第 3 天 Android 基础

第三章 数据存储和界面展现（下）

- 使用 SQliteDatabase 存储数据
- 使用 ListView 展现数据

1.1 Android 中创建数据库

1.1.1 SQLite 简介

SQLite 是一款内置到移动设备上的轻量型的数据库，是遵守 ACID（原子性、一致性、隔离性、持久性）的关联式数据库管理系统，多用于嵌入式系统中。

SQLite 数据库是无类型的，可以向一个 integer 的列中添加一个字符串，但它又支持常见的类型比如: NULL，VARCHAR, TEXT, INTEGER, BLOB, CLOB 等。

Android 系统内置了 SQLite，并提供了一系列 API 方便对其进行操作。

1.1.2 使用 SQLiteOpenHelper 创建数据库

SQLiteOpenHelper 是 Android 提供的一个抽象工具类，负责管理数据库的创建、升级工作。如果我们想创建数据库，就需要自定义一个类继承 SQLiteOpenHelper，然后覆写其中的抽象方法。

一、创建 SQLiteOpenHelper 类

【文件 1-1】 MySQLiteOpenHelper.java

```
1.  package com.itheima.android.sqlite.db;
2.
3. import android.content.Context;
4. import android.database.sqlite.SQLiteDatabase;
5. import android.database.sqlite.SQLiteDatabase.CursorFactory;
6. import android.database.sqlite.SQLiteOpenHelper;
7. import android.util.Log;
8. /**
9.  *
10.  * @author wzy 2015-10-23
11.  * 管理数据库的创建和升级
12.  *
13.  */
14. public class MySQLiteOpenHelper extends SQLiteOpenHelper {
15.
16. private static final String TAG = "MySQLiteOpenHelper";
```

```
17. //定义数据库文件名
18. public static final String TABLE_NAME = "myuser.db";
19. 
20. /**
21.  *
22.  * @param context
23.  * @param name 数据库文件的名称
24.  * @param factory null
25.  * @param version 数据库文件的版本
26.  */
27. private MySQLiteOpenHelper(Context context, String name, CursorFactory factory,
28.  int version) {
29.     super(context, name, factory, version);
30. }
31. /**
32.  * 对外提供构造函数
33.  * @param context
34.  * @param version
35.  */
36. public MySQLiteOpenHelper(Context context,int version){
37.     //调用该类中的私有构造函数
38.     this(context, TABLE_NAME, null, version);
39. }
40. /*
41.  * 当第一次创建数据的时候回调方法
42.  */
43. @Override
44. public void onCreate(SQLiteDatabase db) {
45.     Log.d(TAG, "onCreate");
46.     //创建数据库表的语句
47.     String sql = "create table t_user(uid integer primary key not null,
48. c_name varchar(20),c_age integer,c_phone varchar(20))";
49.     db.execSQL(sql);
50. }
51. /*
52.  * 当数据库升级是回调该方法
53.  */
54. @Override
55. public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
56.     Log.d(TAG, "onUpgrade: oldVersion"+oldVersion+" newVersion="+newVersion);
57.     String sql = "alter table t_user add c_money float";
58.     db.execSQL(sql );
59. }
60. /*
61.  * 当数据库被打开时回调该方法
```

```
62.  */
63. @Override
64. public void onOpen(SQLiteDatabase db) {
65.     Log.d(TAG, "onOpen");
66. }
67.
68. }
```

注意：

上面的代码我们只是定义了一个 MySQLiteOpenHelper 类继承了 SQLiteOpenHelper 类。在 onCreate（）方法中通过执行 sql 语句实现表的创建。

二、使用 SQLiteOpenHelper 类

在 Activity 中可以执行如【文件 1-2】所示代码。如果第一次执行，则会创建一个数据库文件，创建的数据库文件位于/data/data/包名/databases/目录中。

【文件 1-2】 代码片段

```
1.  /*
2.           * 通过构造函数创建一个 MySQLiteOpenHelper 对象，
3.           * 此时数据库文件还未创建
4.           */
5.          MySQLiteOpenHelper openHelper = new MySQLiteOpenHelper(this, VERSION);
6.          /*
7.           * 调用一下任意一个方法可以是数据库文件得以创建
8.           */
9.          openHelper.getWritableDatabase();
10. //      openHelper.getReadableDatabase();
11.         //关闭资源
12.         openHelper.close();
```

注意：

如果 openHelper.getWritableDatabase();或者 openHelper.getReadableDatabase();是第一次被调用，那么数据库文件才会被创建，否则只打开不创建。

创建的数据库文件如图 1-1 所示，包含两个文件，一个就是我们自定义的名称 myuser.db，另外一个 myuser.db-journal，该文件会被自动创建，是 sqlite 的一个临时的日志文件，主要用于 sqlite 数据库的事务回滚操作。

图 1-1　　database 文件

1.2　数据库的增删改查

在 1.1 节中创建了 MySQLiteOpenHelper 类，通过该类可以获取 SQLiteDatabase 对象。而 SQLiteDatabase 对象则可是实现对数据库的增删改查操作。

对数据库的增删改查我分为两种方式，见 1.2.1 和 1.2.2 两节。

1.2.1 使用纯 SQL 语句实现

一、添加数据

【文件 1-3】 添加数据代码

```
1.  /**
2.   * 纯 SQL 方式添加数据
3.   */
4.  public void add1(View view) {
5.      //通过 SQLiteOpenHelper 对象获取 SQLiteDatabase
6.      SQLiteDatabase database = mOpenHelper.getWritableDatabase();
7.      String sql = "insert into t_user (c_name,c_age,c_phone) values (?,?,?)";
8.      /*
9.       * 执行数据库，参数以 Object[]的形式传递
10.      */
11.     database.execSQL(sql, new Object[] { "zhangsan" + new Random().nextInt(100),
12. new Random().nextInt(30), "" + (5550 + new Random().nextInt(100)) });
13.     //关闭数据库释放资源
14.     database.close();
15.     Toast.makeText(this, "执行了", Toast.LENGTH_SHORT).show();
16. }
```

二、删除数据

【文件 1-4】 删除数据代码

```
1.      /**
2.       * 删除年龄小于 21 的用户
3.       */
4.      public void delete1(View view) {
5.          SQLiteDatabase database = mOpenHelper.getReadableDatabase();
6.          String sql = "delete from t_user where c_age<?";
7.          database.execSQL(sql, new Object[] { 21 });
8.          database.close();
9.          Toast.makeText(this, "执行了", Toast.LENGTH_SHORT).show();
10.     }
```

