对于鬼畜机 APP 中“作品”在 SQLite 中的存储方式及结构的讨论

张寅森（千里冰封）

首先，“鬼畜机”是我个人开发的一款 APP，运行在 Android 系统上，向上兼容到安卓 6.0，向下兼容到安卓 4.0.3。原本目的只是为了练习处理媒体数据，后来才发现里面可以引发一定程度的讨论。之所以没有兼容到安卓 2.3.7 是因为在制作过程中缺少 SDK 平台。该 APP 内置两种 UI 风格的主界面，可以在“设置”界面中找到一个 Switch，拨动该 Switch 可以换风格。你的选择将会保存在 SharedPreference 中，属于持久类数据。如图 1 是 APP 的主界面。该风格的 UI 是我自己设计的，另一种风格的 UI 是朋友设计的。在此不列出。

图 1

图 2

该 APP 的创意是来自于国内知名弹幕视频网站哔哩哔哩上存在的一种被称为鬼畜的视频文化。该类视频使用各种素材经过精良的剪辑，配合上背景音乐，很容易让观众观后脑中产生无限循环，尤其是很有节奏感的剪辑作品。而经常被用作“鬼畜”的素材中出现的人物们则被称为“全明星”。早期人物有《帝国的毁灭》中的希特勒、葛平、金坷垃广告中的美国、非洲、日本的三人组、麦当劳广告中的麦当劳叔叔“蓝蓝路”等。在 2014 年后的后起之秀很多，在此不一一列举。由于他们在视频中的行为、言行十分有趣，甚至会让人狂笑，因此常常被鬼畜。鬼畜兴起于日本，传入中国。

该 APP 用途很简单，我在 APP 里面内置了很多由“全明星”的素材中采集的音频，每个音频对应一张图片。在最主要的“录音模式”界面中有数张图片，点击图片就会播放对应的音频。如果按照一定节奏、一定规律按钮的话就会产生类似“弹奏鬼畜”的效果。如图 2 是界面。

在该界面最下方有几行蓝色的文字，点击后会播放对应的音乐。该种音乐一般为 3 分钟的长音乐，不同于鬼畜音源，一般 0.1 秒到 2 秒。

图 3

在中间的“选项”按钮点击之后会弹出菜单，该菜单为录音的选项菜单，如图 3。

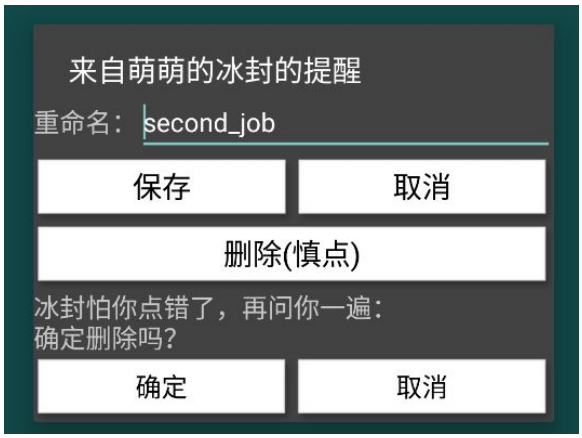

图 5

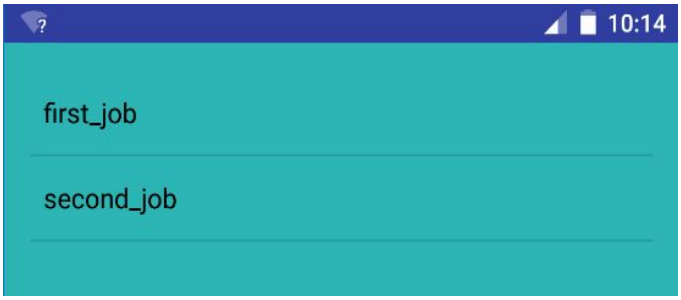

图 4

“录制”按钮用于录音，为最关键最核心的一个按钮。点击之后开始记录你按下图片的行为。相应算法在后文讨论。该按钮在点击之后会变成“停止”，点击“停止”会停止记录你的行为。

