

2018

Android 移动开发实训

程源

广东机电职业技术学院

2018/9/1

目录

7.1 本章目标	- 2 -
7.2 SQLite 数据库	- 2 -
7.2.1 SQLite 简介	- 3 -
7.2.2 SQL 基础知识	- 3 -
7.2.3 定义数据库架构/模式 (Schema)	- 4 -
7.3 建立数据库	- 6 -
7.4 添加数据	- 10 -
7.4.1 修改 Wordlib.....	- 11 -
7.4.2 使用 ContentValues	- 12 -
7.4.3 插入和更新行	- 13 -
7.4.4 更新数据	- 14 -
7.5 从数据库中读取数据	- 16 -
7.5.1 使用 CursorWrapper.....	- 17 -
7.5.2 创建模型层对象	- 19 -
7.6 初始化 SQLITE 数据库	- 21 -

7 使用 SQLite 数据库

7.1 本章目标

将数据保存在单列里终归不是数据存储的长久之计，第 3 章介绍的 `SharedPreferences` 也只能保存少量的轻量级数据，无法满足 `NEC Vocab` 应用持久化保存大量词汇数据的需求。

Android 为此提供了数据的长期存储地：手机或平板设备闪存上的本地文件系统。SQLite 数据库就是这样一个本地文件系统。SQLite 是类似于 MySQL 和 Postgresql 等开源关系型数据库。不同于其他数据库的是，SQLite 使用单个文件存储数据，使用 SQL 语言读取数据。

Android 标准库包含 SQLite 库以及配套的一些 Java 辅助类。

本章，我们将升级 `NEC Vocab` 应用，演示如何使用 Android 的嵌入式数据库引擎 SQLite

7.2 SQLite 数据库

数据库是企业应用程序开发的重要组成部分。但对小规模应用而言，它们太过昂贵、也太笨拙了。近年来，随着小型嵌入式引擎（如 Android 平台自带的引擎）的推出，情况发生了变化。Android 设备上的应用都有一个沙盒目录。将文件保存在沙盒中，可阻止其他应用甚至是设备用户的访问和窥探。（当然，设备被 root 了的话，用户就可以随意访问各种目录和文件了。）

应用沙盒目录是 `/data/data/[your package name]` 的子目录。例如，`NEC Vocab` 应用的沙盒目录是 `/data/data/com.studio.aime.necvocab`。

需要保存大量数据时，大多数应用都不会使用类似 `txt` 这样的普通文件。原因很简单：假设将词汇写入了这样的文件，在仅需要修改某词条时，就得首先读取整个文件的内容，完成修改后再全部保存。数据量大的话，这将非常耗时。怎么办呢？

Android 标准库提供了 SQLite 数据库以及配套的一些 Java 辅助类。

7.2.1 SQLite 简介

SQLite 是一款轻量级的关系数据库，由 Richard.Hipp 博士于 2000 年开发。SQLite 无疑是全球部署最广泛的 SQL 数据库引擎。除 Android 外，其身影还在 Apple iPhone 等众多地方出现。

SQLite 为何如此深受欢迎？下面是其中 3 个原因：

- 它是免费的。作者放弃了版权，不对用户收取任何费用；
- 它很小。最新版本仅为 150KB，在 Android 手机的内存中容纳它绰绰有余；
- 它不需要安装和管理。不需要服务器，没有配置文件，不需要数据库管理员。

SQLite 数据库就是一个文件。移动这个文件，甚至将其复制到其他系统（如 PC 机），它依然能够正常运行。Android 通常将数据库文件存放在目录 `/data/data/packageName/databases` 中。可以用命令 `adb` 来查看、移动和删除它。

要在程序中访问这个文件，需要运行结构化查询语言（SQL）语句，而不是调用 Java I/O 例程。Android 通过辅助类和便利方法隐藏了一些 SQL 语法，但要使用 SQLite，还是需要对 SQL 语言有个大致了解。

7.2.2 SQL 基础知识

为使用 SQL 数据库，需要提交 SQL 语句并获取返回结果。SQL 语句分为三大类：DDL 语句、修改语句和查询语句。注意，SQL 不区分大小写。

1. DDL 语句

数据库文件包含很多表。表由行组成，其中每行包含的列数都是固定的。表的每列都有名称和数据类型（文本字符串、数字等）。首先要运行 DDL（数据定义语言）语句来定义这些表和列。例如，下面的语句用于创建一个包含 5 列的表。

```
create table vocab_table (  
  _id integer primary key autoincrement,  
  vocab text,  
  meaning text,  
  example text,  
  image_id text );
```

其中第一列被指定为主键——唯一标识行的数字。AUTOINCREMENT 意味着每添加一条记录，数据库就将主键加 1，以确保每条记录都是唯一的。根据约定，第一列总是被命名为 `_id`。SQLite 并不要求表必须包含 `_id` 列，但后面使用 Android 内容提供者时需要它。

不同于其他大多数数据库，在 SQLite 中，列的类型只是提示。如果试图在整形列中存储字符串或在字符串列中存储整形数据，不会导致任何错误。这是 SQLite 的特色，而非 bug。

