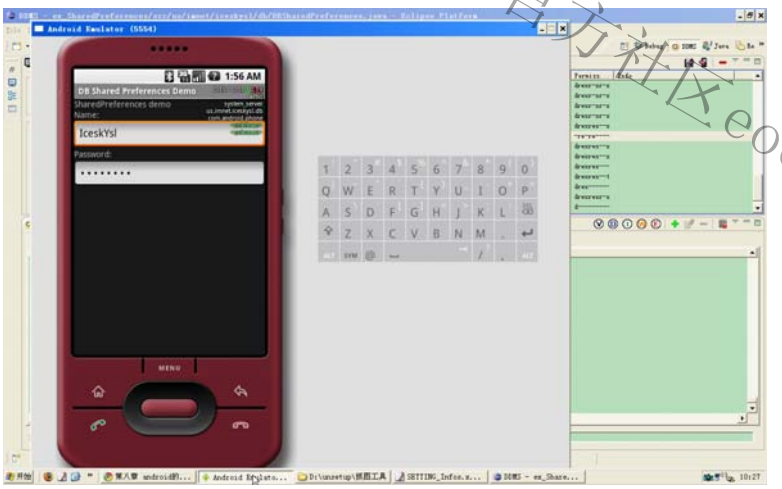


图 8-2-4 重新启动应用

由此可见，应用保存了我们输入的 Name 和 Password，现在来看看其实现的代码，在 DBSharedPreferences.java 文件中，此文件的代码如下所示：

```
package us.imnet.iceskysl.db;

import Android.app.Activity;
import Android.content.SharedPreferences;
import Android.os.Bundle;
import Android.widget.EditText;

public class DBSharedPreferences extends Activity {
public static final String SETTING_INFOS = "SETTING_Infos";
public static final String NAME = "NAME";
public static final String PASSWORD = "PASSWORD";
private EditText field_name;   //接收用户名的组件
private EditText filed_pass;   //接收密码的组件


/** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //Find VIew
        field_name = (EditText) findViewById(R.id.name);   //首先获取用来输入用户名的组件
        filed_pass = (EditText) findViewById(R.id.password); //同时也需要获取输入密码的组件
        // Restore preferences
        SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0); //获取一个 SharedPreferences 对象
String name = settings.getString(NAME, "");   //取出保存的 NAME
String password = settings.getString(PASSWORD, ""); //取出保存的 PASSWORD
//Set value
field_name.setText(name);   //将取出来的用户名赋给 field_name
filed_pass.setText(password);   //将取出来的密码赋给 filed_pass

    }

    @Override
    protected void onStop(){
        super.onStop();
SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0); //首先获取一个 SharedPreferences 对象
settings.edit()
.putString(NAME, field_name.getText().toString())
.putString(PASSWORD, filed_pass.getText().toString())
.commit();
    } //将用户名和密码保存进去
```

}

通过上述代码可以看到，在 onCreate 中使用 findViewById 得到两个 EditView 后，使用 getSharedPreferences 取得 SharedPreferences 对象 settings，然后使用 getString 取得其中保存的值，最后使用 setText 将其值设置为两个 EditText 的值。

而在程序运行onStop过程，也就是在程序退出时，首先使用getSharedPreferences得到settings，然后调用edit()方法使其处于可以编辑状态，并使用putString将两个EditText中的值保存起来，最后使用commit()方法提交即可保存。

小知识：

SharedPreferences 保存到哪里去了？

SharedPreferences 是以 XML 的格式以文件的方式自动保存的，在 DDMS 中的 File Explorer 中展开到 /data/data/<package name>/shared_prefs 下，以上面这个为例，可以看到一个叫做 SETTING_Infos.xml 的文件，如图 8-2-5 所示。

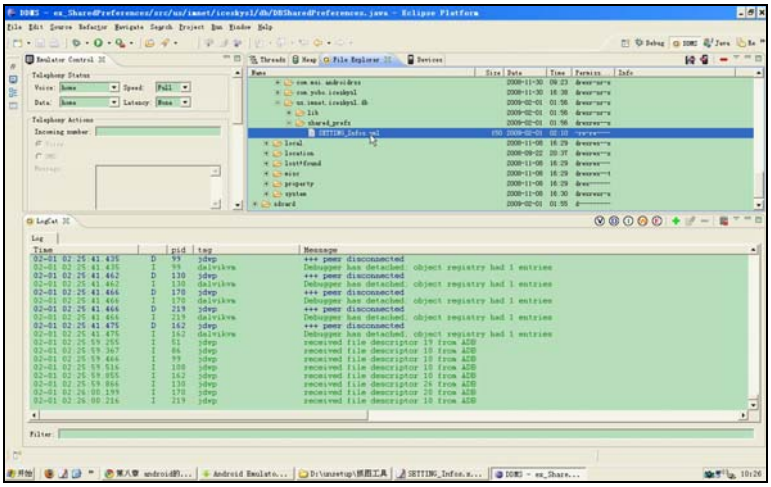


图 8-2-5 SharedPreferences 文件

将其导出到设备中，可以打开这个文件，看到其代码内容为：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="PASSWORD">Password</string>
<string name="NAME">IceskYsl</string>
</map>
```

小知识：

我们可以通过「getXXX」函数，从 SharedPreferences 中读取不同类型的内容，例如，上面我们使用[

getString]读取 String 类型的内容。

注意：

Preferences 只能在同一个包内使用，不能在不同的包之间使用。

8.3 文件存储

前面介绍的 Shared Preferences 存储方式非常方便，但是其只适合存储比较简单的数据，如果需要存储更多的数据，可行选择的方式有好几种，这里先给读者介绍文件存储的方法。

和传统的 Java 中实现 I/O 的程序类似，在 Android 中，其提供了 openFileInput 和 openFileOuput 方法读取设备上的文件，下面看个例子代码，具体如下所示：

```
String FILE_NAME = “tempfile.tmp”;   //确定要操作文件的文件名
// Create a new output file stream that’s private to this application.
```

```
FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE); //初始化
// Create a new file input stream.
FileInputStream fis = openFileInput(FILE_NAME); //创建写入流
```

上述代码中两个方法只支持读取该应用目录下的文件，读取非其自身目录下的文件将会抛出异常。需要提醒的是，如果调用FileOutputStream 时指定的文件不存在，Android 会自动创建它。另外，在默认情况下，写入的时候会覆盖原文件内容，如果想把新写入的内容附加到原文件内容后，则可以指定其mode为Context.MODE_APPEND。

注意：

默认情况下，使用 openFileOutput 方法创建的文件只能被其调用的应用使用，其他应用无法读取这个文件，如果需要在不同的应用中共享数据，可以使用 Content Provider 实现，关于 Content Provider 我们将在稍后的内容中介绍。

小知识：

资源文件放在哪里？

如果你的应用需要一些额外的资源文件，例如，一些用来测试你写的音乐播放器是否可以正常工作的 MP3 文件，可以将这些文件放在应用程序的/res/raw/下，如 *mydatafile.mp3*。那么就可以在你的应用中使用 getResources 获取资源后，以 openRawResource 方法（不带后缀的资源文件名）打开这个文件，实现代码如下所示：

```
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

除了前面介绍的读写文件外，Android 还提供了诸如 deleteFile、fileList 等方法来操作文件，不再赘述。

8.4 SQLite 存储方式

8.4.1 Android 中对数据库进行操作

在前边的章节当中主要学习关于 Android 在布局和显示方面的知识，在这一节中将开始学习 Android 应用的另外一个方面：数据存储。用户可以将自己的数据存储到文件系统或者数据库当中，当然最经常的是，用户将自己的数据存储到 SQLite 数据库当中。SQLite 是 Android 所带的一个标准的数据库，它支持 SQL 语句，它是一个轻量级的嵌入式数据库。

在这个例子里边，我们在程序的主界面有一些按钮，通过这些按钮可以对数据库进行标准的增、删、改、查。

通过这个例子我们可以学到。

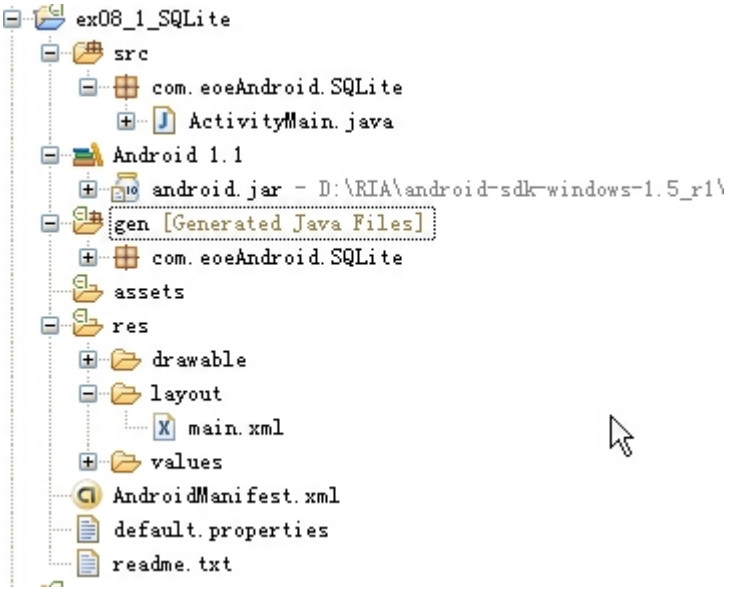
- 如何新建一个数据库。
- 如何新建数据库里边的数据表。
- 如何删除数据库里边的数据表。
- 如何在数据表中添加新数据。
- 如何删除数据库表中的数据。
- 如何使用 Android 提供的工具 File explore，来查看和删除模拟器当中的数据库表。
- 如何使用 LogCat 来看程序当中打印的日志。

1.第一步

在 Eclipse 中，打开 ex08_1_SQLite 项目，具体步骤如下。

- 新建一个项目。单击 File > New > Android Project。
- 在新建项目的对话框中，选择 Create project from existing source.。
  单击浏览，找到 ex08_1_SQLite 项目，然后单击确定。

程序的目录结构如图 8-4-1 所示：



2.第二步

单击运行项目，我们可以看到主界面如图 8-4-2 所示，这个界面的布局信息都在 main.xml 文件中，在一个 LinearLayout 当中数值排列了 5 个 button。

图 8-4-2 主界面

3.第三步

小知识：

什么是 SQLiteDatabase？

一个 SQLiteDatabase 的实例代表了一个 SQLite 的数据库，通过 SQLiteDatabase 实例的一些方法，我们可以执行 SQL 语句，对数据库进行增、删、查、改的操作。需要注意的是，数据库对于一个应用来说是私有的，并且在一个应用当中，数据库的名字也是惟一的。