三、修改数据

【文件 1-5】 修改数据代码

```
1.      /**
2.       * 更改数据
3.       */
4.      public void update1(View view) {
5.          SQLiteDatabase database = mOpenHelper.getWritableDatabase();
6.          String sql = "update t_user set c_name=? where c_age=?";
```

```
        database.execSQL(sql, new Object[] { "lisi", 25 });
        database.close();
        Toast.makeText(this, "执行了", Toast.LENGTH_SHORT).show();
    }
```

四、查询数据

【文件 1-6】 查询数据代码

```
/**
 * 查询数据
 */
public void query1(View view) {
    //将查询到的数据封装到 User 集合中
    ArrayList<User> users = new ArrayList<User>();
    //获取 SQLiteDatabase
    SQLiteDatabase database = mOpenHelper.getReadableDatabase();
    String sql = "select uid,c_name,c_age,c_phone from t_user where c_age<?";
    /*
     * 执行查询语句，返回游标对象，可以将游标看做指向结果集的指针，
     */
    Cursor cursor = database.rawQuery(sql, new String[] { "130" });
    // 游标的默认位置位于第一行数据的前面
    while (cursor.moveToNext()) {
        User user = new User();
        //从第 1 列获取 int 型数据，字段的顺序是由查询语句决定的
        int uid = cursor.getInt(0);
        //从第 2 列获取字符串数据
        String name = cursor.getString(1);
        //从第 3 列获取 int 型数据
        int age = cursor.getInt(2);
        String phone = cursor.getString(3);
        user.setAge(age);
        user.setName(name);
        user.setPhone(phone);
        user.setUid(uid);
        users.add(user);
    }
    cursor.close();
    database.close();
    // 显示数据
    StringBuilder sb = new StringBuilder();
    for (User u : users) {
        sb.append(u.toString() + "\n");
    }
    tv_user.setText(sb.toString());
```

```
38. }
```

注意：

使用纯 SQL 语句操作适合 SQL 比较熟练的程序员。如果 SQL 掌握的不好，没关系，Android 提供了一套 API 可以帮助我们完成以上操作。

1.2.2 使用特有 API 实现

一、添加数据

【文件 1-7】 添加数据代码

```
1.      /**
2.       * 第二种方式添加数据
3.       */
4.      public void add2(View view) {
5.          //获取 SQLiteDatabase
6.          SQLiteDatabase database = mOpenHelper.getWritableDatabase();
7.          /*
8.           * 该类底层是 Map 数据结构
9.           * key 对应数据库表中字段
10.          * value 是想插入的值
11.          */
12.         ContentValues values = new ContentValues();
13.         values.put("c_name", "王五" + new Random().nextInt(10));
14.         values.put("c_age", new Random().nextInt(50));
15.         values.put("c_phone", "5556");
16.         /*
17.          * 第一个参数：表名，注意不是数据库名！
18.          * 第二个参数：如果 ContentValues 为空，那么默认情况下是不允许插入空值的，
19.          * 但是如果给该参数设置了一个指定的列名，那么就允许 ContentValues 为空，
20.          *同时给该列插入 null 值。
21.          * 第三个参数：要插入的数值，封装成了 ContentValues 对象\
22.          * 返回 long 类型的值，代表该条记录在数据库中的 id
23.          */
24.         long insert = database.insert(TABLE_NAME, null, values);
25.         //释放资源
26.         database.close();
27.         Toast.makeText(this, "插入成功==" + insert, Toast.LENGTH_SHORT).show();
28.     }
```

二、 删除数据

【文件 1-8】 删除数据代码

```
1.  /**
2.   * 第二种方式删除数据
3.   */
4.  public void delete2(View view) {
```

```
5.      SQLiteDatabase database = mOpenHelper.getReadableDatabase();
6.      /*
7.       * 第一个参数：表名
8.       * 第二个参数：删除条件，比如 c_age=?或者 c_age<?
9.       * 第三个参数：参数，用于替换?
10.      * 返回值：删除成功的个数
11.      */
12.     int delete = database.delete(TABLE_NAME, "c_age<?", new String[] { "25" });
13.     database.close();
14.     Toast.makeText(this, "删除了" + delete + "条数据", Toast.LENGTH_SHORT).show();
15. }
```

三、修改数据

【文件 1-9】 修改数据代码

```
1.  /**
2.   * 第二种方式修改数据
3.   */
4.  public void update2(View view) {
5.      SQLiteDatabase database = mOpenHelper.getWritableDatabase();
6.      ContentValues values = new ContentValues();
7.      values.put("c_name", "wangwu");
8.      // 将年龄等于 28 的姓名改为 wangwu
9.      int update = database.update(TABLE_NAME, values, "c_age=?",
10. new String[] { "28" });
11.     database.close();
12.     Toast.makeText(this, "修改了" + update + "条数据", Toast.LENGTH_SHORT).show();
13. }
```

四、查询数据

【文件 1-10】 查询数据代码

```
1. /**
2.   * 第二种方式查询数据
3.   */
4.  public void query2(View view) {
5.      ArrayList<User> users = new ArrayList<User>();
6.      SQLiteDatabase database = mOpenHelper.getReadableDatabase();
7.      /*
8.       * 参数 1：表名
9.       * 参数 2：要查询的字段
10.      * 参数 3：查询条件
11.      * 参数 4：条件中？对应的值
12.      * 参数 5：分组查询参数
13.      * 参数 6：分组查询条件
14.      * 参数 7：排序字段和规则
```

```
15.      */
16.     Cursor cursor = database.query(TABLE_NAME, new String[] {
17.     "uid", "c_name", "c_age", "c_phone" }, "c_age<?", new String[] { "130" },
18.     null, null, "c_age desc");
19.      //对游标 Cursor 的遍历
20.      while (cursor.moveToNext()) {
21.          User user = new User();
22.          int uid = cursor.getInt(0);
23.          String name = cursor.getString(1);
24.          int age = cursor.getInt(2);
25.          String phone = cursor.getString(3);
26.          user.setAge(age);
27.          user.setName(name);
28.          user.setPhone(phone);
29.          user.setUid(uid);
30.          users.add(user);
31.      }
32.      cursor.close();
33.      database.close();
34.      // 显示数据
35.      StringBuilder sb = new StringBuilder();
36.      for (User u : users) {
37.          sb.append(u.toString() + "\n");
38.      }
39.      tv_user.setText(sb.toString());
40. }
```