2. 修改语句

SQL 提供了很多让用户能够在数据库中插入、删除和更新记录的语句。例如要添加几个电话号码，可以使用如下代码。

```
insert into vocab_table values(null, 'dolphin', 'noun', '海豚', 'A dolphin is a mammal which lives in the sea and looks like a large fish with a pointed mouth.', 'dolphin.png');
insert into vocab_table values(null, 'education', 'noun', '教育', 'Education involves teaching people various subjects, usually at a school or college, or being taught.', 'education.png');
insert into vocab_table values(null, 'lion', 'noun', '狮子', 'He is as brave as a lion.', 'lion.png');
```

值的排列顺序与 create table 语句中列的排列顺序相同。这里将 `_id` 列的值指定为 `null`，因为 SQLite 会为我们计算该列的值。

3. 查询语句

在表中添加数据后，就可以使用 SELECT 语句对表进行查询了。例如要获取第三条记录，可以采用如下做法。

```
select * from vocab_table where(_id=3);
```

通常我们希望根据单词查询该单词的释义。下面语句展示了如何查找所有单词中以 'ed' 开头的单词的记录。

```
select vocab from vocab_table where(vocab like 'ed%');
```

了解了 SQL 后，下面来看看如何在 NEC Vocab 应用中使用这些知识。

7.2.3 定义数据库架构/模式 (Schema)

在创建数据库之前，必须确定该数据库中存放的是什么。NEC Vocab 存储一张词汇表，所以需要定义一个名为 vocab_table 的表（表 7-1）

表 7-1 vocab_table 表

<code>_id</code>	<code>uuid</code>	<code>vocab</code>	<code>pos</code>	<code>meaning</code>	<code>example</code>	<code>image_id</code>	<code>killed</code>
1	87e93a6.....	dolphin	noun	海豚	A dolphin is.....	dolphin.png	0
2	4a7725b.....	education	noun	教育	Education involves.....	education.png	1
.....

在本章中，我们在 java 代码中定义一个数据库架构，即定义表名和列。

我们先创建一个名为 **VocabDbSchema** 的类来存放数据库架构。在 **New Class** 对话框中，输入 **dao.VocabDbSchema**。这样 VocabDbSchema.java 文件将被放置在它自己的数据库包中，我们可以用该包来组织所有数据库相关的代码。

如代码清单 7-1 所示，在 VocabDbSchema 内部，定义一个称为 VocabTable 的内部类描述表结构。

代码清单 7-1 定义 VocabTable (VocabDbSchema.java)

```
public class VocabDbSchema {
    public static final class VocabTable {
        public static final String  = "vocab_table";
    }
}
```

VocabTable 类是为了定义描述表所需的字符串常量。该定义的第一部分是数据库表的名称：VocabTable.NAME。

接下来，定义列（参见代码清单 7-2）。

代码清单 7-2 定义列 (VocabDbSchema.java)

```
public class VocabDbSchema {
    public static final class VocabTable {
        public static final String NAME = "vocab_table";
        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String VOCAB = "vocab";
            public static final String POS = "part_of_speech";
            public static final String MEANING = "meaning";
            public static final String EXAMPLE = "example";
            public static final String IMAGEID = "image_id";
            public static final String KILLED = "killed";
        }
    }
}
```

这样，我们就能够引用列，如名为“vocab”的列：VocabTable.Cols.VOCAB。如果需要更改该列的名称，或将该列的附加数据添加到表中，则可以更改程序。

7.3 建立数据库

定义了数据库架构之后就可以创建数据库。

Android 为了让我们能够更加方便地管理数据库，专门提供了一个 SQLiteOpenHelper 辅助类，借助这个类就可以非常简单地创建和升级数据库。下面介绍 SQLiteOpenHelper 的基本用法。

首先 SQLiteOpenHelper 是一个抽象类，这就意味着如果我们想要使用它的话，就需要创建一个自己的辅助类来继承它。SQLiteOpenHelper 中有两个抽象方法，分别是 onCreate() 和 onUpgrade()，我们必须在自己的辅助类里覆盖这两个方法，然后分别在这两个方法中实现创建、升级数据库的逻辑。

如代码清单 7-3 所示，在 dao 包中创建一个名为 VocabBaseHelper 的辅助类。

代码清单 7-3 创建 VocabBaseHelper (VocabBaseHelper.java)

```
public class VocabBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "VocabBase.db";

    public VocabBaseHelper (Context context){
        super(context, DATABASE_NAME, null, VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {

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

    }
}
```

按照惯例，我们在 VocabBaseHelper 的构造方法中调用了超类 SQLiteOpenHelper 的构造方法。这个构造方法接收四个参数，第一个参数是 Context，这个没什么可说的，必须有它才能对数据库进行操作；第二个参数是数据库名，使用数据库时使用的就是这里指定的名称；第三个参数允许我们在查询数据时返回一个自定义的 Cursor，一般都是传入 null；第四个参数表示当前数据库的版本号，用于对数据库进行升级操作。