小知识：

什么是 SQLiteOpenHelper ？

根据这名字，我们可以看出这个类是一个辅助类。这个类主要生成一个数据库，并对数据库的版本进行管理。当在程序当中调用这个类的方法 getWritableDatabase()，或者 getReadableDatabase()方法的时候，如果当时没有数据，那么 Android 系统就会自动生成一个数据库。SQLiteOpenHelper 是一个抽象类，我们通常需要继承它，并且实现里边的 3 个函数，具体函数如下所示。

- onCreate(SQLiteDatabase)：在数据库第一次生成的时候会调用这个方法，一般我们在这个方法里边生成数据库表。
- onUpgrade(SQLiteDatabase, int, int) ：当数据库需要升级的时候，Android 系统会主动的调用这个方法。一般我们在这个方法里边删除数据表，并建立新的数据表，当然是否还需要做其他的操作，完全取决于应用的需求。
- onOpen(SQLiteDatabase)：这是当打开数据库时的回调函数，一般也不会用到。

我们在 ActivityMain 当中看下边这个内部类。DatabaseHelper 类继承 SQLiteOpenHelper，具体代码如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {
DatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
// sql 语句
String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
+ " text not null, " + BODY + " text not null " + ");";
Log.i("haiyang:createDB=", sql);
//执行这条 sql 语句
db.execSQL(sql);

}

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

- DatabaseHelper 类继承了SQLiteOpenHelper 类，并且重写了onCreate和onUpgrade方法。
- 在 onCreate()方法里边首先我们构造和 sql 语句，然后调用 db.execSQL(sql)执行 sql 语句。这条 sql 语句为我们生成了一张数据库表。
  目前我们还不需要升级数据库，所以我们在 onUpgrade()函数里边没有执行任何操作。

4.第四步

我们单击插入两条记录的按钮，如果数据成功插入到数据库当中的 diary 表中，那么在界面的 title 区域就会有成功的提示，如图 8-4-3 所示。

图 8-4-3 成功插入两条数据。

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 insertItem 方法，其具体代码如下所示：

```
private void insertItem() {
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
//首先生成 sql 语句
String sql1 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
+ ") values('haiyang', 'Android 的发展真是迅速啊');";
String sql2 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
+ ") values('icesky', 'Android 的发展真是迅速啊');";
try {
Log.i("haiyang:sql1=", sql1);
Log.i("haiyang:sql2=", sql2);
db.execSQL(sql1);
db.execSQL(sql2);
setTitle("插入两条数据成功");
} catch (SQLException e) {
setTitle("插入两条数据失败");
}
}
```

代码解释

SQLiteDatabase db = mOpenHelper.getWritableDatabase()这条语句负责得到一个可写的 SQLite 数据库，如果这个数据库还没有建立，那么 mOpenHelper 辅助类负责建立这个数据库。如果数据库已经建立，那么直接返回一个可写的数据库。

sql1 和 sql2 是我们构造的标准的插入 SQL 语句，如果对 SQL 语句不是很熟悉，可以参考相关的书记。鉴于本书的重点是在 Android 方面，所以对 SQL 语句的构建不进行详细的介绍。

Log.i（）会将参数内容打印到日志当中，并且打印级别是Info级别，在使用LogCat工具的时候我们会进行详细的介绍。

db.execSQL(sql1);对 sql 语句进行执行。

小知识：对 Android 的打印级别介绍

Android 支持五种打印级别，分别是 Verbose，Debug、Info、Warning、Error，当然我们在程序当中一般用到的是 Info 级别，即将一些自己需要知道的信息打印出来。

如图 8-4-4 所示：

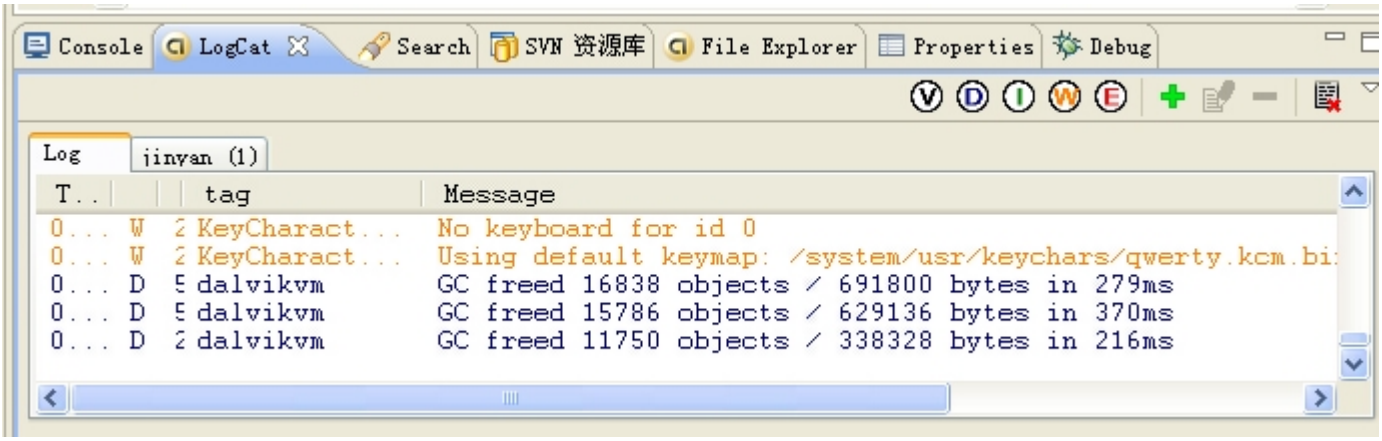


图 8-4-3 Android 中的五种打印级别

注意：

虽然执行 SQL 语句在 Android 当中并没有强制放在 try catch 语句当中，但是我们最好这么做。并在 catch 模块中将错误信息打印在日志当中。这样一方面方便调试，一方面可以加强程序的健壮性。

5.第五步

单击查询数据库的按钮，会在界面的 title 区域显示当前数据表当中数据的条数，刚才我们插入了两条，那么现在单击后应该显示为 2 条，如图 8-4-4 所示。

图 8-4-4 显示两条数据

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 showItems 方法，具体代码如下所示：

```
private void showItems() {

SQLiteDatabase db = mOpenHelper.getReadableDatabase();
String col[] = { TITLE, BODY };
//进行数据库查询
```

```
Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null);
Integer num = cur.getCount();
setTitle(Integer.toString(num) + "  条记录");
}
```

代码解释：

- SQLiteDatabase db = mOpenHelper.getReadableDatabase();首先得到一个可写的数据库。
- Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null);将查询到的数据放到一个 Cursor 当中。这个 Cursor 里边封装了这个数据表 TABLE_NAME 当中的所有条列。 query()方法相当的有用，在这里我们简单的讲一下。
- 第一个参数是数据库里边表的名字，比如在我们这个例子，表的名字就是 TABLE_NAME,也就是"diary"。
- 第二个字段是我们想要返回数据包含的列的信息。在这个例子当中我们想要得到的列有 title、body。我们把这两个列的名字放到字符串数组里边 来。
- 第三个参数为 selection,相当于 sql 语句的 where 部分，如果想返回所有的数据，那么就直接置为 null。
- 第四个参数为 selectionArgs。在 selection 部分，你有可能用到?,那么在 selectionArgs 定义的字符串会代替 selection 中的?。
- 第五个参数为 groupBy。定义查询出来的数据是否分组，如果为 null 则说明不用分组。
- 第六个参数为 having ，相当于 sql 语句当中的 having 部分。
- 第七个参数为 orderBy，来描述我们期望的返回值是否需要排序，如果设置为 null 则说明不需要排序。
- Integer num = cur.getCount();通过 getCount()方法，可以得到 cursor 当中数据的个数。

小知识：

什么是 Cursor ？

Cursor 在 Android 当中是一个非常有用的接口，通过 Cursor 我们可以对从数据库查询出来的结果集进行随机的读写访问。

6.第六步

单击删除一条数据库的按钮后，如果成功删除，我们可以看到在屏幕的标题（title）区域有文字提示，如图 8-4-5 所示。

图 8-4-5 成功删除一条记录

现在我们再单击查询数据库按钮，看数据库里边的记录是不是少了一条。单击查询数据库按钮后，出现如图 8-4-6 所示的界面。

图 8-4-6 还剩一条记录

下面我们来看一下如何删除数据。

单击删除一条记录的按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 deleteItem 方法，其代码如下所示：

```
private void deleteItem() {
    try {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    //进行删除操作
```

```
db.delete(TABLE_NAME, " title = 'haiyang'", null);
setTitle("删除 title 为 haiyang 的一条记录");
} catch (SQLException e) {

}

}
```

代码解释：

- db.delete(TABLE_NAME, " title = 'haiyang'", null); 删除了一条 title='haiyang'的数据。当然如果有很多条数据 title 都为'haiyang',那么一并删除。我们对 delete 方法的参数进行以下介绍：
- 第一个参数是数据库表名，在这里是 TABLE_NAME，也就是 diary。
- 第二个参数，相当于 sql 语句当中的 where 部分，也就是描述了删除的条件。
- 如果在第二个参数当中有？符号，那么第三个参数中的字符串会依次替换在第二个参数当中出现的？符号。

7.第七步

单击删除数据表，我们可以删除 diary 这张数据表，如图 8-4-7 所示。

图 8-4-7 删除数据库表

下边我们看在代码部分，是怎么实现删除的，具体代码如下所示：

```
private void dropTable() {
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
String sql = "drop table " + TABLE_NAME;
try {
//执行 sql 语句
db.execSQL(sql);
setTitle("数据表成功删除：" + sql);
} catch (SQLException e) {
setTitle("数据表删除错误");
}
}
```

代码解释：

首先我们构造了一个标准的删除数据表的 sql 语句，然后执行这条语句 db.execSQL(sql);

8.第八步

现在单击其他的按钮，程序运行时有可能会出现异常，我们单击重新建立数据表按钮，如图 8-4-8 所示。

图 8-4-8 重新建立数据库表

现在我们单击查询数据库，看里边是否有数据，如图 8-4-9 所示。

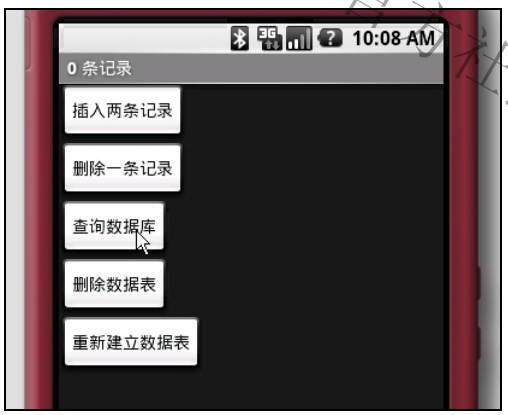


图 8-4-9 新建的表里边没有数据

下边我们看一下程序是怎么建立一张新表的，具体实现代码如下所示：

```
private void CreateTable() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
    + " text not null, " + BODY + " text not null " + ");";
    Log.i("haiyang:createDB=", sql);

    try {
    db.execSQL("DROP TABLE IF EXISTS diary");
    db.execSQL(sql);
    setTitle("数据表成功重建");
    } catch (SQLException e) {
    setTitle("数据表重建错误");
    }
    }
```