1.2.3 两种 SQLiteDatabase 的不同

SQLiteOpenHelper 有两个方法均可返回 SQLiteDatabase 对象：

一、getWritableDatabase()

该方法返回的对象和另外一个方法返回的对象没有任何差异，返回的对象对数据库都可以进行读、写操作，当磁盘已满或者权限不足的情况下该方法会抛出异常。

二、getReadableDatabase()

跟另外一个方法相比，在磁盘已满的情况下，该方法不会抛出异常，而是返回一个只读的数据库操作对象。

根据这两种方法返回对象的差异，如果需要对数据库进行查询操作则推荐使用后者，如果添加、修改、删除数据则推荐使用前者。

1.3 数据库的升级和事务操作

1.3.1 数据库的升级

在 1.1 节中创建的 MySQLiteOpenHelper 类中有如【文件 1-11】方法。该方法里执行了一条 sql 语句，该语句为数据库添加了一个 c_moeny 字段，那么该方法什么时候会被调用呢？

【文件 1-11】 代码片段

```
1.  /*
2.   * 当数据库升级是回调该方法
3.   */
4.  @Override
5.  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
6.      Log.d(TAG, "onUpgrade: oldVersion"+oldVersion+" newVersion="+newVersion);
7.      String sql = "alter table t_user add c_money float";
8.      db.execSQL(sql );
9.  }
```

在创建 MySQLiteOpenHelper 对象的时候需要传递一个 int 类型的 version 参数，代表数据库的版本号，数值从 1 开始。如果在 new MySQLiteOpenHelper 对象的时候传递的 version 大于先去创建的 version，那么就会导致系统回调 onUpgrade 方法，从而实现了数据库的升级。

1.3.2 事务的操作

在 SQLiteDatabase 提供了对事务的支持，处于事务中的操作都是“临时性”的，只有事务提交了才会将数据保存到数据库。

事务的使用不仅可以保证数据的一致性，也可以提高批处理时的执行效率。

SQLiteDatabase 提供的 beginTransaction（）打开事务，endTransaction（）结束事务。注意：结束事务并不代表事务提交，如果想让数据写入的数据库需要在结束事务前执行 setTransactionSuccessful（）方法。这是事务提交的唯一方式。

【文件 1-12】展示了当开启事务后，如果因为异常导致事务没有提提交，那么整个转账过程都不会成功。

【文件 1-12】 代码片段

```
1. /**
2.   * 数据库的事务操作
3.   */
4.  public void exchangeMoney(View v) {
5.      SQLiteDatabase database = mOpenHelper.getReadableDatabase();
6.      String sql = "update t_user set c_money=c_money-500 where c_name=?";
7.      // 模拟转账
8.      // 开启了事务，那么之后的操作都是在缓存中执行
9.      database.beginTransaction();
10.     database.execSQL(sql, new String[] { "wangwu" });
```

```
11.     // 在事务中模拟一个异常的发生
12.     int a = 1 / 0;
13.     String sql2 = "update t_user set c_money=c_money+500 where c_name=?";
14.     database.execSQL(sql2, new String[] { "lisi" });
15.     // 只有执行该代码，才将缓存中的数据写到数据库中
16.     database.setTransactionSuccessful();
17.     database.endTransaction();
18.     database.close();
19. }
```

【文件 1-13】展示了如果批处理数据时，使用事务可以显著提高效率。运行结果见图 1-2。

【文件 1-13】 代码片段

```
1. /**
2.   * 测试批处理效率
3.   */
4.  public void testBatch(View view) {
5.      SQLiteDatabase database = mOpenHelper.getWritableDatabase();
6.      /*
7.       * 普通方式添加 10000 条记录
8.       */
9.      String sql = "insert into t_user (c_name,c_age,c_phone) values (?,?,?)";
10.     long startTime = SystemClock.currentThreadTimeMillis();
11.     for (int i = 0; i < 1000; i++) {
12.         database.execSQL(sql, new Object[] { "zhangsan" + new Random().nextInt(100),
13.  new Random().nextInt(30), "" + (5550 + new Random().nextInt(100)) });
14.     }
15.     System.out.println("普通模式耗时:
16. "+(SystemClock.currentThreadTimeMillis()-startTime)+"毫秒");
17.     /*
18.      * 开启事务的方式添加 1000 条记录
19.      */
20.     startTime = SystemClock.currentThreadTimeMillis();
21.     database.beginTransaction();
22.     for(int i=0;i<1000;i++){
23.         database.execSQL(sql, new Object[] { "zhangsan" + new Random().nextInt(100),
24.  new Random().nextInt(30), "" + (5550 + new Random().nextInt(100)) });
25.     }
26.     database.setTransactionSuccessful();
27.     database.endTransaction();
28.     System.out.println("事务模式耗时:
29. "+(SystemClock.currentThreadTimeMillis()-startTime)+"毫秒");
30.     // 关闭数据库释放资源
31.     database.close();
32. }
```

Application	Tag	Text
com.itheima.android.sqlite	System.out	普通模式耗时：1509毫秒
com.itheima.android.sqlite	System.out	事务模式耗时：380毫秒

图 1-2　事务执行效率对比

分析：

通过如图 1-2 的测试结果我们发现使用事务大大提高了批量处理的效率。

1.4 sqlite3 工具的使用

sqlite3 是 Android 内置的操作数据库工具，使用该工具可以直接对 SQLite 数据库进行操作。

操作步骤（如图 1-3）：

1、在命令行界面使用 adb shell 命令进入 linux 内核

2、使用 cd 命令进入数据库所在目录（数据库的路径为”/data/data/应用包名/databases/数据库”）

3、使用”sqlite3 数据库名”进入数据库操作模式

图 1-3　sqlite3 使用截图

1.5　ListView 入门

ListView 是 Android 中最重要的控件之一，用于对数据进行列表展示。其特点如下：

1、屏幕上可以展示几个条目，ListView 就初始化几个，节省内存。

2、通过使用 convertView 对创建的视图对象进行复用，ListView 始终保持创建的对象个数为: 屏幕显示的条目的个数 ＋1。

3、ListView 自带 ScrollView 的功能，可以实现界面滚动。

4、ListView 控件的设计遵循 MVC 设计模式：

mode 数据模型(数据)：要被显示到 ListView 上的数据集合

view 视图(展示数据)： ListView

controller 控制层(把数据展示到空间上)：适配器 Adapter

1.5.1 ListView 的使用

1、在布局中的使用

【文件 1-14】 代码片段

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <ListView
8.         android:id="@+id/lv"
9.         android:layout_width="match_parent"
10.         android:layout_height="match_parent" >
11.     </ListView>
12.
13. </LinearLayout>
```

2、在代码中的使用

ListView 采用的是 MVC 的模式，需要配合 Controller 也就是 ListView 中的 Adapter 使用。

【文件 1-15】 代码片段