目前 onCreate() 和 onUpgrade() 方法什么也不做，但有了它们，我们便能够在 Wordlib 中利用 VocabBaseHelper 创建数据库。

如代码清单 7-4 所示，我们在 **Wordlib** 中利用 **VocabBaseHelper** 创建数据库。

代码清单 7-4 打开 **SQLiteDatabase** (**Wordlib.java**)

```
public class WordLib {
    private static WordLib sWordLib;
    private List<WordEntity> mWordList; // 创建存储数量类型为 WordEntity 对象的序列
    private Context mContext;
    private SQLiteDatabase mDatabase;

    public static WordLib get(Context context){
        if (sWordLib == null) {
            sWordLib = new WordLib(context);
        }
        return sWordLib;
    }
    private WordLib(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new VocabBaseHelper(mContext).getWritableDatabase();
        mWordList = new ArrayList<>();
        mWordList.addAll(sWords);
    }
    public List<WordEntity> getWordList() {
        return mWordList;
    }
    public WordEntity getWordEntity(UUID id){
        for (WordEntity wordEntity : mWordList){
            if (wordEntity.getId().equals(id)){
                return wordEntity;
            }
        }
        return null;
    }
}
.....
```

在这里我们调用了 **getWritableDatabase()** 方法。VocabBaseHelper 中提供了两个非常重要的实例方法，**getWritableDatabase()** 和 **getReadableDatabase()** 方法。这两个方法都可以创建或打开一个现有的数据库（如果数据库已经存在则直接打开，否则创建一个新的数据库），并返回一个可对数据库进行读写操作的对象。两个方法不同之处在于，当数据库不可写入时（如磁盘空间已满），**getReadableDatabase()** 方法返回的对象将以只读的方式打开数据库，而 **getWritableDatabase()** 方法将出现异常。

具体而言，在代码清单 7-4 中，**getWritableDatabase()** 将完成以下几项工作：

1. 打开 `/data/data/com.studio.aime.necvocab/database/VocabBase.db`，如果不存在，创建一个新的数据库文件。
2. 如果是首次创建数据库，调用的 `onCreate(SQLiteDatabase)`，然后储存最新的版本号。

3. 如果这是不是首次打开, 检查数据库的版本号。如果 VocabOpenHelper 的版本号更高, 调用 `onUpgrade(SQLiteDatabase, int, int)`。

其结果是这样的: 将创建数据库的代码放在 `onCreate(SQLiteDatabase)` 中, 将处理升级的代码放在 `onUpgrade(SQLiteDatabase, int, int)` 中。

目前, NEC Vocab 只有一个版本, 因此暂时忽略 `onUpgrade(...)`。只需要在 `onCreate(...)` 中创建数据库表。

在 `SQLiteDatabase` 类中封装了一些操作数据库的 API, 使用该类可以完成对数据进行添加(Create)、查询(Retrieve)、更新(Update)和删除>Delete)操作(这些操作简称为 CRUD)。其中 `execSQL(...)` 方法可以执行 `create table`、`insert`、`delete` 和 `update` 之类有更改行为的 SQL 语句; `rawQuery(...)` 方法用于执行 `select` 语句。

`execSQL()` 方法的使用例子:

```
SQLiteDatabase db = ....;
db.execSQL("insert into person(name, age) values('测试数据', 4)");
db.close();
```

注释: 执行上面 SQL 语句会往 `person` 表中添加进一条记录。在实际应用中, 语句中的“测试数据”这些参数值由用户输入界面提供, 如果把用户输入的内容原样组拼到上面的 `insert` 语句, 当用户输入的内容含有单引号时, 组拼出来的 SQL 语句就会存在语法错误。要解决这个问题需要对单引号进行转义, 也就是把单引号转换成两个单引号。有些时候用户往往还会输入像 “&” 这些特殊 SQL 符号, 为保证组拼好的 SQL 语句语法正确, 必须对 SQL 语句中的这些特殊 SQL 符号都进行转义, 显然, 对每条 SQL 语句都做这样的处理工作是比较烦琐的。 `SQLiteDatabase` 类提供了一个重载后的 `execSQL(String sql, Object[] bindArgs)` 方法, 使用这个方法可以解决前面提到的问题, 因为这个方法支持使用占位符参数(?)。使用例子如下:

```
SQLiteDatabase db = ....;
db.execSQL("insert into person(name, age) values(?,?)", new Object[]{"测试数据", 4});
db.close();
```

`execSQL(String sql, Object[] bindArgs)` 方法的第一个参数为 SQL 语句, 第二个参数为 SQL 语句中占位符参数的值, 参数值在数组中的顺序要和占位符的位置对应。

下面我们来完善 `VocabBaseHelper` 类中的 `onCreate(...)` 方法, 即在 `onCreate(...)` 中创建数据库表。为此, 需要引用 `VocabDbSchema` 的内部类 `VocabTable`。

首先, 完成 SQL 创建代码的起始部分, 如代码清单 7-5 所示:

代码清单 7-5 利用 `onCreate(...)` 创建数据库表 (`VocabBaseHelper.java`)

```
public class VocabBaseHelper extends SQLiteOpenHelper {
```

```

private static final int VERSION = 1;
private static final String DATABASE_NAME = "VocabBase.db";

public VocabBaseHelper (Context context){
    super(context, DATABASE_NAME, null, VERSION);
}
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + VocabDbSchema.VocabTable.NAME);
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVwesion){

}
}

```

可以看到，我们把建表语句作为字符串常量，并在 `onGreate(...)` 方法中调用 `SQLiteDatabase` 的 `execSQL(...)` 方法来执行建表语句。这样，可以保证在数据库创建的同时创建 `Vocabs` 表。

当然，表的创建还没有完成，我们还要引用 `VocabDbSchema.VocabTable` 中的其他字符串常量，如 `UUID` 等。但每次引入字符串常量都要写入 `VocabDbSchema.VocabTable...` 一长串类型名称，太过繁琐。

为简化输入，将光标放在 `VocabTable` 上，按下 **Option+Return** (**Alt + Enter**) 键，然后选择：**Add import for 'com.me.android.Vocabing.database.VocabDbSchema.VocabTable'**，如图 7-2 所示。



图 7-2 导入 VocabTable

Android Studio 会产生如下导入项：

```

.....
import com.studio.aime.necvocab.dao.VocabDbSchema.VocabTable;
public class VocabBaseHelper extends SQLiteOpenHelper {
.....

```

这样，当引用 `VocabDbSchema.VocabTable` 中的字符串常量，如 `UUID` 时，我们只需键入 `VocabTable.Cols.UUID`，而不必键入全部的 `VocabDbSchema.VocabTable.Cols.UUID`。

接下来完成表定义的代码（代码清单 7-6）。

代码清单 7-6 创建 `Vocabs` 表 (`VocabBaseHelper.java`)

```

public class VocabBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;

```

```

private static final String DATABASE_NAME = "VocabBase.db";
public VocabBaseHelper (Context context){
    super(context, DATABASE_NAME, null, VERSION);
}
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create TABLE " + VocabTable. + "(" +
        "_id integer primary key autoincrement, " +
        VocabTable.Cols.UUID + ", " +
        VocabTable.Cols.VOCAB + ", " +
        VocabTable.Cols.POS + ", " +
        VocabTable.Cols.MEANING + ", " +
        VocabTable.Cols.EXAMPLE + ", " +
        VocabTable.Cols.IMAGEID + ", " +
        VocabTable.Cols.KILLED + ")"
    );
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVwesion){
}
}

```

在 SQLite 中创建一个表要比在其他数据库中简单：不必在创建时指定列的类型。这样能节省很多工作量。

数据库调试问题

当编写代码处理 SQLite 数据库时，有时需要调整数据库的布局。例如，当我们需要为每一个词条添加一个音标时，这将需要在表上增加一列。“正确”的方式是在 SQLiteOpenHelper 中写代码写上版本号，然后在 onUpgrade(...)方法内部更新表。

那么，这种“正确”的方式涉及的代码中有相当一部分是不必要的，比如当只是试图获取数据库正确版本时，如 1 或 2，这些代码就是多余的。在实践中，最好的做法是删除数据库，并重新开始，再次调用 SQLiteOpenHelper.onCreate(...)。

删除数据库的最简单方法是从设备上直接卸载应用程序。

对于本章数据库中的表，如果遇到任何问题，记住这个方法（卸载 app）。

7.4 添加数据

我们已经创建了数据库，接下来我们往数据库表中的数存放一些数据。

7.4.1 修改 Wordlib

现在有了数据库，下一步是修改 Wordlib 中代码，换用 `mDatabase` 存储数据，而不再使用 `mWordList`。

首先删除一些代码。在 Wordlib 中删除所有 `mWordList` 相关的代码。

代码清单 12-7 删除 `mWordList` 相关代码 (`Wordlib.java`)

```
public class WordLib {
    private static WordLib sWordLib;
    private List<WordEntity> mWordList;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    public static WordLib get(Context context){
        if (sWordLib == null) {
            sWordLib = new WordLib(context);
        }
        return sWordLib;
    }
    private WordLib(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new VocabBaseHelper(mContext).getWritableDatabase();
        mWordList = new ArrayList<>();
        mWordList.addAll(sWords);
    }
    public List<WordEntity> getWordList() {
        return mWordList;
        return new ArrayList<>();
    }
    public WordEntity getWordEntity(UUID id){
        for (WordEntity wordEntity : mWordList){
            if (wordEntity.getId().equals(id)){
                return wordEntity;
            }
        }
        return null;
    }
    .....
}
```

这暂时会使 Vocabing 处于瘫痪状态。我们只能看到一个空的词汇表。

我们已经创建了数据库，接下来就是如何对表中的数据进行操作。其实我们可以对数据进行的操作无非就是 CRUD 四种。其中 C 代表添加（Create），R 代表查询（Retrieve），U 代表更新（Update），D 代表删除（Delete）。