“播放”按钮用于跳转到播放界面。该按钮功能与图 1 中“我的作品”按钮作用相同，转到一个界面，该界面罗列了用户的录音作品，如图 4。在录音后会随机生成一个字符串作为作品名。

长按弹出菜单，执行的功能一目了然，因此不再赘述，如图 5。其中最下方 5 排的内容只有在点击了“删除”按钮才会出现，目的是防止用户误操作。用户可以将其命名成自己认识的名字。

短按播放该作品。

其他模式说明：

试音模式就是没有录音功能的录音模式，用于熟悉音源。个人设置目前只有 UI 调整。

挑战模式暂时未开放，在计划中该模式是一个类似 lovelive 手游的游戏模式。

关于作者中有一些与 APP 开发者有关的个人信息、鸣谢、推广等不大重要的信息。

在该 APP 的早期版本中，存储数据使用 SharedPreference。原因很简单，该数据库很方便，采用键值对的结构存储数据，目前的 UI 调整仍然在使用该数据库。不过由于该数据库运行效率较低，所以在后来的版本中被弃用。如图 6 是关于谷歌在 SDK 中提供的 SharedPreference 的官方文档。

The `SharedPreferences` class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use `SharedPreferences` to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

To get a `SharedPreferences` object for your application, use one of two methods:

- `getSharedPreferences()` - Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
- `getPreferences()` - Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.

> **User Preferences**
>
> Shared preferences are not strictly for saving "user preferences," such as what ringtone a user has chosen. If you're interested in creating user preferences for your application, see `PreferenceActivity`, which provides an Activity framework for you to create user preferences, which will be automatically persisted (using shared preferences).

To write values:

1. Call `edit()` to get a `SharedPreferences.Editor`.
2. Add values with methods such as `putBoolean()` and `putString()`.
3. Commit the new values with `commit()`

To read values, use `SharedPreferences` methods such as `getBoolean()` and `getString()`.

图 6

如图 7 是官方的样例代码。

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

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

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

图 7

可以看出，键值对的 API 非常方便。

不过为了使用户获得良好的体验，我在一次更新中将数据库重新封装了一遍，核心是 SQLite。官方相关说明和示例代码如图 8。由于后文将详细说明 SQLite，所以此处不再过多引用官方文档。

Android provides full support for SQLite databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.

The recommended method to create a new SQLite database is to create a subclass of `SQLiteOpenHelper` and override the `onCreate()` method, in which you can execute a SQLite command to create tables in the database. For example:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

图 8

首先是一个 SQLiteOpenHelper 的类的封装，即图 8 中示例代码的封装。
注：大量的 import 由于篇幅原因已经被删减。

```
package util;

public class DatabaseOpenHelper extends SQLiteOpenHelper implements Closeable{
    public static final String DATABASE_NAME = "ice1000.db" ;
    public static final String GROUP = "names" ;
    public static final String TABLE_NAME = "ice1000" ;
    public static final int DATABASE_VERSION = 1 ;
    public DatabaseOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE IF NOT EXISTS " + GROUP + "(name TEXT PRIMARY KEY);");
    }
    public void createTable(SQLiteDatabase db,String name){
        db.execSQL("CREATE TABLE IF NOT EXISTS " + name +
                "(name TEXT, id INTEGER, time INTEGER, cnt INTEGER);");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("ALTER TABLE ice1000 ADD COLUMN other TEXT");
```

```
            db.execSQL("ALTER TABLE names ADD COLUMN other TEXT");
        }
        @Override
        public synchronized void close() {
            super.close();
        }
}
```