```
1. package com.itheima.android.listview;
2.
3. import java.util.ArrayList;
4. import android.os.Bundle;
5. import android.app.Activity;
6. import android.util.Log;
7. import android.view.View;
8. import android.view.ViewGroup;
9. import android.widget.BaseAdapter;
10. import android.widget.ListView;
11. import android.widget.TextView;
12.
13. /**
14.  *
15.  * @author wzy 2015-10-24
16.  *
17.  */
18. public class MainActivity extends Activity {
19.
20. public static final String TAG = "MainActivity";
21. private ListView listView;
22. private ArrayList<String> data;
23.
```

```
24. @Override
25. protected void onCreate(Bundle savedInstanceState) {
26.     super.onCreate(savedInstanceState);
27.     setContentView(R.layout.activity_main);
28.     // 初始化测试数据（Model）
29.     data = new ArrayList<String>();
30.     for (int i = 0; i < 10000; i++) {
31.         data.add("我是数据" + i);
32.     }
33.     // 获取 ListView 控件（View）
34.     listView = (ListView) findViewById(R.id.lv);
35.     // 创建适配器（Controller）
36.     MyAdapter adapter = new MyAdapter();
37.     // 给 ListView 设置适配器
38.     listView.setAdapter(adapter);
39.
40. }
41.
42. /**
43.  *
44.  * @author wzy 2015-10-24 自定义类继承 BaseAdapter
45.  */
46. private class MyAdapter extends BaseAdapter {
47.     /*
48.      * 必选方法，只有该方法返回值大于 0，才会调用 getView 方法
49.      */
50.     @Override
51.     public int getCount() {
52.         Log.d(TAG, "getCount");
53.         return data.size();
54.     }
55.     /*
56.      * 可选方法
57.      */
58.     @Override
59.     public Object getItem(int position) {
60.         return data.get(position);
61.     }
62.     /*
63.      * 可选方法
64.      */
65.     @Override
66.     public long getItemId(int position) {
67.         return position;
```

```
68.     }
69.     /*
70.      * 必选方法，ListView 的一个每一个条目的显示都是通过该方法返回的
71.      */
72.     @Override
73.     public View getView(int position, View convertView, ViewGroup parent) {
74.         Log.d(TAG, position + "" + convertView);
75.         //创建一个简单的 TextView 对象
76.         TextView textView = new TextView(MainActivity.this);
77.         //设置文本内容
78.         textView.setText((String) getItem(position));
79.         return textView;
80.     }
81. }
82.
83. }
84.
```

上面代码运行结果如图 1-4 所示。

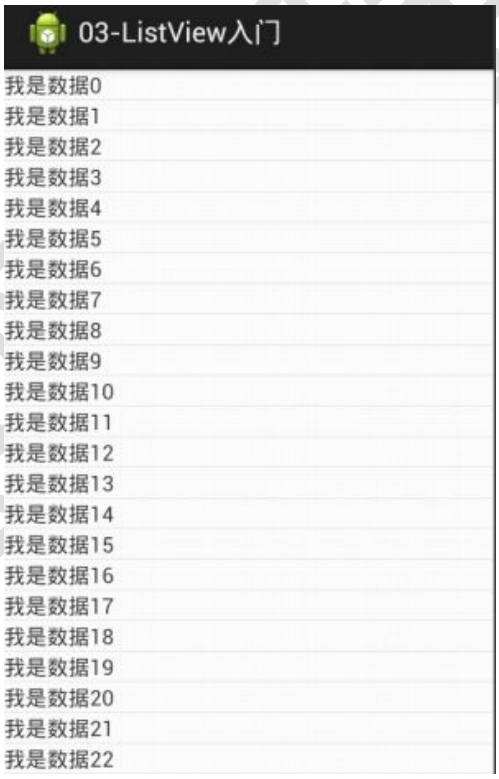

图 1-4　ListView 的使用

1.5.2 ListView 的优化

在 1.5.1 节中创建的 ListView 如果我们快速的不断往上滑动，那么很可能遇到 OOM 异常，如图 1-5。

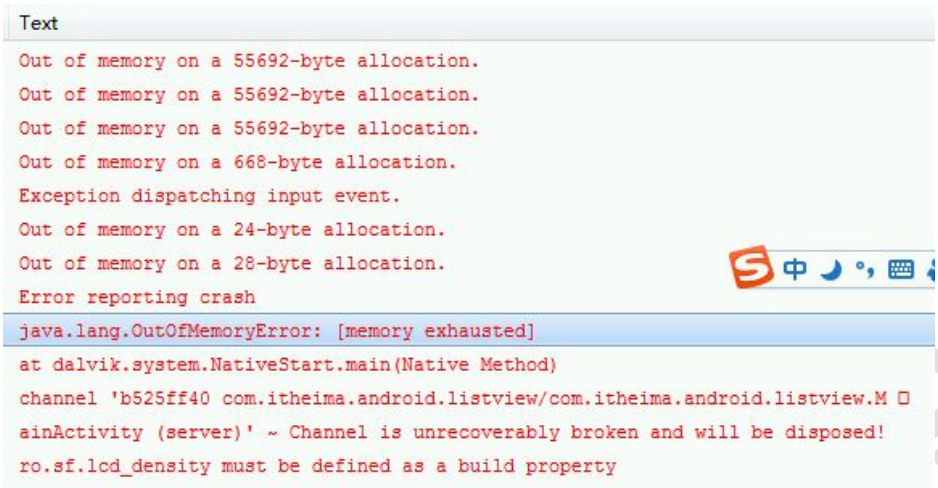

图 1-5　　ListView 报 OOM 异常

对 ListView 进行优化的最简单有效的措施就是复用 getView 方法中的 convertView 对象。convertView 来自缓存池，缓存池由 ListView 维护，缓存池中的数据来自 getView 方法的返回值，也就是说 getView 方法的返回值用完后并没有“浪费”，而是被系统放到 ListView 的缓存池里了。

缓存池的大小=屏幕显示的条目数+1，当滑动屏幕时，被隐藏的项会作为缓存对象，作为 getView 的参数传递进来。只需修改此缓存对象的数据，就可以直接使用，而不需要再重新 new 一个新的对象，节省了内存，防止内存溢出。

【文件 1-16】　　优化后的代码片段