代码解释：

- sql 语句为标准的 sql 语句，负责按要求建立一张新表。
- db.execSQL("DROP TABLE IF EXISTS diary");如果存在 diary 这张表，我们需要先删除，因为在同一个数据库当中不能出现两张同样名字的表。
- db.execSQL(sql);执行 sql 语句，新表建立。

8.4.2 安全和方便地访问数据库

在上一个例子中，我们对 Android 系统自带的 SQLite 数据库进行了初步的学习，了解了一些增、删、改、查的基本工作。在这一节的例子当中，我们做了一个非常简便的日记本程序，虽然没有完善，但是已经是基本可以使用了。在例子当中，我们不但要对数据库进行增、删、改、查的操作，而且还要把数据库当中的数据显示在一个 ListView 当中，通过对 ListView 的操作，实现对数据的增、删、改、查操作。

通过这个例子我们可以学到。

- 如何对 DatabaseHelper 和 SQLiteDatabase 封装，以便让我们访问数据库更加方便和安全。
- 如何利用 ContentValues 类来代替原始的 sql 语句进行数据库的操作
  如何使用 SimpleCursorAdapter 类和 ListView 配合进行 ListView 的显示。

日记本具体实现步骤如下所示。

1.第一步

在 Eclipse 中打开 ex08_2_SQLite 项目，具体操作步骤如下。

- 新建一个项目。但击 File > New > Android Project。
- 在新建项目的对话框中，选择 Create project from existing source.
  单击浏览，找到 ex08_2_SQLite 项目，然后单击确定。

程序的目录结构如图 8-4-10 所示：

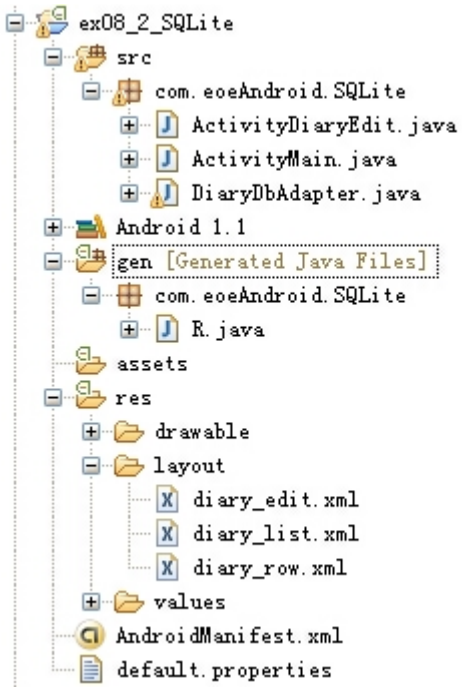


图 8-4-10 程序的目录结构

2.第二步

我们首先运行一下建立的程序，将会出现如图 8-4-11 所示。

图 8-4-11 没有任何数据的程序主界面

程序的主Activity是ActivityMain，它是一个ListActivity，和它关联的布局文件是diary_list.xml。关于ListActivity的介绍。请参阅第 7 章关于ListView的介绍。

3.第三步：

在继续操作前，让我们重点关注一下 DiaryDbAdapter 类，这个类封装了 DatabaseHelper 和 SQLiteDatabase 类，使得我们对数据库的操作更加安全和方便。

在 DiaryDbAdapter 的类变量里，主要定义了一下几个变量。

- 数据库、数据表、数据表中列的名字。
- DatabaseHelper 和 SQLiteDatabase 的实例。
- Context 实例。

DatabaseHelper 类的定义和上一个例子一样，只不过这个例子里边，我们在 onUpgrade 增加了升级的代码，具体如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
//生成数据库
        db.execSQL(DATABASE_CREATE);

    @Override
```

```
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS diary");
            onCreate(db);
        }
    }
```

代码解释：

在 DiaryDbAdapter 类里，我们向外界提供了这么一些方法：

- open()，调用这个方法后，如果数据库还没有建立，那么会建立数据库，如果数据库已经建立了，那么会返回可写的数据库实例。
- close(),调用此方法，DatabaseHelper 会关闭对数据库的访问。
- createDiary(String title, String body)通过一个 title 和 body 字段在数据库当中创建一条新的纪录。
- deleteDiary(long rowId) 通过记录的 id，删除数据库中的那条记录。
- getAllNotes()得到 diary 表中所有的记录，并且以一个 Cursor 的形式进行返回。
- getDiary(long rowId)通过记录的主键 id，得到特定的一条记录。
- updateDiary(long rowId, String title, String body)更新主键 id 为 rowId 那条记录中的两个字段 title 和 body 字段的内容。

小知识：

什么是 ContentValues 类？

ContentValues 类和 Hashtable 比较类似，它也是负责存储一些名值对，但是它存储的名值对当中的名是一个 String 类型，而值都是基本类型。

我们回顾一下，在上一个例子当中，我们是通过SQL语句进行插入操作，SQL语句的好处是比较直观，但是容易出错。但是在这个例子当中我们有更好的办法，在这里我们将要插入的值都放到一个ContentValues的实例当中，然后执行插入操作，具体代码如下所示：

```
public long createDiary(String title, String body) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_TITLE, title);
        initialValues.put(KEY_BODY, body);
        Calendar calendar =  Calendar.getInstance();
// 生成年月日字符串
        String created = calendar.get(Calendar.YEAR)+"年"+calendar.get(Calendar.MONTH)+"月
"+calendar.get(Calendar.DAY_OF_MONTH)+"日"+calendar.get(Calendar.HOUR_OF_DAY)+"时
"+calendar.get(Calendar.MINUTE)+"分";
        initialValues.put(KEY_CREATED, created);
        return mDb.insert(DATABASE_TABLE, null, initialValues);
    }
```

- ContentValues initialValues = new ContentValues();实例化一个 contentValues 类。
- initialValues.put(KEY_TITLE, title);将列名和对应的列值放置到 initialValues 里边。
- mDb.insert(DATABASE_TABLE, null, initialValues);负责插入一条新的纪录，如果插入成功则会返回这条记录的 id，如果插入失败会返回-1.

在更新一条记录的时候，我们也是采用 ContentValues 的这套机制，具体代码如下所示：

```
public boolean updateDiary(long rowId, String title, String body) {
ContentValues args = new ContentValues();
args.put(KEY_TITLE, title);
args.put(KEY_BODY, body);
Calendar calendar = Calendar.getInstance();
String created = calendar.get(Calendar.YEAR) + "年"
+ calendar.get(Calendar.MONTH) + "月"
+ calendar.get(Calendar.DAY_OF_MONTH) + "日"
+ calendar.get(Calendar.HOUR_OF_DAY) + "时"
+ calendar.get(Calendar.MINUTE) + "分";
args.put(KEY_CREATED, created);

return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

4.第四步

现在返回到程序的主界面，对应的Activity是ActivityMain.
当我们单击 menu 按钮后会出现如图 8-4-12 所示界面。

图 8-4-12 点击 menu

根据我们第 7 章对 menu 的学习，对单击 menu 里边按钮的处理逻辑全部放在 onMenuItemSelected 函数里，具体代码如下所示：

```
public boolean onMenuItemSelected(int featureId, MenuItem item) {
switch (item.getItemId()) {
case INSERT_ID:
createDiary();
return true;
case DELETE_ID:
mDbHelper.deleteDiary(getListView().getSelectedItemId());
renderListView();
return true;
}
return super.onMenuItemSelected(featureId, item);
}
```

代码解释：

- 如果点击添加一篇新日记按钮那么会执行到 createDiary();
- 如果点击删除一条记录，会执行：
- mDbHelper.deleteDiary(getListView().getSelectedItemId()); 首先先删除当前被选中的条列所对应的数据库当中的记录
- renderListView();重新对界面刷新。

在 createDiary()函数里边的代码如下所示：

```
private void createDiary() {
Intent i = new Intent(this, ActivityDiaryEdit.class);
startActivityForResult(i, ACTIVITY_CREATE);
}
```

代码解释：

- 首先构造了一个 intent,这个 intent 负责跳转到 ActivityDiaryEdit 里。
- 然后启动这个 intent，并且需要返回值。

5.第五步

在ActivityMain中，有多处地方都用到了renderListView()函数。在onCreate里边用这个函数显示ListView。当ListView需要发生变化后，例如，删除了一条记录或者增加了一条记录的时候，我们调用这个函数进行刷新ListView，下边来看一下此函数的实现，具体代码如下所示：

```
private void renderListView() {
mDiaryCursor = mDbHelper.getAllNotes();
startManagingCursor(mDiaryCursor);
String[] from = new String[] { DiaryDbAdapter.KEY_TITLE,
DiaryDbAdapter.KEY_CREATED };
int[] to = new int[] { R.id.text1, R.id.created };
SimpleCursorAdapter notes = new SimpleCursorAdapter(this,
R.layout.diary_row, mDiaryCursor, from, to);
setListAdapter(notes);
}
```

代码解释：

- mDiaryCursor = mDbHelper.getAllNotes();我们首先获取数据库当中的所有数据，这些数据以 Cursor 的形式存在。
- startManagingCursor(mDiaryCursor);我们将生成的 Cursor 交给 Activity 来管理，这样的好处是系统能自动

做很多事情，比如当程序暂停的时候，这个系统可以卸载 Cursor 以节省空间，当程序重新启动的时候系统重新查询生成 Cursor

- String[] from 里边定义了 ListView 每一排对应的数据是从数据库中的哪个列表里边取。
- 和 SimpleAdapter 类似 int[] to 里边是一个 View 的数组。这些 View 只能是 TextView 或者 ImageView。这些 View 是以 id 的形式来表示的，比如 Android.R.id.text1 。
- SimpleCursorAdapter notes = new SimpleCursorAdapter(this,R.layout.diary_row, mDiaryCursor, from, to);生成一个 SimpleCursorAdapter ，我们说一个下每一个参数的意义：
    - 第一个参数是 Context。
    - 第二个参数为 R.layout.diary_row，它关联在 diary_row.xml 文件当中定义的 Layout，这个 Layout 规定 ListView 当中每一项的布局。
    - 第三个参数就是 Cursor
    - 第四个参数是数据表当中的列的数组，只有在这里边出现的列名，数据才会对应的填充在 to 里边对应的 TextView 或者 ImageView 当中。
    - 第五个参数是在 LisView 里边每一项中需要被数据填充的 TextView 或者 ImageView。
- setListAdapter(notes);将 SimpleCursorAdapter 和 ListActivity 里边的 ListView 绑定起来，至此在界面当中才会显示出列表来。

小知识:

什么是 SimpleCursorAdapter ?