代码很简单，仅有短短 46 行，在此不再赘述。
为了方便操作，所有涉及数据库操作的内部实现全部封装到另一个类中。
注：大量的 import 由于篇幅原因已经被删减。

```
package util;
/**
 * 封装 SQLite 操作。
 * Created by Administrator on 2015/12/14 0014.
 */
public class DatabaseManager implements Closeable{
    private DatabaseOpenHelper helper;
    private SQLiteDatabase database;
    public DatabaseManager(Context context) {
        onCreate(context);
    }
    private void onCreate(Context context){
        helper = new DatabaseOpenHelper(context);
//          可读可写
        database = helper.getWritableDatabase();
    }
//    增加一组声音
    public void addSounds(String groupName, List<OneSound> sounds) {
        database.beginTransaction();
        try{
            for (OneSound sound:sounds) {
                ContentValues contentValues = new ContentValues();
//                  contentValues.put("name", sound.name);
                contentValues.put("id", sound.id);
                contentValues.put("time", sound.time);
                contentValues.put("cnt", sound.cnt);
                database.insert(groupName, null, contentValues);
            }
            database.setTransactionSuccessful();
        }
        finally {
            database.endTransaction();
```

```
        }
    }
    public void addSound(String name, OneSound sound) {
        ContentValues contentValues = new ContentValues();
//            contentValues.put("name", sound.name);
        contentValues.put("id", sound.id);
        contentValues.put("time", sound.time);
        contentValues.put("cnt", sound.cnt);
        database.insert(name, null, contentValues);
    }
    // 听说少产生一些对象会减少内存消耗?
    public void addSound(String name, int id, long time, int cnt) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("id", id);
        contentValues.put("time", time);
        contentValues.put("cnt", cnt);
        database.insert(name, null, contentValues);
    }
    public String findSoundGroupName(){
        String name ;
        int i = (int) (Math.random()*1000000);
        name = String.format("%8s%d", JSON, i);
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name);
        helper.createTable(database, name);
        database.insert(GROUP, null, contentValues);
        return name;
    }
    public void deleteSoundGroup(String name){
        // 万恶的 where 表达式
        database.delete(name, null, null);
        database.execSQL("DROP TABLE " + name);
        database.delete(GROUP, "name = ?", new String[]{name});
    }
    public void renameSoundGroup(String oldName,String newName){
        ContentValues contentValues = new ContentValues();
        contentValues.put("name",newName);
        // 万恶的 where 表达式
        database.update(GROUP, contentValues, "name = ?", new String[]{oldName});
        database.execSQL("ALTER TABLE " + oldName + " RENAME TO " + newName);
    }
    public ArrayList<OneSound> querySounds(String name){
        ArrayList<OneSound> oneSoundArrayList = new ArrayList<>();
        Cursor cursor = database.query(name,
```

```
                    null, null,
                    null, null,
                    null, "cnt");
            while(cursor.moveToNext()){
                oneSoundArrayList.add(new OneSound(
//                          cursor.getString(cursor.getColumnIndex("name")),
                        cursor.getInt(cursor.getColumnIndex("id")),
                        cursor.getLong(cursor.getColumnIndex("time"))
                        // sound 的顺序已经由游标本身确定了，所以不用构造
                ));
            }
            cursor.close();
            return oneSoundArrayList;
        }
        public String[] queryGroups(){
            ArrayList<String> namesList = new ArrayList<>();
            String[] names;
            Cursor cursor = database.query(GROUP,
                    null, null,
                    null, null,
                    null, null);
            while(cursor.moveToNext()){
                namesList.add(cursor.getString(cursor.getColumnIndex("name")));
            }
            cursor.close();
            names = new String[namesList.size()];
            for(int i = 0; i < namesList.size(); i++)
                names[i] = namesList.get(i);
            return names;
        }
        private void onDestroy(){
            database.close();
            helper.close();
        }
        @Override
        public void close(){
            onDestroy();
        }
}
```

代码相关说明：（代码中也有非常人性化的注解）方法的功能说明如下。

```
    private void onCreate(Context context)
    private void onDestroy()
```