```
1.      /*
2.       * 必选方法，ListView 的一个每一个条目的显示都是通过该方法返回的
3.       */
4.      @Override
5.      public View getView(int position, View convertView, ViewGroup parent) {
6.          Log.d(TAG, position + "" + convertView);
7.          TextView textView = null;
8.          if (convertView==null) {
9.              //如果缓存中没有数据则需要创建一个新的 TextView
10.             textView = new TextView(MainActivity.this);
11.         }else {
12.             //如果缓存中有数据则直接强转即可
13.             textView = (TextView) convertView;
14.         }
15.         //创建一个简单的 TextView 对象
16.         //设置文本内容
17.         textView.setText((String) getItem(position));
18.         return textView;
19.     }
```

1.6 案例-老虎机

需求：如图 1-6 所示。界面被水平划分为 3 份，用于显示 3 个 ListView。每个 ListView 的条目都依次

显示红黄绿三种颜色。在界面的底部有 Button，单击后 3 个 ListView 开始自动滚动到一个随机位置。如果随机位置的颜色相同则中奖。中奖结果在界面中奖的 TextView 中显示。

图 1-6　老虎机效果

1.6.1 编写布局

【文件 1-17】　activity_main.xml

```
1.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:layout_width="match_parent"
3.     android:layout_height="match_parent" >
4.
5.     <LinearLayout
6.        android:layout_width="match_parent"
7.        android:layout_height="match_parent"
8.        android:orientation="horizontal">
9.
10.        <ListView
11.           android:id="@+id/lv1"
12.           android:layout_width="0dp"
13.           android:layout_height="match_parent"
14.           android:layout_weight="1" >
15.        </ListView>
16.
```

```
17.         <ListView
18.             android:id="@+id/lv2"
19.             android:layout_width="0dp"
20.             android:layout_height="match_parent"
21.             android:layout_weight="1" >
22.         </ListView>
23.
24.         <ListView
25.             android:id="@+id/lv3"
26.             android:layout_width="0dp"
27.             android:layout_height="match_parent"
28.             android:layout_weight="1" >
29.         </ListView>
30.     </LinearLayout>
31.
32.     <TextView
33.         android:id="@+id/tv_result"
34.         android:layout_width="match_parent"
35.         android:layout_height="30dp"
36.         android:layout_centerVertical="true"
37.         android:background="#55AA0000"
38.         android:gravity="center"
39.         android:text="摇呀摇..." />
40.
41.     <Button
42.         android:layout_width="match_parent"
43.         android:layout_height="wrap_content"
44.         android:layout_alignParentBottom="true"
45.         android:onClick="start"
46.         android:text="开始抽奖" />
47.
48. </RelativeLayout>
```

1.6.2 编写代码

【文件 1-18】 MainActivity.java

```
1. package com.itheima.android.tiger;
2.
3. import java.util.Random;
4. import android.os.Bundle;
5. import android.app.Activity;
6. import android.graphics.Color;
7. import android.util.Log;
```

```
8. import android.view.View;
9. import android.view.ViewGroup;
10. import android.widget.AbsListView;
11. import android.widget.AbsListView.OnScrollListener;
12. import android.widget.BaseAdapter;
13. import android.widget.ListView;
14. import android.widget.TextView;
15. /**
16.  *
17.  * @author wzy 2015-10-24
18.  * 老虎机小游戏
19.  */
20. public class MainActivity extends Activity implements OnScrollListener {
21. //声明适配器
22. private MyAdapter myAdapter;
23. //三个 ListView 对象
24. private ListView lv1;
25. private ListView lv2;
26. private ListView lv3;
27. //3 个初始随机数，作为 ListView 初始时开始条目的位置
28. private int random1;
29. private int random2;
30. private int random3;
31. //3 个 ListView 的滚动状态
32. private int currentState1 = -1;
33. private int currentState2 = -1;
34. private int currentState3 = -1;
35. //用于显示结果
36. private TextView tv_result;
37. // 是否中奖
38. private boolean isBingo = false;
39. //抽奖后的新的 3 个随机数
40. private int rad1;
41. private int rad2;
42. private int rad3;
43.
44. @Override
45. protected void onCreate(Bundle savedInstanceState) {
46.     super.onCreate(savedInstanceState);
47.     setContentView(R.layout.activity_main);
48.     //获取 ListView
49.     lv1 = (ListView) findViewById(R.id.lv1);
50.     lv2 = (ListView) findViewById(R.id.lv2);
51.     lv3 = (ListView) findViewById(R.id.lv3);
52.     //获取 TextView
```

```
53.     tv_result = (TextView) findViewById(R.id.tv_result);
54.     //初始化 Adapter
55.     myAdapter = new MyAdapter();
56.     //设置适配器
57.     lv1.setAdapter(myAdapter);
58.     lv2.setAdapter(myAdapter);
59.     lv3.setAdapter(myAdapter);
60.     //生成随机数 范围为【0-int 最大值/2】
61.     random1 = new Random().nextInt(Integer.MAX_VALUE / 2);
62.     random2 = new Random().nextInt(Integer.MAX_VALUE / 2);
63.     random3 = new Random().nextInt(Integer.MAX_VALUE / 2);
64.     //将 ListView 第一个条目滚动到随机数位置
65.     lv1.setSelection(random1);
66.     lv2.setSelection(random2);
67.     lv3.setSelection(random3);
68.     //给 ListView 设置滚动监听
69.     lv1.setOnScrollListener(this);
70.     lv2.setOnScrollListener(this);
71.     lv3.setOnScrollListener(this);
72.
73. }
74.
75. /**
76.  * 开始游戏
77.  *
78.  * @param view
79.  */
80. public void start(View view) {
81.     /*
82.      * 生成新的随机数，该随机数介于死一个随机数±50 的范围，
83.      * 这样滚动时间不至于太久
84.      */
85.     rad1 = random1 + new Random().nextInt(100) - 50;
86.     rad2 = random2 + new Random().nextInt(100) - 50;
87.     rad3 = random3 + new Random().nextInt(100) - 50;
88.     // 计算是否中奖
89.     if (rad1 % 3 == rad2 % 3 && rad2 % 3 == rad3 % 3) {
90.         isBingo = true;
91.     } else {
92.         isBingo = false;
93.     }
94.     tv_result.setText("抽奖中。。。");
95.     // 让 ListView 平滑的滚动起来
96.     lv1.smoothScrollToPosition(rad1);
```