在第 7 章，我们已经介绍过了 ArrayAdapter 和 SimpleAdapter。和它们俩类似，SimpleCursorAdapter 也是集成 Adapter。ArrayAdapter 负责把一个字符串数组中的数据填充到一个 ListView 当中，而对应的 SimpleCursorAdapter 负责把 Cursor 里边的内容填充到 ListView 当中。通过 SimpleCursorAdapter 可以把数据库当中一列的数据和 ListView 中一排进行对应起来。和前两个 Adapter 类似，要求和数据进行对应的 View 必须是 TextView 或者 ImageView。

6.第六步

单击添加一条数据的按钮，程序运行界面如图 8-4-13 所示。

图 8-4-13 新建一篇日记的界面

这个界面对应的 Activity 是 ActivityDiaryEdit，对应的布局文件是 diary_edit.xml。对于这个布局文件里边用到了 LinearLayout、TextView 和 EditText，这些我们都已经讲过，在这里不在赘述，具体看一下代码：

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
Android:orientation="vertical" Android:layout_width="fill_parent"
Android:layout_height="fill_parent">

<LinearLayout Android:orientation="vertical"
Android:layout_width="fill_parent"
```

```
Android:layout_height="wrap_content">

<TextView Android:layout_width="wrap_content"
Android:layout_height="wrap_content" Android:text="@string/title"
Android:padding="2px" />
<EditText Android:id="@+id/title"
Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:layout_weight="1" />
</LinearLayout>

<TextView Android:layout_width="wrap_content"
Android:layout_height="wrap_content" Android:text="@string/body" />
<EditText Android:id="@+id/body" Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:layout_weight="1"
Android:scrollbars="vertical" />

<Button Android:id="@+id/confirm" Android:text="@string/confirm"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content" />

</LinearLayout>
```

当程序运行输入内容后，单击确定按钮，日记就会保存到数据库当中。下边来看一下代码具体是怎么执行的。 当但击确定按钮后，系统回调执行和按钮绑定的单击监听器里边的 onClick 方法，具体代码如下所示：

```
public void onClick(View view) {
String title = mTitleText.getText().toString();
String body = mBodyText.getText().toString();
if (mRowId != null) {
mDbHelper.updateDiary(mRowId, title, body);
} else
mDbHelper.createDiary(title, body);
Intent mIntent = new Intent();
setResult(RESULT_OK, mIntent);
finish();
}
```

代码解释：

- 首先获得 EditView 里边的数据。
- 目前 mRowId 为 null，所以执行 mDbHelper.createDiary(title, body);将数据保存到数据当中。
- setResult(RESULT_OK, mIntent);设置返回值。

执行完上边的代码后，系统跳转到 ActivityMain，并执行回调函数 onActivityResult，具体代码如下所示：

```
protected void onActivityResult(int requestCode, int resultCode,
Intent intent) {
super.onActivityResult(requestCode, resultCode, intent);
renderListView();
}
```

代码解释：

- 在回调函数中，我们重新对 ListView 进行刷新，将所有的数据重新显示出。

7.第七步

单击ListView里边的条列，可以对刚才保存的数据进行编辑。具体怎么实现这个功能的，我们可以看一下ActivityMain当中的onListItemClick方法代码：

```
protected void onListItemClick(ListView l, View v, int position, long id) {
super.onListItemClick(l, v, position, id);
Cursor c = mDiaryCursor;
c.moveToPosition(position);
Intent i = new Intent(this, ActivityDiaryEdit.class);
i.putExtra(DiaryDbAdapter.KEY_ROWID, id);
i.putExtra(DiaryDbAdapter.KEY_TITLE, c.getString(c
.getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE)));
i.putExtra(DiaryDbAdapter.KEY_BODY, c.getString(c
.getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY)));
startActivityForResult(i, ACTIVITY_EDIT);
```

}
代码解释：

- c.moveToPosition(position);将在 Cursor 当中的指针移到 position 位置，这个 position 是我们点击的这个一列在整个列表中的位置。
- Intent i = new Intent(this, ActivityDiaryEdit.class);构造一个跳转到 ActivityDiaryEdit 的 intent.
- putExtra()方法负责将要传递的数据放到 intent 当中。
- c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE))得到这一条数据中列名为 title 的值。
- c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY))得到这一条数据中列名为 body 的值。
- startActivityForResult(i, ACTIVITY_EDIT);启动 intent,发生 Activity 的跳转。

我们来看一下 ActivityDiaryEdit 中的 onCreate()里边的代码：

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
String title = extras.getString(DiaryDbAdapter.KEY_TITLE);
String body = extras.getString(DiaryDbAdapter.KEY_BODY);
mRowId = extras.getLong(DiaryDbAdapter.KEY_ROWID);

if (title != null) {
mTitleText.setText(title);
}
if (body != null) {
mBodyText.setText(body);
}
}
```

代码解释：

- 对于 ActivityDiaryEdit 这个 Activity 有两个 intent 可以跳转进来，一个是新建一篇日记的时候，这个时候 intent 里边的 extras 部分没有任何数据。第二中情况是在点击列表的某一个条列的时候，这个时候的 intent 如上所示会携带 extras 数据。所以在 ActivityDiaryEdit 中我们通过判断 extras 是否为 null,就可以判断是那种 intent 启动的。
- 当 extras 不为 null,我们将 extras 里边的数据显示出来。

8.第八步

在程序的主界面当中，上下移动焦点（可以通过键盘的上下键或者模拟器中的上下按键），可以对拥有焦点的那一项进行删除，如图 8-4-14、图 8-4-15、图 8-4-16 所示。

图 8-4-14 上下移动焦点

图 8-4-15 进行删除

图 8-4-16 删除后

具体执行为 onMenuItemSelected 中的代码：

```
mDbHelper.deleteDiary(getListView().getSelectedItemId());
renderListView();
```

代码解释：

- getListView()方法获取当前的 ListView 引用。
- getSelectedItemId()方法得到当前这一列所对应的数据项的 rowId,也就是这条数据在数据库中的主键 id。
- mDbHelper.deleteDiary()方法删去数据库中的这一列数据。
- renderListView()负责对列表重新刷新一遍。

至此，对数据库的学习就先告一段落，接下来将学习 contentProvider，它是 Android 应用当中非常重要的一部分。而且程序间的大部分数据交换都是通过 contentProvider 机制进行。

8.5 ContentProvider 方式

8.5.1 初识 ContentProvider

在第 6 章当中，介绍了组成 Android 程序的主要 4 部分，它们分别是。

- Activity。
- Broadcast Intent Receiver。
- Service。
- Content Provider。

关于 Actvity 和相关 View 的部分，已经在前边章节进行了比较详细的介绍，在这一节中，将学习 Android 应用里另外一个非常重要的部分 contentProvider。

1.什么是 ContentProvider

Android这个系统和其它的操作系统还不太一样，读者需要记住的是，数据在Android当中是私有的，当然这些数据包括文件数据和数据库数据以及一些其他类型的数据。那这个时候有读者就会提出问题，难道两个程序之间就没有办法对于数据进行交换？Android这么优秀的系统不会让这种情况发生的。解决这个问题主要靠ContentProvider。一个Content Provider类实现了一组标准的方法接口，从而能够让其他的应用保存或读取此Content Provider的各种数据类型。也就是说，一个程序可以通过实现一个content Provider的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个应用暴露的数据在应用当中是如何存储的，或者是用数据库存储还是用文件存储，还是通过网上获得，这些一切都不重要，重要的是外界可以通过这一套标准及统一的接口和程序里的数据打交道，可以读取程序的数据，也可以删除程序的数据，当然，中间也会涉及到一些权限的问题。下边列举一些较常见的接口，这些接口如下所示。

- query(Uri uri, String[] projection, String selection, String[] selectionArgs,String sortOrder):通过 Uri 进行查询，返回一个 Cursor。
- insert(Uri url, ContentValues values):将一组数据插入到 Uri 指定的地方。
- update(Uri uri, ContentValues values, String where, String[] selectionArgs) 更新 Uri 指定位置的数据。
- delete(Uri url, String where, String[] selectionArgs) 删除指定 Uri 并且符合一定条件的数据

2.什么是ContentResolver

外界的程序通过 ContentResolver 接口可以访问 ContentProvider 提供的数据，在 Activity 当中通过 getContentResolver()可以得到当前应用的 ContentResolver 实例。ContentResolver 提供的接口和 ContentProvider 中需要实现的接口对应，主要有一下几个。

- query(Uri uri, String[] projection, String selection, String[] selectionArgs,String sortOrder):通过 Uri 进行查询，返回一个 Cursor。
- insert(Uri url, ContentValues values):将一组数据插入到Uri 指定的地方。
- update(Uri uri, ContentValues values, String where, String[] selectionArgs) 更新 Uri 指定位置的数据

- delete(Uri url, String where, String[] selectionArgs) 删除指定 Uri 并且符合一定条件的数据

3.ContentProvider 和 ContentResolver 中用到的 Uri

在 ContentProvider 和 ContentResolver 当中用到了 Uri 的形式通常有两种，一种是指定全部数据，另一种是指定某个 ID 的数据。我们看下面的例子。

- **content://contacts/people/** 这个 Uri 指定的就是全部的联系人数据。
- **content://contacts/people/1** 这个 Uri 指定的是 ID 为 1 的联系人的数据。

在上边两个类中用到的 Uri 一般由 3 部分组成

- 第一部分是："content://" 。
- 第二部分是要获得数据的一个字符串片段。
- 最后就是 ID（如果没有指定 ID，那么表示返回全部）。

由于 URI 经常比较长，而且有时候容易出错，且难以理解。所以，在 Android 当中定义了一些辅助类，并且定义了一些常量来代替这些长字符串的使用，例如下边的代码：