由于功能神似生命周期方法，所以就采用了类似的命名法。

```
    @Override
```

```
public void close()
```
关闭该类，释放内存。
```
public String findSoundGroupName()
```
创建一个新的“作品”。
```
public String[] queryGroups()
```
读取所有的“作品”。
```
public void addSound(String name, int id, long time, int cnt)
public void addSound(String name, OneSound sound)
```
往指定“作品”中输入一个字段。
```
public void addSounds(String groupName, List<OneSound> sounds)
```
往指定“作品”中输入一组字段。
```
public void addSounds(String groupName, List<OneSound> sounds)
```
从指定“作品”中读取一个字段。
```
public ArrayList<OneSound> querySounds(String name)
```
从指定“作品”中读取一组字段。
```
public void deleteSoundGroup(String name)
```
删除指定的“作品”。
```
public void renameSoundGroup(String oldName,String newName)
```
重命名指定的“作品”。
为了使代码更加优美，字段也被封装成了类。

```
package util;
public class OneSound {
	public int id;
	public int cnt;
	public long time;
	public OneSound(int id, long time, int cnt) {
		this.id = id;
		this.time = time;
		this.cnt = cnt;
	}
	public OneSound( int id, long time) {
		this.id = id;
		this.time = time;
		cnt = 0;
	}
	public OneSound() { }
}
```

最后提供了一个空构造方法以防万一。
为了方便数据操作，所有数据全部公开。

为了能更加详细地解释封装好的 SQLite 操作，下面提供“作品”在外部存取时的代码。
主界面“录制”按钮的监听器：

```
@TargetApi(Build.VERSION_CODES.JELLY_BEAN)
public void writeGhostAnimal(View v){
	if (isWriting) {
```

```
                stopPlaying();
                isWriting = false;
                ((TextView)v).setText(R.string.start);
            } else {
                startPlaying();
                isWriting = true;
                idCounter = 0;
                ((TextView)v).setText(R.string.stop);
        }
```

各个变量的作用很明显，不再赘述。

```
/**  开始播放方法  */
    private void startPlaying()    {
        manager = new DatabaseManager(this);
        sounds = new ArrayList<>();
        fileName = manager.findSoundGroupName();
    }
    /**  结束播放方法  */
    private void stopPlaying(){
        Toast.makeText(this,"已停止",Toast.LENGTH_SHORT).show();
        manager.addSounds(fileName, sounds);
        manager.close();
    }
```

作用已经在 Javadoc 中说明。

```
public void playSound(View v){
    if(isWriting){
        saveData(v.getId());
    }
   playAudios.chooseOneToPlay(v.getId());
}
```

这是每个图片的点击监听事件。
方法的作用均可顾名思义，不过核心方法 saveData()
是必须展示的。
这段代码是处理数据的关键接口。

```
    /**  保存数据方法  */
    private void saveData(final int id){
//          先把上次的 time 保存起来
        long lastTime = timeCounter;
        long fuck = 0;
//          然后再更新 time 的值
        timeCounter = System.currentTimeMillis();
//          如果不是第一次就把时间记录下来
        if(idCounter > 0)
            fuck = timeCounter - lastTime;
        sounds.add(new OneSound(id, fuck, idCounter));
//          manager.addSound(FILENAME, id, fuck,  idCounter);
```

```
            idCounter++;
        }
```

变量 fuck 代表本次按下按钮的时间与上次的差。加上图片在 xml 中定义的、通过监听器获取的 id 和 SQLite 封装中获取的 name，就构成了一个 OneSound 对象。产生对象并将它放到一个 ArrayList 容器中。当然也可以使用不产生对象，一个字段一个字段传递的方法。不过那样的话可能会对高速按钮不利，因为频繁访问数据库，过多的“事务”会让 SQLite 占用很多内存。