```
97.     lv2.smoothScrollToPosition(rad2);
98.     lv3.smoothScrollToPosition(rad3);
99.
100.    }
101.
102.    private class MyAdapter extends BaseAdapter {
103.
104.        @Override
105.        public int getCount() {
106.            return Integer.MAX_VALUE;
107.        }
108.
109.        @Override
110.        public Object getItem(int position) {
111.            return null;
112.        }
113.
114.        @Override
115.        public long getItemId(int position) {
116.            return position;
117.        }
118.
119.        @Override
120.        public View getView(int position, View convertView, ViewGroup parent) {
121.            if (convertView == null) {
122.                convertView = new TextView(MainActivity.this);
123.            }
124.            TextView textView = (TextView) convertView;
125.            /*
126.             * 模拟数据
127.             * 根据条目位置%3 的值分配对应的文字和背景
128.             */
129.            switch (position % 3) {
130.            case 0:
131.                textView.setText("香蕉");
132.                textView.setTextColor(Color.YELLOW);
133.                break;
134.            case 1:
135.                textView.setText("苹果");
136.                textView.setTextColor(Color.GREEN);
137.                break;
138.            case 2:
139.                textView.setText("草莓");
140.                textView.setTextColor(Color.RED);
141.                break;
```

```
142.            default:
143.                break;
144.            }
145.            textView.setTextSize(24);
146.            return convertView;
147.        }
148.
149.    }
150.    /**
151.     * OnScrollListener 接口中的方法，
152.     * 当 ListView 滚动状态改变时会回调该方法，
153.     * 滚动状态有三种：0 静止，1 手指滑动，2 惯性滚动
154.     * 参数 1：ListView 对象
155.     * 参数 2：滚动状态
156.     */
157.    @Override
158.    public void onScrollStateChanged(AbsListView view, int scrollState) {
159.        //根据 ListView 的 id 判断是哪个 ListView 回调的该方法
160.        switch (view.getId()) {
161.        case R.id.lv1:
162.            //记录 ListView 的滚动状态
163.            currentState1 = scrollState;
164.            break;
165.        case R.id.lv2:
166.            currentState2 = scrollState;
167.            break;
168.        case R.id.lv3:
169.            currentState3 = scrollState;
170.            break;
171.        default:
172.            break;
173.        }
174.        /*
175.         * 如果状态同时为 0，则代表滚动结束
176.         * 公布结果
177.         */
178.        if (currentState1 == 0 && currentState2 == 0 && currentState3 == 0) {
179.            /*
180.             * 因为 smoothScrollToPosition 方法有 bug 导致条目停止的位置有偏差
181.             * 因此需要调用 setSelection 将位置精准确定
182.             */
183.            lv1.setSelection(rad1);
184.            lv2.setSelection(rad2);
185.            lv3.setSelection(rad3);
```

```
            if (isBingo) {
                tv_result.setText("恭喜您，中奖啦...");
            } else {
                tv_result.setText("这次没中奖，下次中奖概率更高哦...");
            }
        }
    }
    /**
     * 每滚动一个条目，该方法被回调一次。
     */
    @Override
    public void onScroll(AbsListView view, int firstVisibleItem,
int visibleItemCount, int totalItemCount) {
        Log.d("Scroll", "onScroll firstVisibleItem=" + firstVisibleItem
+ " visibleItemCount=" + visibleItemCount + " totalItemCount" + totalItemCount);
    }

}
```

1.6.3 功能总结

一、让 ListView 定位到某一位置

【文件 1-19】 代码片段

```
lv1.setSelection(random1);
```

该方法可以 ListView 定位到让指定位置（random1），就是让指定位置的条目位于当前界面第一行。非常准确。

二、让 ListView 平滑滚动到某一位置

【文件 1-20】 代码片段

```
lv1.smoothScrollToPosition(rad1);
```

该方法可以让 ListView 平滑的滚动到指定位置（rad1），但是非常不准确。

三、监听 ListView 的滚动状态

【文件 1-21】 给 ListView 设置监听器

```
lv1.setOnScrollListener(OnScrollListener);
```

【文件 1-22】 覆写 onScrollStateChanged 和 onScroll 方法

```
@Override
public void onScrollStateChanged(AbsListView view, int scrollState) {
@Override
public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount,
 int totalItemCount)
```

1.7 复杂条目的 ListView

如图 1-7 所示，ListView 的每个条目不是单独的一个 View 而是一个组合 View。跟 1.5 节中 ListView

的适配器相比，本节中适配器需要在其 getView 方法中将一个复杂布局填充成为一个 View 对象，这也是本节的重点所在。

图 1-7　　复杂条目的 ListView

1.7.1 案例-新闻客户端

通过模拟新闻客户端来学习使用复杂条目的 ListView。

一、编写布局

MainActivity 的布局非常简单，不再给出，只给出作为 ListView 的条目的布局，见【文件 1-23】。

【文件 1-23】　　list_item.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content" >
5.
6.     <ImageView
7.         android:id="@+id/iv"
8.         android:contentDescription="@null"
9.         android:layout_width="100dp"
10.        android:layout_height="70dp"
```

```
11.         android:src="@drawable/ic_launcher"
12.         />
13.         <TextView
14.             android:id="@+id/tv_title"
15.             android:layout_marginLeft="5dp"
16.             android:layout_toRightOf="@id/iv"
17.             android:layout_width="wrap_content"
18.             android:layout_height="wrap_content"
19.             android:text="这是新闻的标题"
20.             android:singleLine="true"
21.             android:ellipsize="end"
22.             />
23.         <TextView
24.             android:id="@+id/tv_desc"
25.             android:layout_marginLeft="5dp"
26.             android:layout_below="@id/tv_title"
27.             android:layout_toRightOf="@id/iv"
28.             android:layout_width="wrap_content"
29.             android:layout_height="wrap_content"
30.             android:text="这是新闻的描述"
31.             android:textSize="12sp"
32.             android:textColor="#999999"
33.             android:maxLines="2"
34.             android:ellipsize="end"
35.             />
36.         <TextView
37.             android:id="@+id/tv_type"
38.             android:layout_marginRight="5dp"
39.             android:layout_alignParentRight="true"
40.             android:layout_alignBottom="@id/iv"
41.             android:layout_width="wrap_content"
42.             android:layout_height="wrap_content"
43.             android:text="专题"
44.             android:textColor="#Ff0000"
45.             android:textSize="12sp"
46.             />
47. </RelativeLayout>
48.
```