- Contacts.People.CONTENT_URI （联系人的URI）

8.5.2 使用 contentProvider 读取系统数据

在这个例子里边，首先在系统的联系人应用当中插入一些联系人信息，然后把这些联系人的名字和电话再显示出来，通过这个例子可以学到。

- 如何在联系人应用当中添加联系人。
- 如何使用系统提供的 ContentProvder。
- 如何使用 ContentResolver 当中的 query()方法。

具体实现步骤如下所示。

1.第一步

在 Eclipse 中打开 ex09_1_ContentProvider 项目，具体操作如下。

（1）新建一个项目，依次单击 File > New > Android Project 项。
（2）在新建项目的对话框中，选择 Create project from existing source。
（3）单击浏览按钮，找到 ex09_1_ContentProvider 项目，然后单击确定按钮。

程序的目录结构如图 8-5-1 所示：

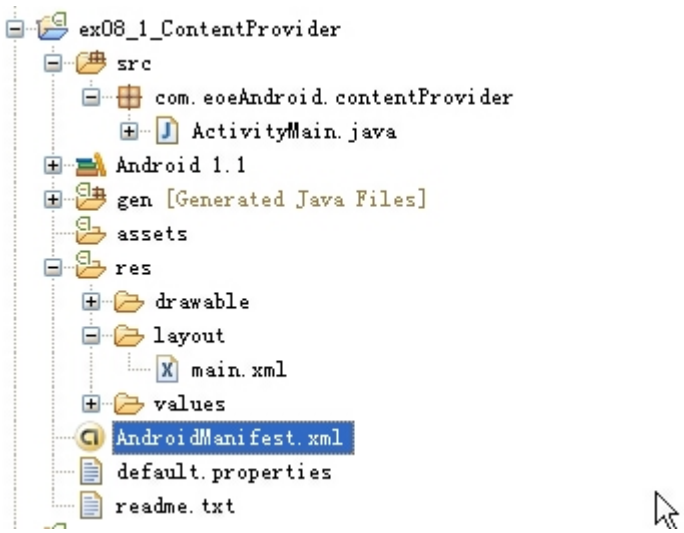


图 8-5-1 程序的目录结构

2.第二步

首先运行这个项目，将会看到如图 8-5-2 所示的界面。

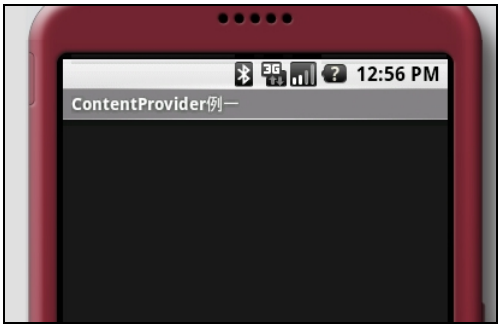


图 8-5-2 未添加任何数据的主界面

图 8-18 所示的列表中没有任何数据，接下来的操作是为应用添加几条联系人数据。

3.第三步:

按照下列图示添加几条数据到联系人列表中,具体步骤如下。

（1）单击模拟器的 home 键，在转出来的界面上，单击桌面上的 Contacts 应用，如图 8-5-3 所示。

图 8-5-3 点击 Contacts 应用

（2）进入应用后，单击 Menu 项，在出现的界面上单击 New contact 按钮，如图 8-5-4 所示。

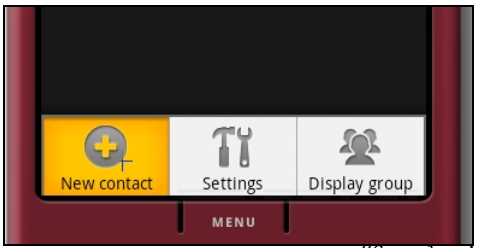


图 8-5-4 单击 New contact 选项

（3）添加联系人姓名和电话号码信息，如图 8-5-5 所示。

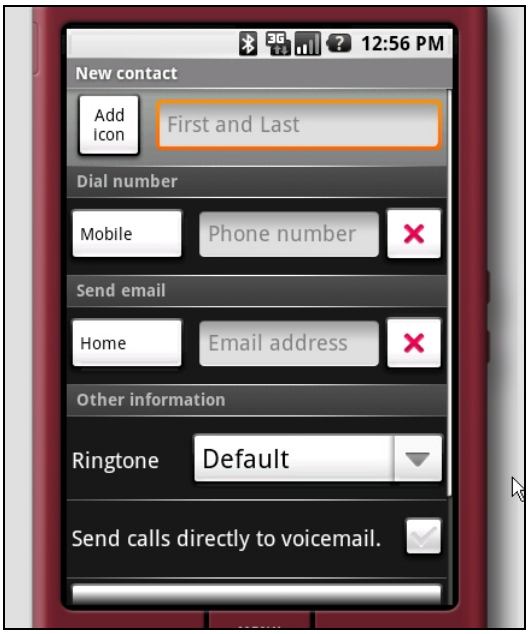


图 8-5-5 添加联系人姓名和电话号码

（4）单击 Menu 项，在返回的界面上单击 Save 项保存，如图 8-5-6 所示。

图 8-5-6 保存联系人姓名和电话号码信息

（5）按照上边的操作步骤，添加了两条数据后显示如下图 8-5-7。

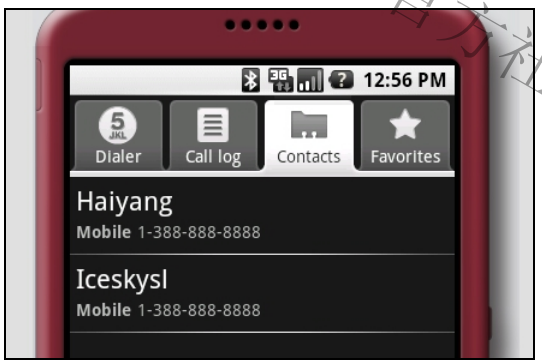


图 8-5-7 添加后结果

4.第四步
再次运行程序，模拟器显示如图 8-5-8 所示。

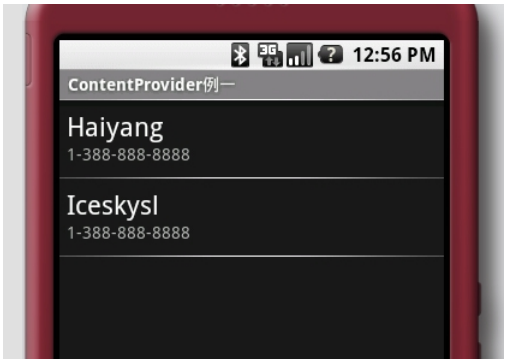


图 8-5-8 模拟器显示

我们看一下程序 ActivityMain 中的 onCreate()方法，具体代码如下所示：

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Cursor c = getContentResolver().query(Phones.CONTENT_URI, null, null, null, null);
        startManagingCursor(c);
        ListAdapter adapter = new SimpleCursorAdapter(this,
                Android.R.layout.simple_list_item_2, c,
                        new String[] { Phones.NAME, Phones.NUMBER },
                        new int[] { Android.R.id.text1, Android.R.id.text2 });
        setListAdapter(adapter);
    }
```

代码解释：

- getContentResolver()方法得到应用的 ContentResolver 实例。
- query(Phones.CONTENT_URI, null, null, null, null)。它是 ContentResolver 里的方法，负责查询所有联系人，并返回一个 Cursor。这个方法参数比较多，每个参数的具体含义如下。
  - 第一个参数为 Uri,在这个例子里边这个 Uri 是联系人的 Uri.。
  - 第二个参数是一个字符串的数组，数组里边的每一个字符串都是数据表中某一列的名字，它指定返回数据表中那些列的值。
  - 第三个参数相当于 sql 语句的 where 部分，描述哪些值是我们需要的。
  - 第四个参数是一个字符串数组，它里边的值依次代替在第三个参数中出现的?符号。
  - 第五个参数指定了排序的方式。
- startManagingCursor(c); 让系统来管理生成的 Cursor.
- ListAdapter adapter = new SimpleCursorAdapter(this,Android.R.layout.simple_list_item_2, c, new String[] { Phones.NAME, Phones.NUMBER }, new int[] { Android.R.id.text1, Android.R.id.text2 });生成一个 SimpleCursorAdapter。（关于 SimpleCursorAdapter 我们在第 7 章已经详细介绍过了）
- setListAdapter(adapter)。将 ListView 和 SimpleCursorAdapter 进行绑定。

8.5.3 使用 ContentProvider 操作数据

在上一个例子当中学习了 ContentProvider，并且使用了系统中的一个联系人的 ContentProvider，在本节当中，我们仍然实现类似第 8 章所讲的日记本的例子，只不过这次我们用 ContentProvider 实现，而不是直接用数据库实现。这样外界的程序就可以访问得到日记本这个应用数据。

通过这个例子可以学到。

- 如何实现一个 ContentProvider。
- 理解 UriMatcher 的含义。
- onPrepareOptionsMenu 方法介绍

具体实现步骤如下所示。

1.第一步

在 Eclipse 中打开 ex09_2_contentProvider 项目 ，具体操作步骤如下。

（1）新建一个项目，依次单击 File → New → Android Project 项。
（2）在新建项目的对话框中，选择 Create project from existing source 项。
（3）单击浏览项，找到 ex09_2_contentProvider 项目，然后单击确定。

程序的目录结构如图 8-5-9

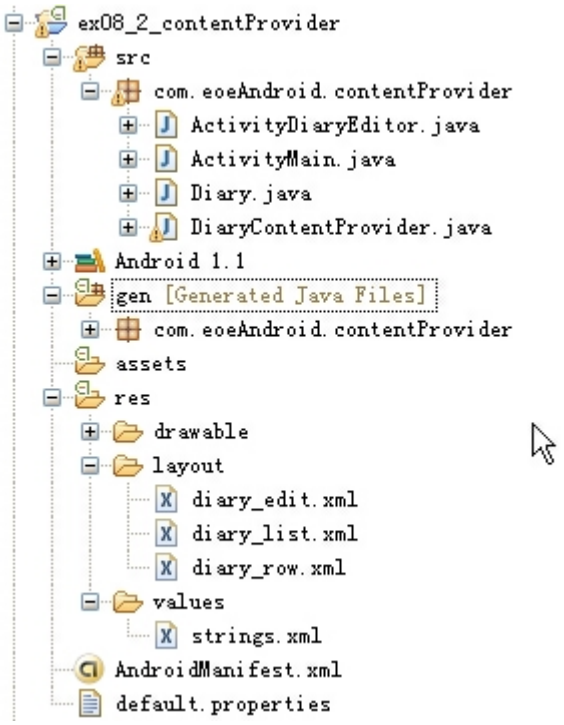


图 8-5-9 程序的目录结构

2.第二步

首先运行这个项目，将会看到如图 8-5-10 所示界面。

图 8-5-10 未添加任何数据的主界面

图 8-5-10 所示的这个界面我们在第 8 章的例子里边已经见过。本例的主 activity 仍然是一个 ListActivity，而且关联的布局文件也是 diary_list.xml，关于 ListActivity 的使用方法和详细说明可以参见第 7 章第 6 节。diary_list.xml 的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content">

<ListView Android:id="@+id/Android:list"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content" />
<TextView Android:id="@+id/Android:empty"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content" Android:text="您还没有开始写日记呢!点击下边的 Menu 按钮开始写日记吧:)" />
</LinearLayout>
```