熟悉 SQL 语言的同学应该知道：添加数据时使用 `insert`，查询数据使用 `select`，更新数据使用 `update`，删除数据使用 `delete`。但是开发者的水平是参差不齐的，未必每一个开发者都能

非常熟悉 SQL 语言。因此 Android 提供了一系列辅助方法，使得在 Android 开发中即使不去编写 SQL 语句，也能轻松完成所有的 CRUD 操作。

我们已经知道，调用 `VocabBaseHelper` 的 `getWritableDatabase()` 或 `getReadableDatabase()` 方法可以创建和升级数据库。不仅如此，这两个方法都返回一个 `SQLiteDatabase` 对象。借助这个对象就可以对数据库进行 CRUD 操作了。

首先，我们需要在 `Vocabing` 的记录表中插入新的行，并在 `Vocabs` 更改时更新那些已经存在的行。

`SQLiteDatabase` 中提供了 `Insert(...)` 方法用于添加数据。其基本用法如下：

```
db.insert(String, String, ContentValues);
```

该方法接收三个参数。第一个参数是表名，我们希望向哪张表里添加数据，这里就传入哪张表名。第二个参数用于在未指定添加数据的情况下给某些可为空的列自动赋值 `NULL`，一般我们用不到这个功能，直接传入 `null` 即可。第三个参数是一个 `ContentValues` 对象，它提供了一系列 `put(...)` 方法重载，用于向 `ContentValues` 中添加数据。

下面我们就来看看如何使用这个 `ContentValues`。

7.4.2 使用 ContentValues

我们将在 `ContentValues` 类的协助下完成写入和更新数据库工作。和 `SharedPreferences` 一样，`ContentValues` 是一个 **key-value** 对存储类。但与 `SharedPreferences` 不同的是，`ContentValues` 是专门为存储各类 `SQLite` 数据而设计的。

在 `Wordlib` 中，我们将多次创建 `ContentValues` 实例。为此，添加一个私有方法来，以便多次在 `ContentValues` 中放入 `WordEntity` 变量。利用前面相同的步骤添加 `VocabTable` 的导入包：将光标放在 `VocabTable.Cols.UUID` 的 `VocabTable` 上，键入 **Option+Return**（**Alt+Enter** 键）键，选择 **Add import for “com.studio.aime.necvocab.database.VocabDbSchema.Vocabable”**。

代码清单 7-8 创建 ContentValues (Wordlib.java)

```
.....
public WordEntity getWordEntity(UUID id){
    return null;
}
private static ContentValues getContentValues(WordEntity wordEntity){
    ContentValues contentValues = new ContentValues();
    contentValues.put(VocabTable.Cols.UUID, wordEntity.getId().toString().trim());
    contentValues.put(VocabTable.Cols.VOCAB, wordEntity.getEnglish().trim());
}
```

```

        contentValues.put(VocabTable.Cols.POS, wordEntity.getPartOfSpeech().trim());
        contentValues.put(VocabTable.Cols.MEANING, wordEntity.getChinese().trim());
        contentValues.put(VocabTable.Cols.EXAMPLE, wordEntity.getExample().trim());
        contentValues.put(VocabTable.Cols.IMAGEID, wordEntity.getImgID());
        contentValues.put(VocabTable.Cols.KILLED, wordEntity.isKilled() ? 1 : 0);
        return contentValues;
    }
    .....

```

对于键，我们使用列名。这些都不是任意名称；他们指定需要插入或更新的列。如果拼写错误或相比数据库内容敲错字，插入或更新将失败。除了 `_id`，每一列都在这里指定，`_id` 是自动创建的唯一行 ID。

7.4.3 插入和更新行

现在，既然有了 `ContentValues`，便可以将行添加到数据库中了。如代码清单 7-9 所示，`addVocab(Vocab)` 方法向数据库插入一行，`addVocabList()` 方法则向数据库插入一组数据。在这里，我们先将存放在 `sWords` 中的 120 条词汇数据插入的数据库中。

代码清单 7-9 插入数据 ([Wordlib.java](#))

```

.....
private WordLib(Context context) {
    mContext = context.getApplicationContext();
    mDatabase = new VocabBaseHelper(mContext).getWritableDatabase();
    addVocabList();
}
.....
public void addVocab(WordEntity wordEntity) {
    ContentValues contentValues = getContentValues(wordEntity);
    mDatabase.insert(VocabTable.NAME, null, contentValues);
}
public void addVocabList() {
    for (WordEntity wordEntity : sWords){
        addVocab(wordEntity);
    }
}
.....