技能：

上面布局中使用到了 ImageView 控件，该控件用于显示图片，其 android:src="@drawable/ic_launcher" 属性指定了要显示的图片内容。

二、编写代码

【文件 1-24】 MainActivity.java

```
1. package com.itheima.android.superlistview;
```

```
2.
3. import android.os.Bundle;
4. import android.app.Activity;
5. import android.view.View;
6. import android.view.ViewGroup;
7. import android.widget.BaseAdapter;
8. import android.widget.ImageView;
9. import android.widget.ListView;
10. import android.widget.TextView;
11.
12. /**
13.  *
14.  * @author wzy 2015-10-24 复杂条目的 ListView
15.  */
16. public class MainActivity extends Activity {
17.
18. private ListView listView;
19.
20. @Override
21. protected void onCreate(Bundle savedInstanceState) {
22.     super.onCreate(savedInstanceState);
23.     setContentView(R.layout.activity_main);
24.     listView = (ListView) findViewById(R.id.lv);
25.     // 设置 Adapter
26.     listView.setAdapter(new MyAdapter());
27. }
28.
29. private class MyAdapter extends BaseAdapter {
30.
31.     @Override
32.     public int getCount() {
33.         // 模拟 1000 条数据
34.         return 1000;
35.     }
36.
37.     @Override
38.     public Object getItem(int position) {
39.         return null;
40.     }
41.
42.     @Override
43.     public long getItemId(int position) {
44.         return 0;
45.     }
```

```
46.
47.     @Override
48.     public View getView(int position, View convertView, ViewGroup parent) {
49.         ViewHolder holder;
50.         if (convertView == null) {
51.             /*
52.              * 使用 View 的 inflate 静态方法将一个 xml 布局文件填充为 View 对象
53.              * 参数 1：上下文
54.              * 参数 2：xml 布局文件的 id
55.              * 参数 3：ViewGroup，如果不为 null 那么就将该布局挂载到 ViewGroup 树中
56.              */
57.             convertView = View.inflate(MainActivity.this, R.layout.list_item, null);
58.             //每新创建一个条目，就创建一个 ViewHolder 类
59.             holder = new ViewHolder();
60.             /*
61.              * 将 convertView 中子控件对象赋值给 ViewHolder 对应的成员变量，
62.              * 这样就相当于将 convertView 的所有子控件都封装成了一个 ViewHolder 对象，
63.              * 这样当需要 convertView 的子控件的时候就不需要再调用 findViewById 方法了，
64.              * 因此该方法内部是递归操作，性能比较低
65.              */
66.             holder.iv = (ImageView) convertView.findViewById(R.id.iv);
67.             holder.tv_title = (TextView) convertView.findViewById(R.id.tv_title);
68.             holder.tv_desc = (TextView) convertView.findViewById(R.id.tv_desc);
69.             holder.tv_type = (TextView) convertView.findViewById(R.id.tv_type);
70.             /*
71.              * 将封装好的 holder 对象作为 convertView 的属性，
72.              * 只有这样拿到了 convertView，就能拿到 holder，
73.              * 拿到了 holder 就拿到了里面的子控件，
74.              * 从而避免了过度执行 findViewById 方法
75.              *
76.              */
77.             convertView.setTag(holder);
78.         }
79.         //从 convertView 中获取 ViewHolder 对象
80.         holder = (ViewHolder) convertView.getTag();
81.         //给 holder 中的子控件赋值
82.         switch (position % 4) {
83.         case 0:
84.             holder.iv.setImageResource(R.drawable.image0);
85.             holder.tv_type.setText("专题");
86.             break;
87.         case 1:
88.             holder.iv.setImageResource(R.drawable.image1);
89.             holder.tv_type.setText("活动");
90.             break;
```

```
91.         case 2:
92.             holder.iv.setImageResource(R.drawable.image2);
93.             holder.tv_type.setText("跟帖");
94.             break;
95.         case 3:
96.             holder.iv.setImageResource(R.drawable.image3);
97.             holder.tv_type.setText("推广");
98.             break;
99.         default:
100.                break;
101.            }
102.
103.            holder.tv_title.setText("这是标题" + position);
104.            holder.tv_desc.setText("这是新闻的描述信息" + position);
105.
106.            return convertView;
107.        }
108.
109.    }
110.    /*
111.     * 定义一个静态类，该类的成员变量跟 ListView 条目中用到的控件类型一致，
112.     * 目的是为了用该类将 ListView 中的控件封装起来，
113.     * 然后将该类跟 ListView 的条目绑定起来，
114.     * 目的是为了减少 findViewById 方法被调用次数
115.     */
116.    static class ViewHolder {
117.        ImageView iv;
118.        TextView tv_title;
119.        TextView tv_desc;
120.        TextView tv_type;
121.    }
122. }
123.
```

技能：

上面代码使用 ViewHolder 对 ListView 作了进一步的优化，其优化的核心思想是：findViewById 方法是一个内部递归遍历的方法，查找 view 比较耗时，因此我们应该尽量避免过多的使用该方法。

当第一次执行 findViewById 获取到子控件的时候，将这些子控件封装到 ViewHolder 中，然后再将 ViewHolder 作为属性设置给 convertView，这样当 convertView 需要子控件的时候就不用执行 findViewById 了，而是直接找 ViewHolder 类就可以了。

1.7.2 Inflater 的常见实现方法

将 xml 布局文件填充为 View 对象总共有三种方法：

【文件 1-25】 Inflater（俗称：打气筒、填充器）的三种实现方法

```
1.  /*
2.  * 第一种方法，
3.  * 底层使用的其实就是第二种方法
4.  */
5.   convertView = View.inflate(MainActivity.this, R.layout.list_item, null);
6.  /*
7.   * 第二种方法，
8.   * 先通过 LayoutInfalter 的静态方法 from 获取 LayoutInflater 对象
9.   * 然后调用 inflate 方法
10.  */
11. LayoutInflater layoutInflater1 = LayoutInflater.from(MainActivity.this);
12. convertView = layoutInflater1.inflate(R.layout.list_item, null);
13. /*
14.  * 第三种方法，
15.  * 先通过上下文提供的 getSystemService 方法获取 LayoutInfater 对象
16.  *  然后调用 inflate 方法
17.  */
18. LayoutInflater layoutInflater2 =
19.  (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
20. convertView = layoutInflater2.inflate(R.layout.list_item, null);
```