3.第三步

和第 8 章的日记本例子一样，日记本的数据还是存储在 SQLite 数据库中，但是不一样的是，在这个例子里，执行增、删、改、查操作时不是直接访问数据库，而是通过日记本程序的 ContentProivder 来实现。

我们先来看一下 Diary 这个类，这个类里边有一个内部静态类 DiaryColumns，和它的名字意思一样，这个类里边主要定义了日记本数据库的列表字段的名字，具体代码如下所示：

```
public static final class DiaryColumns implements BaseColumns {
        private DiaryColumns() {}
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/diaries");
        public static final String CONTENT_TYPE = "vnd.Android.cursor.dir/vnd.google.diary";
        public static final String CONTENT_ITEM_TYPE = "vnd.Android.cursor.item/vnd.google.diary";
        public static final String DEFAULT_SORT_ORDER = "created DESC";
        public static final String TITLE = "title";
        public static final String BODY = "body";
        public static final String CREATED = "created";
    }
```

代码解释：

- BaseColumns 是一个接口，里边有两个变量，一个是_ID=“_id” ，一个是_COUNT="_count" 。在 Android 当中，每一个张数据库表至少有一个字段，而且这个字段是_id.所以当我们构造列名的辅助类时，直接实现 BaseColumns ，这样我们便默认的拥有了_id 字段。
- 在我们的日记本的数据表里边，一共有四个字段，分别是：“id”、"title"、"body"、"created"。

小知识：

在 Android 的设计“哲学”里边是鼓励开发者使用内部类的，这样不但使用方便，而且执行效率也高。

4.第四步

现在我们来看 DiaryContentProvider 类，这个类继承自 ContentProvider，在这个类里边实现了 ContentProvider 的一些接口方法，并且成为日记本的一个 ContentProvider。我们现在通过学习这个类，来详细了解实现一个自定义的 ContentProvider 的具体步骤。

（1）继承 ContentProvider.DiaryContentProvider 类是继承 ContentProvider 类的。
（2）定义一个 public static final 的 Uri 类型的变量，并且命名为 CONTENT_URI。DiaryContentProvider 所能处理的 Uri 都是基于 CONTENT_URI 来构建的，需要注意的是，这个 CONTENT_URI 中的内容以 content://开头，并且全部小写，且全局惟一。

下边是在这个例子中的一个普通 URI，通过分析这个 URI，可以了解 content URI 的构成，代码如下所示：

content://com.ex09_2_contentprovider.diarycontentprovider/diaries/1

代码解释：

●第一部分是 content://，这部分是支持存在的，也是不用做什么修改的。
●第二部分是授权（AUTHORITY）部分，在这个例子里边就是com.ex09_2_contentprovider.diarycontentprovider，授权部分是惟一的，一般为在程序是我们实现的那个ContentProvider的全称，并且全都小写。这部份是和在AndroidManifest.xml文件当中的<provider
Android:name="DiaryContentProvider" Android:authorities="com.ex09_2_contentprovider.diarycontentprovider" />部分对应的。
●第三部分是请求数据的类型。例如，在这个例子当中定义的类型是 diaries。当然这一部分可以是 0 个片段或者多个片段构成，如果 content provider 只是暴露出了一种类型的数据，那么这部分可以为空，但是如果暴露出了多种，尤其是包含子类的的时候，就不能为空。例如，日记本程序里边可以暴露出来两种数据，一种是用户自己的"diaries/my"，另一种是其他人的"diaries/others。
●第四部分就是“1”，当然这部分是允许为空的。如果为空，表示请求全部数据；如果不为空，表示请求特定ID的数据。

3.构建用户的数据存储系统。在这个例子当中，是将数据存储到数据库系统当中。通常是将数据存储在数据库系统中，但是也可以将数据存储在其他的地方，如文件系统等。
4.实现 ContentProvider 这个抽象类的抽象方法，具体如下所示：

- public boolean onCreate()，当 ContentProvider 生成的时候调用此方法。
- public Cursor query(Uri uri, String[] projection, String selection,String[] selectionArgs, String sortOrder) ，此方法返回一个 Cursor 对象作为查询结果集。
- public Uri insert(Uri uri, ContentValues initialValues)，此方法负责往数据集当中插入一列，并返回这一列的 Uri。
- public int delete(Uri uri, String where, String[] whereArgs)，此方法负责删除指定 Uri 的数据。
- public int update(Uri uri, ContentValues values, String where,String[] whereArgs) ，此方法负责更新指定 Uri 的数据。
- public String getType(Uri uri) ，返回所给 Uri 的 MIME 类型。

5.在 AndroidManifest.xml 文件中增加<provider>标签。例如，在这个例子中，实现的代码为：<provider Android:name="DiaryContentProvider" Android:authorities="com.ex09_2_contentprovider.diarycontentprovider" />， 其中 Android:name 为我们实现的这个 content provider 的类名；Android:authorities 为 content URI 的第二部分，即授权部分，代码为："com.ex09_2_contentprovider.diarycontentprovider"。

5.第五步：
继续来看 DiaryContentProvider 类的代码。此实例的数据是存储在数据库当中，和前面章节学过的数据例子一样，此实例中我们定义了 DatabaseHelper 辅助类。这个类在第 8 章中有详细的讲解，这里就不再进行详细说明，具体代码如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {

DatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE " + DIARY_TABLE_NAME + " ("
+ DiaryColumns._ID + " INTEGER PRIMARY KEY,"
+ DiaryColumns.TITLE + " TEXT," + DiaryColumns.BODY
+ " TEXT," + DiaryColumns.CREATED + " TEXT" + ");");
```

```
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
db.execSQL("DROP TABLE IF EXISTS notes");
onCreate(db);
}
}
```

在 DiaryContentProvider 中，我们定义了一些变量和常量，其中这些常量主要是描述数据库的的信息。

```
private static final String DATABASE_NAME = "database";
private static final int DATABASE_VERSION = 1;
private static final String DIARY_TABLE_NAME = "diary";

private static HashMap<String, String> sDiariesProjectionMap;

private static final int DIARIES = 1;
private static final int DIARY_ID = 2;

private static final UriMatcher sUriMatcher;
```

小知识：

什么是 UriMatcher?

UriMatcher 是匹配 Uri 的一个辅助类。例如，在我们的 DiaryContentProvider 中的 static 模块中，有下边的代码：

```
sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries", DIARIES);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID);
```

代码解释：

- sUriMatcher.addURI(Diary.AUTHORITY, "diaries", sUriMatcher.match(uri))表示，如果我们的 Uri 是 content://com.ex09_2_contentprovider.diarycontentprovider/diaries/，那么 sUriMatcher.match(uri) 的返回值就是 DIARIES。
- sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID)表示，如果我们的 Uri 是 content://com.ex09_2_contentprovider.diarycontentprovider/diaries/id（其中后边的 id 是一个数字），那么 sUriMatcher.match(uri)的返回值就是 DIARY_ID。

通过 UriMatcher 类我们可以很方便的判断一个 URi 的类型，特别是判断这个 Uri 是对单个数据的请求，还是对全部数据的请求。

6.第六步

现在我们来看一下在第四步列出来的抽象方法具体是怎么实现的。我们就从增、删、改、查的顺序说起。

（1） 首先来看插入的方法：insert()。

具体实现代码如下所示：

```
@Override
public Uri insert(Uri uri, ContentValues initialValues) {
if (sUriMatcher.match(uri) != DIARIES) {
throw new IllegalArgumentException("Unknown URI " + uri);
}

ContentValues values;
if (initialValues != null) {
values = new ContentValues(initialValues);
} else {
values = new ContentValues();
}

if (values.containsKey(Diary.DiaryColumns.CREATED) == false) {
values.put(Diary.DiaryColumns.CREATED, getFormateCreatedDate());
}
```

```
if (values.containsKey(Diary.DiaryColumns.TITLE) == false) {
Resources r = Resources.getSystem();
values.put(Diary.DiaryColumns.TITLE, r
.getString(Android.R.string.untitled));
}

if (values.containsKey(Diary.DiaryColumns.BODY) == false) {
values.put(Diary.DiaryColumns.BODY, "");
}

SQLiteDatabase db = mOpenHelper.getWritableDatabase();
long rowId = db.insert(DIARY_TABLE_NAME, DiaryColumns.BODY, values);
if (rowId > 0) {
Uri diaryUri= ContentUris.withAppendedId(
Diary.DiaryColumns.CONTENT_URI, rowId);
return diaryUri;
}

throw new SQLException("Failed to insert row into " + uri);
}
```

代码解释：

- 首先我们通过语句 sUriMatcher.match(uri) != DIARIES 对传进来的 Uri 进行了判断，如果这个 Uri 不是 DIARIES 类型的，那么这个 Uri 就是一个非法的 Uri。
- SQLiteDatabase db = mOpenHelper.getWritableDatabase()语句负责得到一个 SQLiteDatabase 的实例。
- db.insert(DIARY_TABLE_NAME, DiaryColumns.BODY, values)语句负责插入一条记录到数据库中。
- 需要注意的是，insert()返回的是一个 Uri，而不是一个记录的 id，所以我们还应该把记录的 id 构造成一个 Uri:Uri diaryUri= ContentUris.withAppendedId(Diary.DiaryColumns.CONTENT_URI, rowId).withAppendedId()方法在下边的小知识当中详细介绍。

小知识:

什么是 ContentUris?

ContentUris 是 content URI 的一个辅助类。它有两个方法很有用，具体如下所示。

- public static Uri withAppendedId(Uri contentUri, long id)，这个方法负责把 id 和 contentUri 连接成一个新的 Uri。比如在我们这个例子当中是这么使用的：ContentUris.withAppendedId(Diary.DiaryColumns.CONTENT_URI, rowId). 如果 rowId 为 100 的话，那么现在的这个 Uri 的内容就是：content://com.ex09_2_contentprovider.diarycontentprovider/diaries/100.
- public static long parseId(Uri contentUri),这个方法负责把 content URI 后边的 id 解析出来.比如现在这个 content URI 是 content://com.ex09_2_contentprovider.diarycontentprovider/diaries/100,那么这个函数的返回值就是 100.

（2） 现在来看删除方法：delete()。

具体实现代码如下所示：