```

需要注意的是，我们只在私有构造函数中完成了数据插入，即数据库的初始化。也就是只有重新创建单列时，才将所有数据插入数据库。应用在内存中驻留多久，单列就存在多久。因此，除非用户清除 NEC Vocab 应用，否则不会再次初始化数据库。

这里还是存在一个不易被发现的重大问题，请读者思考一下是什么问题。我们将在本章最后来解决它。

7.4.4 更新数据

完成了添加数据后，接下来看看如何修改表中已有的数据。SQLiteDatabase 中提供了一个非常好用的 `update(String, ContentValues, String, String[])` 方法用于对数据进行更新。这个方法接收四个参数。第一个参数和 `insert(...)` 方法一样，也是表名，在这里指定更新哪张表里的数据。第二个参数是 `ContentValues` 对象，把要更新的数据在这组装进去。第三、第四个参数用于约束更新某一行或某几行中的数据，不指定的话某人就是更新所有行。

如代码清单 7-10 所示，我们利用 `ContentValues` 完成更新数据库行的方法。

代码清单 7-10 更新记录 (`Wordlib.java`)

```
.....
public void addVocab(WordEntity wordEntity) {
    ContentValues contentValues = getContentValues(wordEntity);
    mDatabase.insert(VocabTable.NAME, null, contentValues);
}
public void addVocabList() {
    for (WordEntity wordEntity : sWords){
        addVocab(wordEntity);
    }
}
public void updateVocab(WordEntity wordEntity){
    String uuidString = wordEntity.getId().toString();
    ContentValues contentValues = getContentValues(wordEntity);
    mDatabase.update(VocabTable.NAME, contentValues,
        VocabTable.Cols.UUID + " = ?",
        new String[] {uuidString});
}
.....
```

可以看到，首先传递我们想要更新的表名和要分配给更新每一行的 `ContentValues`。这里使用了第三、第四个参数来指定具体更新哪几行。第三个参数对应的是 SQL 语句的 `where` 子句，表示去更新 `UUID` 等于“?”的行。这里“?”是个占位符，可以通过第四个参数提供的一个字符串数组为第三个参数中的每个占位符指定相应的内容。因此，上述代码表达的意图是：将 `VocabTable.Cols.UUID` 为字符串 `uuidString` 的记录更改为 `contentValues` 中相应的内容。

你可能想知道为什么你不把 `uuidString` 直接放入 `where` 子句。毕竟，这将比使用“? ”，并把它当作一个 `String[]` 来传递简单一些。

原因是，在某些情况下，字符串本身可能包含 SQL 代码。如果直接把这个字符串放入查询中，该代码可能改变查询的含义，甚至改变数据库。这就是所谓的 SQL 注入攻击，这的确不是一件好事。

如果用“? ”，则把它作为一个字符串值，而不是代码。所以是安全的。

WordEntity 实例在 VocabularyFragment 中更改, 所以添加覆盖其 onPause(), 更新 WordEntity 的 Wordlib 副本。

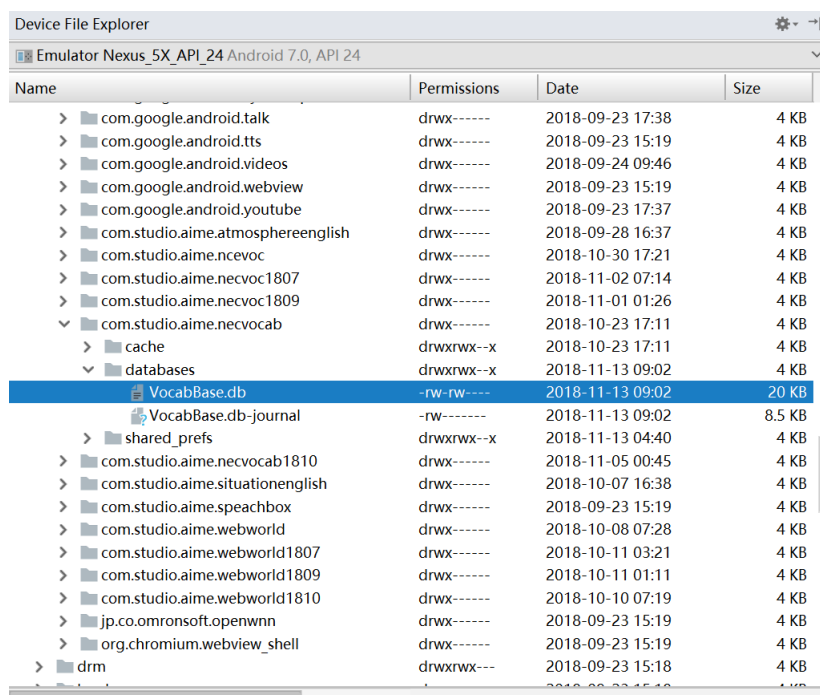
代码清单 7-11 在 VocabularyFragment 中跟新数据 (VocabularyFragment.java)

```
.....
@Override
public void onPause(){
    super.onPause();
    WordLib.get(getActivity()).updateVocab(mWord);
}
.....
```

遗憾的是, 我们目前无法验证这个代码。需要等待, 直到可以在记录中读取更新。要确保所有编译的正确, 在进入下一节之前运行 NEC Vocab 一次。

现在运行 NEC Vocab 应用, 应该看到一个空白列表, 但会创建数据库,

如果在仿真器上运行, 可以直接查看数据库 (见图 7-3): **AndroidStudio** → **View** → **Tool Windows** → **Device File Explorer**, 查看 `/data/data/com.me.android.Vocabing.database/`。