注意：

getSystemService(String) 方法是 Activity 类提供的，根据该方法传入的不同参数可以获取不同的系统服务对象。

1.8 ListView 的常见适配器

除了经常使用到的 BaseAdapter 外，系统还提供了几个已经实现好的适配器，如图 1-8 所示。

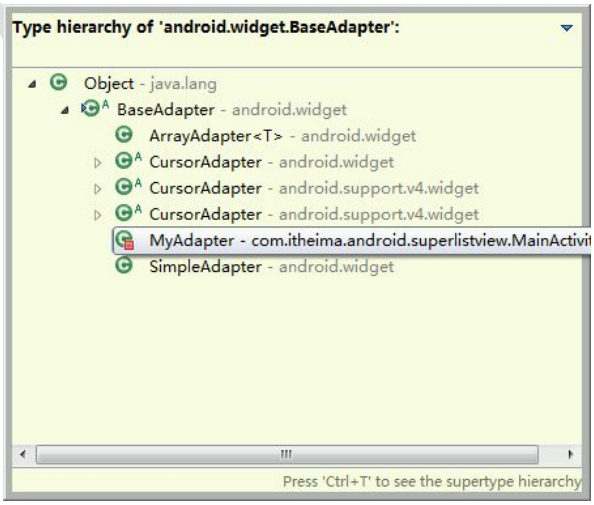

图 1-8　常见适配器

1.8.1 ArrayAdapter

【文件 1-26】　ArrayAdapter 的用法

```
1.  ListView listView = (ListView) findViewById(R.id.listview);
2.      String[] datas = new String[]{"zhangsan","lisi","wangwu","zhaoliu"};
3.      /**
4.       * 第一个参数是：上下文
5.       * 第二个参数是：布局文件的 id，这里使用 Android 系统提供的简单布局
6.       * 第三个参数是：要显示的数据，数组或者 List 集合都行
7.       */
8.      ArrayAdapter<String> arrayAdapter =
9. new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, datas );
10.     listView.setAdapter(arrayAdapter);
```

运行结果如图 1-9 所示：

图 1-9　ArrayAdapter 效果图

观测 ArrayAdapter 源码，查看 getView 方法，见【文件 1-27】，发现其已经通过覆写 convertView 实现了对 ListView 的优化，

【文件 1-27】　ArrayAdapter getView 方法源码

```
1.   public View getView(int position, View convertView, ViewGroup parent) {
2.       return createViewFromResource(position, convertView, parent, mResource);
3.    }
4.    private View createViewFromResource(int position, View convertView,
5. ViewGroup parent,
6.           int resource) {
7.       View view;
8.       TextView text;
```

```
9.
10.         if (convertView == null) {
11.             view = mInflater.inflate(resource, parent, false);
12.         } else {
13.             view = convertView;
14.         }
15.         try {
16.             if (mFieldId == 0) {
17.               //If no custom field is assigned, assume the whole resource is a TextView
18.                 text = (TextView) view;
19.             } else {
20.                 //  Otherwise, find the TextView field within the layout
21.                 text = (TextView) view.findViewById(mFieldId);
22.             }
23.         } catch (ClassCastException e) {
24.             Log.e("ArrayAdapter", "You must supply a resource ID for a TextView");
25.             throw new IllegalStateException(
26.                     "ArrayAdapter requires the resource ID to be a TextView", e);
27.         }
28.         T item = getItem(position);
29.         if (item instanceof CharSequence) {
30.             text.setText((CharSequence)item);
31.         } else {
32.             text.setText(item.toString());
33.         }
34.         return view;
35.     }
```

1.8.2 SimpleAdapter

SimpleAdapter 可以实现比 ArrayAdapter 复杂一点的布局。使用 SimpleAdapter 的数据一般都是 HashMap 构成的 List，List 的每一个对象对应 ListView 的每一行。HashMap 的每个键值数据映射到布局文件中对应 id 的组件上。因为系统没有对应的布局文件可用，我们可以自己定义一个布局文件。

在本节中我们用 TextView 和 ImageView 组合作为布局以演示 SimpleAdapter 的用法。ListView 的条目布局如图 1-10。

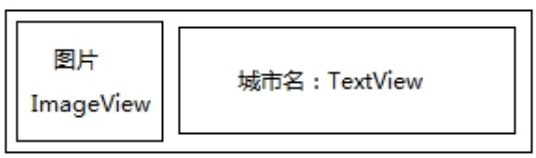

图 1-10　ListView 条目布局

【文件 1-28】　list_item.xml

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

```
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="wrap_content"
5.     android:orientation="horizontal"
6.     >
7.
8.     <ImageView
9.        android:id="@+id/iv_icon"
10.        android:layout_width="wrap_content"
11.        android:layout_height="wrap_content"
12.        android:src="@drawable/ic_launcher" />
13.
14.     <TextView
15.        android:id="@+id/tv"
16.        android:textSize="26sp"
17.        android:layout_width="wrap_content"
18.        android:layout_height="wrap_content"
19.        android:layout_gravity="center_vertical"
20.        android:text="城市" />
21.
22. </LinearLayout>
```

【文件 1-29】 SimpleAdapter 代码

```
1.     ListView listView = (ListView) findViewById(R.id.lv);
2.      //创建 List 集合
3.      List<Map<String, Object>> data = new ArrayList<Map<String,Object>>();
4.      String[] from = new String[]{"icon","tv"};
5.      int[] to = new int[]{R.id.iv_icon,R.id.tv};
6.      //往 map1 集合中添加数据
7.      HashMap<String, Object> map1 = new HashMap<String, Object>();
8.      map1.put(from[0],R.drawable.beijing);
9.      map1.put(from[1], "北京");
10.     //将 map1 添加到集合
11.     data.add(map1);
12.     //往 map2 集合中添加数据
13.     HashMap<String, Object> map2 = new HashMap<String, Object>();
14.     map2.put(from[0], R.drawable.shanghai);
15.     map2.put(from[1], "上海");
16.     //将 map2 添加到集合
17.     data.add(map2);
18.
19.     /**
20.      * 第一个参数：上下文
21.      * 第二个参数：
```

```
22.      * 第三个参数：布局文件的资源 id
23.      * 第四个参数：Map 集合中 key 的数组
24.      * 第五个参数：list_item 布局中 TextView 的控件的 id
25.      */
26.     SimpleAdapter simpleAdapter = new SimpleAdapter(this, data,
27.     R.layout.list_item, from, to);
28.     listView.setAdapter(simpleAdapter);
```

运行结果如图 1-11。

图 1-11　SimpleAdapter 效果图