```
@Override
public int delete(Uri uri, String where, String[] whereArgs) {
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
String rowId = uri.getPathSegments().get(1);
return db
.delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null);
}
```

代码解释：

- rowId = uri.getPathSegments().get(1)负责得到 rowId 的值.getPathSegments()方法得到一个 String 的 List,在我们例子当中 uri.getPathSegments().get(1)为 rowId,如果是 uri.getPathSegments().get(0)那就值为"diaries".
- db.delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null)是标准的 SQLite 删除操作.第一个参数数据表的名字,第二个参数相当于 sql 语句当中的 where 部分。

（3） 更新一条数据的方法：update()。
具体实现代码如下所示：

```
@Override
public int update(Uri uri, ContentValues values, String where,
String[] whereArgs) {
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
String rowId = uri.getPathSegments().get(1);
return db.update(DIARY_TABLE_NAME, values, DiaryColumns._ID + "="
+ rowId, null);

}
```

代码解释：

- 首先得到 SQLiteDatabase 的实例，然后得到 rowId.最后再调用 db.update(DIARY_TABLE_NAME, values, DiaryColumns._ID + "="+ rowId, null);执行更新工作。

（4） 插入一条数据的方法：query()。
具体实现代码如下所示：

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

switch (sUriMatcher.match(uri)) {
case DIARIES:
qb.setTables(DIARY_TABLE_NAME);
break;

case DIARY_ID:
qb.setTables(DIARY_TABLE_NAME);
qb.appendWhere(DiaryColumns._ID + "="
+ uri.getPathSegments().get(1));
break;

default:
throw new IllegalArgumentException("Unknown URI " + uri);
}

String orderBy;
if (TextUtils.isEmpty(sortOrder)) {
orderBy = Diary.DiaryColumns.DEFAULT_SORT_ORDER;
} else {
orderBy = sortOrder;
}

SQLiteDatabase db = mOpenHelper.getReadableDatabase();
Cursor c = qb.query(db, projection, selection, selectionArgs, null,
null, orderBy);
return c;
}
```

代码解释：

- SQLiteQueryBuilder 是一个构造 SQL 查询语句的辅助类。
- sUriMatcher.match(uri)，根据返回值可以判断这次查询请求时，它是请求全部数据还是某个 id 的数据。
- 如果返回值是 DIARIES，那么只需要执行 qb.setTables(DIARY_TABLE_NAME)语句就可以了。
- 如果返回值是 DIARY_ID，那么还需要将 where 部分的参数设置进去，代码为 qb.appendWhere(DiaryColumns._ID

+ "="+ uri.getPathSegments().get(1))。

- SQLiteDatabase db = mOpenHelper.getReadableDatabase(),得到一个可读的 SQLiteDatabase 实例。
- Cursor c = qb.query(db, projection, selection, selectionArgs, null,null, orderBy);这个查询类似于一个标准的 SQL 查询，但是这个查询是 SQLiteQueryBuilder 来发起的，而不是 SQLiteDatabase 直接发起的，所以在参数方面略有不同。这个函数为 query(SQLiteDatabase db, String[] projectionIn, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder, String limit)下边将各个参数介绍一下。
    - 第一个参数为要查询的数据库实例。
    - 第二个参数是一个字符串数组，里边的每一项代表了需要返回的列名。
    - 第三个参数相当于 sql 语句中的 where 部分。
    - 第四个参数是一个字符串数组，里边的每一项依次替代在第三个参数中出现的问号（?）
    - 第五个参数相当于 sql 语句当中的 groupby 部分。
    - 第六个参数相当于 sql 语句当中的 having 部分。
    - 第七个参数描述是怎么进行排序。
    - 第八个参数相当于 sql 当中的 limit 部分。控制返回的数据的个数。

在DiaryContentProvider类里边还有两个方法，一个是getType(Uri uri) ，另外一个是getFormateCreatedDate()。后者根据时间得到一个我们特定格式的字符串，比较简单。而前者是必须要重写的方法。 下边我们看一下我们如何重写了getType方法。

```
public String getType(Uri uri) {
switch (sUriMatcher.match(uri)) {
case DIARIES:
return DiaryColumns.CONTENT_TYPE;

case DIARY_ID:
return DiaryColumns.CONTENT_ITEM_TYPE;

default:
throw new IllegalArgumentException("Unknown URI " + uri);
}
}
```

代码解释：

- 此方法返回一个所给 Uri 的指定数据的 MIME 类型。它的返回值如果以 vnd.Android.cursor.item 开头，那么就代表这个 Uri 指定的是单条数据。如果是以 vnd.Android.cursor.dir 开头的话，那么说明这个 Uri 指定的是全部数据。

7.第七步

在程序主界面但击 Menu 键，程序运行界面如图 8-5-11 所示。

图 8-5-11 单击 Menu 键

在 ActivityMain 中执行的代码如下所示：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
super.onCreateOptionsMenu(menu);
menu.add(0, MENU_ITEM_INSERT, 0, R.string.menu_insert);
return true;
}
```

代码解释：

- 我们给菜单(menu)添加了一项，如图 8-5-11 所示。

单击添加日记按钮后进入如图 8-5-12 所示界面。

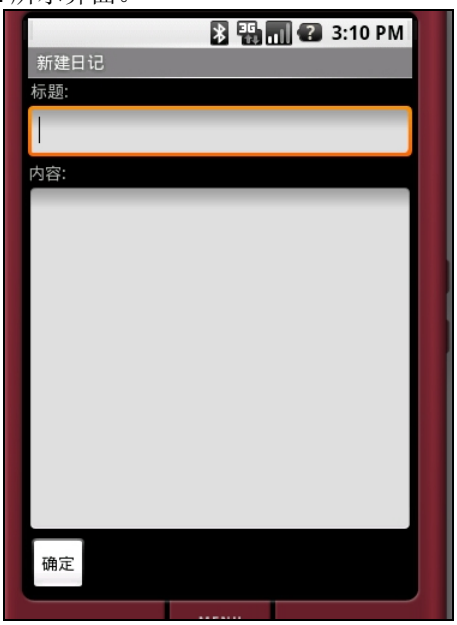


图 8-5-12 新建日记的页面

图 8-5-12 所示的这个页面的布局和第 8 章日记本程序里的布局完全一样，关于布局，读者可以参看第 8 章日记本程序例子的讲解。

下面来看一下在 ActivityDiaryEditor 中，是怎么处理两种不同情况进入日记本程序界面的。一种是通过新建日记进入此界面，另一种是经过编辑日记进入此日记运行界面。下面是在 ActivityDiaryEditor 程序中使用的 onCreate()函数，其代码如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setTheme(Android.R.style.Theme_Black);
final Intent intent = getIntent();
final String action = intent.getAction();
setContentView(R.layout.diary_edit);
mTitleText = (EditText) findViewById(R.id.title);
mBodyText = (EditText) findViewById(R.id.body);
confirmButton = (Button) findViewById(R.id.confirm);

if (EDIT_DIARY_ACTION.equals(action)) {// 编辑日记
mState = STATE_EDIT;
mUri = intent.getData();
mCursor = managedQuery(mUri, PROJECTION, null, null, null);
mCursor.moveToFirst();
String title = mCursor.getString(1);
mTitleText.setTextKeepState(title);
String body = mCursor.getString(2);
mBodyText.setTextKeepState(body);
setResult(RESULT_OK, (new Intent()).setAction(mUri.toString()));
setTitle("编辑日记");
} else if (INSERT_DIARY_ACTION.equals(action)) {// 新建日记
mState = STATE_INSERT;
setTitle("新建日记");
} else {
Log.e(TAG, "no such action error");
finish();
return;
}

confirmButton.setOnClickListener(new View.OnClickListener() {
```

```
public void onClick(View view) {
if (mState == STATE_INSERT) {
insertDiary();
} else {
updateDiary();
}
Intent mIntent = new Intent();
setResult(RESULT_OK, mIntent);
finish();
}

});

}
```

代码解释：

- 通过getIntent()得到启动当前Activity的那个Intent。
- 通过 intent.getAction()得到这个 intent 的动作(action)。在此操作中，当前的 Activiy 是通过单击先前的新建日记的按钮进来的，所以，看一下 ActivityMain 当中的 onOptionsItemSelected()函数，了解一下动作(action)是怎么设置的，此代码如下所示：

```
                Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
intent0.setData(getIntent().getData());
startActivity(intent0);
```

代码解释：

- 从上述代码可以看到，通过语句 intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION)设置的 action 是 ActivityDiaryEditor.INSERT_DIARY_ACTION)实现的。
- 在 ActivityDiaryEditor 中的 onCreate()函数中，当动作(action)为 INSERT_DIARY_ACTION 时候，我们执行 mState = STATE_INSERT，也就是标记当前的状态时插入状态。
- 如果当前的动作(action)是 EDIT_DIARY_ACTION，那么当前就是编辑状态：mState = STATE_EDIT。

当运行程序填充数据后单击确定按钮，执行 confirmButton 的点击监听器当中的 onClick()函数。这个函数根据不同的状态执行插入或者更新数据的操作。

按照现在的程序执行流程，单击确定按钮后，程序将执行插入操作，实现插入操作代码如下所示：

```
private void insertDiary() {
String title = mTitleText.getText().toString();
String body = mBodyText.getText().toString();
ContentValues values = new ContentValues();
values.put(Diary.DiaryColumns.CREATED, DiaryContentProvider
.getFormateCreatedDate());
values.put(Diary.DiaryColumns.TITLE, title);
values.put(Diary.DiaryColumns.BODY, body);
getContentResolver().insert(Diary.DiaryColumns.CONTENT_URI, values);

}
```

代码解释：

- 首先通过 getText()方法将编辑框中的数据都读出来。
- 将要插入的数据都放到一个 ContentValues 的实例当中。
- 调用 getContentResolver()得到当前应用的一个 ContentResolver 的实例。
- insert(Diary.DiaryColumns.CONTENT_URI, values)语句为 ContentResolver 的插入方法，这个方法有两个参数，一个参数是数据的 Uri，第二个参数是包含要插入数据的 ContentValues 的实例。执行这条语句的最终会调用 DiaryContentProvider 中的 insert()方法。