这样就构成了一个完整的数据存储系统。
在一次性添加一组字段时，SQLite 封装采用了事务处理，节省内存。

数据库的访问本身是一个很耗时的工作。在大多数情况下应该为数据库操作单独开启线程，并在 Activity 中定义一个 Handler 与之通信，保证主线程不被堵塞或者占用，影响用户体验。由于谷歌考虑到形似

```
    new Thread(){
        @Override
        public void run(){
                Message message = Message.obtain(handler,
                R.id.decode_succeeded, rawResult);
                Bundle bundle = new Bundle();
                bundleThumbnail(source, bundle);
                message.setData(bundle);
                message.sendToTarget();
        }
    }.start();
```

加上

```
private Handler brainMessageHandler;
    brainMessageHandler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what){
                case T.ANSWER_MESSAGE_SENT:
                    adapter.notifyItemInserted(brain.getDataSize()-1);
                    break;
                default:
                    break;
            }
        }
    };
```

这种形式的线程通讯过于累赘，代码量大，于是封装了 AsnycTask 类，具体实现与本文内容无关，不再赘述。
但是由于本 APP 的 SQLite 操作仅仅是获取一个 TEXT 字段，几个 INTEGER，所以没有单

独开线程，直接在主线程访问 SQLite。

每次新建一个“作品”时，首先随机产生一个字符串，然后在索引表（后文中有说明）中加入这个字符串。然后以该字符串为表名，创建一个表。这样的表数据结构都是相同的，只是表名不同。在接下来的数据库操作中，当用户按下“停止”的时候，将整个放字段的 ArrayList 发送给 SQLite，SQLite 收到 ArrayList 之后开启一个事务，依次将数据存入对应的表中。开启事务也在一定程度上弥补了数据库操作耗时的缺点。

然后是读取端。这是“我的作品”的 Activity 中的代码。作品用一个 ListView 呈现。

```
private boolean findArrays(){
        manager = new DatabaseManager(this);
        names = manager.queryGroups();
        manager.close();
        return names.length >= 0;
    }
```

这个方法读取了前文中提到的“索引表”中的所有字段。这些字段分别对应各自的“作品”的表名。这样的访问效率很高，因为不涉及 where 表达式相关的过滤方法，只需要获取全部表名。这些表名分别映射到一个个的“作品”。

播放的代码实现：

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, final int arg2, long arg3) {
                new Thread() {
                    @Override
                    public void run() {
                        int id;
                        long time;
                        manager = new DatabaseManager(RecordsActivity.this);
                        try {
                            ArrayList<OneSound> sounds =
manager.querySounds(names[arg2]);
                            for (int i = 0; i < sounds.size(); i++) {
                                id = sounds.get(i).id;
                                time = sounds.get(i).time;
                                Thread.sleep(time);
                                if (id != 0)
                                    player.chooseOneToPlay(id);
                            }
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                        manager.close();
                    }
                }.start();
            }
        });
```

一目了然。开线程，放声音，睡线程。

对于重命名和删除操作的实现，本处给出代码。

```
switch (requestCode){
      case 1:
            switch (resultCode){
                  case RESULT_OK:
                        manager = new DatabaseManager(RecordsActivity.this);
                        if(data.getBooleanExtra(DELETE, false))
                              manager.deleteSoundGroup(data.getStringExtra(ORIGINAL_NAME));
                        else
                              manager.renameSoundGroup(data.getStringExtra(ORIGINAL_NAME),
data.getStringExtra(NEW_NAME));
                        manager.close();
                        break;
                  case RESULT_CANCELED:
                        break;
                  default:
                        break;
            }
            break;
      default:
            break;
}
```

很简单的代码，所以省略解释。唤起 Dialog 是通过 startActivityForResult 实现的。

总结：这种以映射的方式封装数据的方法简明易懂方便操作，时间上并不复杂。
但是这样也是在牺牲了空间的基础之上的，每个“作品”对应一个表，着实有些浪费。

2016 年 1 月 24 日