Name	Permissions	Date	Size
> com.google.android.talk	drwx-----	2018-09-23 17:38	4 KB
> com.google.android.tts	drwx-----	2018-09-23 15:19	4 KB
> com.google.android.videos	drwx-----	2018-09-24 09:46	4 KB
> com.google.android.webview	drwx-----	2018-09-23 15:19	4 KB
> com.google.android.youtube	drwx-----	2018-09-23 17:37	4 KB
> com.studio.aime.atmosphereenglish	drwx-----	2018-09-28 16:37	4 KB
> com.studio.aime.ncevoc	drwx-----	2018-10-30 17:21	4 KB
> com.studio.aime.ncevoc1807	drwx-----	2018-11-02 07:14	4 KB
> com.studio.aime.ncevoc1809	drwx-----	2018-11-01 01:26	4 KB
✓ com.studio.aime.ncevocab	drwx-----	2018-10-23 17:11	4 KB
cache	drwxrwx--x	2018-10-23 17:11	4 KB
databases	drwxrwx--x	2018-11-13 09:02	4 KB
VocabBase.db	-rw-rw----	2018-11-13 09:02	20 KB
VocabBase.db-journal	-rw-----	2018-11-13 09:02	8.5 KB
shared_prefs	drwxrwx--x	2018-11-13 04:40	4 KB
> com.studio.aime.ncevocab1810	drwx-----	2018-11-05 00:45	4 KB
> com.studio.aime.situationenglish	drwx-----	2018-10-07 16:38	4 KB
> com.studio.aime.speechbox	drwx-----	2018-09-23 15:19	4 KB
> com.studio.aime.webworld	drwx-----	2018-10-08 07:28	4 KB
> com.studio.aime.webworld1807	drwx-----	2018-10-11 03:21	4 KB
> com.studio.aime.webworld1809	drwx-----	2018-10-11 01:11	4 KB
> com.studio.aime.webworld1810	drwx-----	2018-10-10 07:19	4 KB
> jp.co.omronsoft.openwnn	drwx-----	2018-09-23 15:19	4 KB
> org.chromium.webview_shell	drwx-----	2018-09-23 15:19	4 KB
> drm	drwxrwx---	2018-09-23 15:18	4 KB

图 7-3 查看数据库

7.5 从数据库中读取数据

读取 SQLite 数据库中数据需要用到 `query(...)` 方法。这个方法有好几个重载版本。我们要用的版本如下：

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

由于之前没有接触过 SQL，这里我们只关心将会使用的参数，如上述代码中粗体部分。

参数 `table` 是要查询的数据表。参数 `columns` 指定要依次获取哪些字段的值。参数 `where` 和 `whereArgs` 的作用与 `update(...)` 方法中的一样。

新增一个便利方法调用 `query(...)` 方法查询 `VocabTable` 中的记录，如代码清单 7-12 所示。

代码清单 7-12 Querying for Vocab (`Wordlib.java`)

```
.....
private static ContentValues getContentValues(WordEntity wordEntity){
    ContentValues contentValues = new ContentValues();
    contentValues.put(VocabTable.Cols.UUID, wordEntity.getId().toString().trim());
    contentValues.put(VocabTable.Cols.VOCAB, wordEntity.getEnglish().trim());
    contentValues.put(VocabTable.Cols.POS, wordEntity.getPartOfSpeech().trim());
    contentValues.put(VocabTable.Cols.MEANING, wordEntity.getChinese().trim());
    contentValues.put(VocabTable.Cols.EXAMPLE, wordEntity.getExample().trim());
    contentValues.put(VocabTable.Cols.IMAGEID, wordEntity.getImgID());
    contentValues.put(VocabTable.Cols.KILLED, wordEntity.isKilled() ? 1 : 0);
    return contentValues;
}
public void addVocab(WordEntity wordEntity) {
    ContentValues contentValues = getContentValues(wordEntity);
    mDatabase.insert(VocabTable.NAME, null, contentValues);
}
public void addVocabList() {
    for (WordEntity wordEntity : sWords){
        addVocab(wordEntity);
    }
}
public void updateVocab(WordEntity wordEntity){
    String uuidString = wordEntity.getId().toString();
    ContentValues contentValues = getContentValues(wordEntity);
    mDatabase.update(VocabTable.NAME, contentValues,
        VocabTable.Cols.UUID + " = ?",
        new String[] {uuidString});
}
```

```
private Cursor queryWords(String whereClause, String[] whereArgs){
    Cursor cursor = mDatabase.query(
        VocabTable.NAME,
        null, //Columns - null selects all columns
        whereClause,
        whereArgs,
        null, //groupBy
        null, //having
        null //orderBy
    );
    return cursor;
}
.....
```

7.5.1 使用 CursorWrapper

Cursor 是个神奇的表数据处理工具，其任务就是封装数据表中的原始字段值。从 Cursor 获取数据的代码大致如下所示：

```
String uuidString = getString(getColumnIndex(VocabTable.Cols.UUID));
String vocab = getString(getColumnIndex(VocabTable.Cols.VOCAB));
String pos = getString(getColumnIndex(VocabTable.Cols.POS));
String meaning = getString(getColumnIndex(VocabTable.Cols.MEANING));
String example = getString(getColumnIndex(VocabTable.Cols.EXAMPLE));
String image = getString(getColumnIndex(VocabTable.Cols.IMAGEID));
int isKilled = getInt(getColumnIndex(VocabTable.Cols.KILLED));
```