当单击确定按钮后，程序运行出现如图 8-5-13 所示的界面。

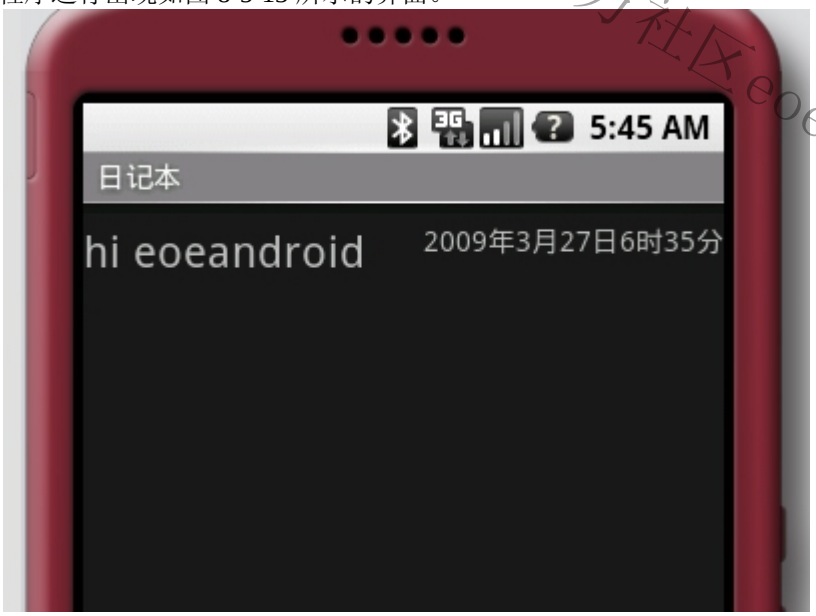


图 8-5-13 已经插入了一条数据

8.第八步

按方向键向下键将焦点移动到这条数据上，然后单击 Menu 键会出现如图 8-5-14 所示界面。

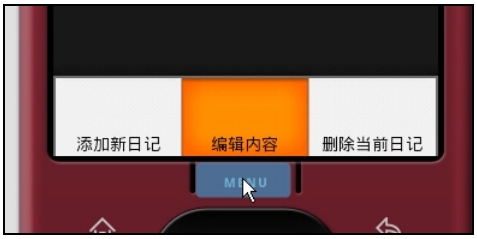


图 8-5-14 单击 Menu 键界面

图 8-5-14 所示的这个界面和刚才讲到的单击 Menu 键出现的界面一样，下面看一下实现的代码，以便了解此界面是怎么生成，具体实现代码如下所示：

```
public boolean onPrepareOptionsMenu(Menu menu) {
super.onPrepareOptionsMenu(menu);
final boolean haveItems = getListAdapter().getCount() > 0;
if (haveItems) {
if (getListView().getSelectedItemId() > 0) {
menu.removeGroup(1);
Uri uri = ContentUris.withAppendedId(getIntent().getData(),
getSelectedItemId());
Intent intent = new Intent(null, uri);
menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent);
menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记");
}

}else{
menu.removeGroup(1);
}
return true;
}
```

代码解释：

- 当前和这个 ListView 相关联的 ListAdapter 里边元素的个数大于零的时候 haveItems 为真，否则为假。
- menu.removeGroup(1)，如果 menu 里边有分组为 1 组的子项的话，那么就先全部删除。
- getIntent().getData()得到的 Uri 是 DiaryColumns.CONTENT_URI。

- 通过 ContentUris.withAppendedId(getIntent().getData(),getSelectedItemId())生成一个和 ListView 中获得焦点这一项的数据的 Uri。
- menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent)，在 menu 当中增加了一项“编辑内容”，并且这一项和一个 intent 关联，这个 intent 主要用于携带数据，它里边携带的就是一个 Uri 的数据，在下边的 onOptionsItemSelected()函数当中会用到。
- menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记")，增加另外一项。
- 如果 haveItems 为假，也就是当前和这个 ListView 相关联的 ListAdapter 里边元素的个数为零，即数据显示在 ListView 上的时候，点击 Menu 按钮的话，如果 menu 当中还有第 1 分组的子项的话，就给现删除了：menu.removeGroup(1)。

小知识:

单击 Menu 按钮的时候，在 Activity 中回调的函数可能有两个。
第一个是 onOptionsItemSelected(),这个函数只在第一次在当前应用当中单击 Menu 键的时候回调，以后再不回调。
第二个是 onPrepareOptionsMenu(),这个函数在每次点击 Menu 键后显示 menu 的时候被系统回调，每次 menu 显示前都会回调此函数。我们一般的根据条件改变 menu 显示的逻辑都放在这个函数里边。

当单击 Menu 键时出现 3 个按扭，单击其中的某一个按钮会触发 Android 系统回调 onOptionsItemSelected()函数，此函数实现代码如下所示：

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
// 插入一条数据
case MENU_ITEM_INSERT:
Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
intent0.setData(getIntent().getData());
startActivity(intent0);
return true;
// 编辑当前数据内容
case MENU_ITEM_EDIT:
Intent intent = new Intent(this, ActivityDiaryEditor.class);
intent.setData(item.getIntent().getData());
intent.setAction(ActivityDiaryEditor.EDIT_DIARY_ACTION);
startActivity(intent);
return true;
// 删除当前数据
case MENU_ITEM_DELETE:
Uri uri = ContentUris.withAppendedId(getIntent().getData(),
getListView().getSelectedItemId());
getContentResolver().delete(uri, null, null);
renderListView();

}
return super.onOptionsItemSelected(item);
}
```

代码解释：

- 当用户单击添加新日记按钮后，程序新建一个跳转 Activity 的 intent0，并且设置了 Action 和 data，这两个部分在程序跳转到 ActivityDiaryEditor 后会用到。
- 当用户单击了编辑按钮后，程序也新建了一个跳转 Activity 的 intent,并且也设置了 Action 和 data。同样，这两部分在程序跳转到 ActivityDiaryEditor 后会用到。需要注意的是，通过 item.getIntent().getData()得到了所需要的 Uri。
- 当用户单击删除按钮后，程序通过 ContentUris.withAppendedId(getIntent().getData(),getListView().getSelectedItemId())先得到需要删除数据的 Uri，然后得到当前的 ContentResolvor，然后调用它的 delete 方法进行删除。
- 删除数据后，调用 renderListView()函数对 ListView 进行及时刷新。

8.6 网络存储

前面介绍的几种存储都是将数据存储在本地设备上，除此之外，还有一种存储（获取）数据的方式，通过网络来实现数据的存储和获取，下面看一个在 Android 上调用 WebService 的例子。

注意：

在 Android 的早期版本中，曾经支持过进行 XMPP SERVICE 和 WEB SERVICE 的远程访问。Android SDK 1.0 以后的版本对它以前的 API 作了许多的变更。Android 1.0 以上版本不再支持 XMPP SERVICE，而且访问 WEB SERVIC 的 API 全部变更。

1.例子介绍

通过邮政编码查询该地区的天气预报，以POST发送的方式发送请求到webservicex.net站点，访问WebService.webservicex.net站点上提供查询天气预报的服务，具体信息请参考其WSDL文档，网址为：http://www.webservicex.net/WeatherForecast.asmx?WSDL。
输入：美国某个城市的邮政编码。
输出：该邮政编码对应城市的天气预报 。

2.实现步骤如下
（1）如果需要访问外部网络，则需要在AndroidManifest.xml文件中加入如下代码申请权限许可：

```
<!-- Permissions -->
<uses-permission Android:name="Android.permission.INTERNET" />
```

（2）以 HTTP POST 的方式发送（注意：SERVER_URL 并不是指 WSDL 的 URL，而是服务本身的 URL）。实现的代码如下所示：

```
private static final String SERVER_URL =
"http://www.webservicex.net/WeatherForecast.asmx/GetWeatherByZipCode"; //定义需要获取的内容来源地址
HttpPost request = new HttpPost(SERVER_URL); //根据内容来源地址创建一个 Http 请求
// Add Parameters
List <NameValuePair> params = new ArrayList <NameValuePair>();
// set a zip code: WASHINGTON
params.add(new BasicNameValuePair("ZipCode", "200120"));   //添加必须的参数
request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8)); //设置参数的编码
try {
// Send request
HttpResponse httpResponse = new DefaultHttpClient().execute(request); //发送请求并获取反馈
//  解析返回的内容
if(httpResponse.getStatusLine().getStatusCode() != 404)
{
  String result = EntityUtils.toString(httpResponse.getEntity());
  Log.d(LOG_TAG, result);
}
} catch (Exception e) {
Log.e(LOG_TAG, e.getMessage());
}
```

代码解释：如上代码使用Http从webservicex获取ZipCode为“200120”（美国WASHINGTON D.C）的内容，其返回的内容如下：

```
<WeatherForecasts xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.webservicex.net">
    <Latitude>38.97571</Latitude>
    <Longitude>77.02825</Longitude>
    <AllocationFactor>0.024849</AllocationFactor>
    <FipsCode>11</FipsCode>
    <PlaceName>WASHINGTON</PlaceName>
    <StateCode>DC</StateCode>
    <Details>
```

```
    <WeatherData>
      <Day>Saturday, April 25, 2009</Day>
      <WeatherImage>http://forecast.weather.gov/images/wtf/sct.jpg</WeatherImage>
      <MaxTemperatureF>88</MaxTemperatureF>
      <MinTemperatureF>57</MinTemperatureF>
      <MaxTemperatureC>31</MaxTemperatureC>
      <MinTemperatureC>14</MinTemperatureC>
    </WeatherData>
    <WeatherData>
      <Day>Sunday, April 26, 2009</Day>
      <WeatherImage>http://forecast.weather.gov/images/wtf/few.jpg</WeatherImage>
      <MaxTemperatureF>89</MaxTemperatureF>
      <MinTemperatureF>60</MinTemperatureF>
      <MaxTemperatureC>32</MaxTemperatureC>
      <MinTemperatureC>16</MinTemperatureC>
    </WeatherData>
…..
  </Details>
</WeatherForecasts>
```

这个例子演示了如何在 Android 中通过网络获取数据，掌握该类内容，开发者需要熟悉 java.net.*，Android.net.*这两个包的内容，在这就不赘述了，请读者参阅相关文档。

8.7 本章小结

本章介绍了如何在 Android 中实现数据的存储和获取，详细介绍了轻量级的的 SharedPreferences，比较传统的文件存储和数据库存储，也介绍了 Android 特有的 ContentProvider 方式，最后以一个获取天气预报的例子，演示了如何通过网络获取 WebService 的数据。通过本章的介绍，读者应该比较全面地了解了 Andoid 中的存储数据的方式，至于在应用程序中使用哪一种数据存储方式取决于开发者的具体需求，通过这章的介绍，读者将会做出合适的选择。