每从 Cursor 中取出一条 WordEntity，以上代码都要重复写一次。（这还不包括按照这些字段值创建 WordEntity 实例的代码。）

显然，遇到这种情况，我们应考虑到前面说过的代码复用原则。与其机械地编写重复代码，不如创建可复用的专用 Cursor 子类。使用 CursorWrapper 可快速方便地创建 Cursor 子类。顾名思义，CursorWrapper 能够封装一个个 Cursor 的对象，并允许在其上添加新的有用方法。

参照代码清单 7-13，在数据库包中新建 VocabCursorWrapper 类。

代码清单 7-13 创建 VocabCursorWrapper (VocabCursorWrapper.java)

```
public class VocabCursorWrapper extends CursorWrapper {
    public VocabCursorWrapper(Cursor cursor) {
        super(cursor);
    }
}
```

可以看到，以上代码创建了一个 Cursor 封装类。该类继承了 Cursor 类的全部方法。注意，这样封装的目的就是为了定制新方法，以方便操作内部 Cursor。

参照代码清单 7-14，新增获取相关字段值的 getVocab()方法。（别忘了使用前面介绍的两

步导入 VocabTable 的技巧。)

代码清单 7-14 Adding getVocab() method (VocabCursorWrapper.java)

```
import com.studio.aime.necvocab.dao.VocabDbSchema.VocabTable;
public class VocabCursorWrapper extends CursorWrapper {
    public VocabCursorWrapper(Cursor cursor) {
        super(cursor);
    }
    public WordEntity getVocab(){
        String uuidString = getString(getColumnIndex(VocabTable.Cols.UUID));
        String vocab = getString(getColumnIndex(VocabTable.Cols.VOCAB));
        String pos = getString(getColumnIndex(VocabTable.Cols.POS));
        String meaning = getString(getColumnIndex(VocabTable.Cols.MEANING));
        String example = getString(getColumnIndex(VocabTable.Cols.EXAMPLE));
        String image = getString(getColumnIndex(VocabTable.Cols.IMAGEID));
        int isKilled = getInt(getColumnIndex(VocabTable.Cols.KILLED));
        return null;
    }
}
```

我们需要返回具有 UUID 的 WordEntity。在 WordEntity.java 中新增一个有此用途的构造方法，如代码清单 7-15 所示。

代码清单 7-15 添加 WordEntity 的构造函数 (WordEntity.java)

```
.....
public WordEntity(){
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
}

public WordEntity(UUID uuid){
    mId = uuid;
}
.....
```

最后，完成 getVocab()方法，如代码清单 7-16 所示。

代码清单 7-16 完成 getVocab 方法 (VocabCursorWrapper.java)

```
public class VocabCursorWrapper extends CursorWrapper {
    public VocabCursorWrapper(Cursor cursor) {
        super(cursor);
    }
    public WordEntity getVocab(){
        String uuidString = getString(getColumnIndex(VocabTable.Cols.UUID));
        String vocab = getString(getColumnIndex(VocabTable.Cols.VOCAB));
        String pos = getString(getColumnIndex(VocabTable.Cols.POS));
        String meaning = getString(getColumnIndex(VocabTable.Cols.MEANING));
        String example = getString(getColumnIndex(VocabTable.Cols.EXAMPLE));
        String image = getString(getColumnIndex(VocabTable.Cols.IMAGEID));
        int isKilled = getInt(getColumnIndex(VocabTable.Cols.KILLED));
        WordEntity wordEntity = new WordEntity(UUID.fromString(uuidString));
```

```

        wordEntity.setEnglish(vocab);
        wordEntity.setPartOfSpeech(pos);
        wordEntity.setChinese(meaning);
        wordEntity.setExample(example);
        wordEntity.setImgID(valueOf(image));
        wordEntity.setKilled(isKilled != 0);
        return wordEntity;
    }
}

```

7.5.2 创建模型层对象

使用 VocabCursorWrapper 类，可直接从 Wordlib 中取得 List< WordEntity >。大致思路无外乎将查询返回的 cursor 封装到 VocabCursorWrapper 类中，然后调用 getVocab()方法遍历取出 Vocab。

首先让 queryVocabs (...)方法返回 VocabCursorWrapper 对象，如代码清单 7-17 所示。

代码清单 7-17 使用 cursor 封装 (Wordlib.java)

```

private Cursor queryVocabs(String whereClause, String[] whereArgs){
private VocabCursorWrapper queryVocabs(String whereClause, String[] whereArgs){
    Cursor cursor = mDatabase.query(
        VocabTable.NAME,
        null, //Columns - null selects all columns
        whereClause,
        whereArgs,
        null, //groupBy
        null, //having
        null //orderBy
    );
return cursor;
    return new VocabCursorWrapper(cursor);
}

```

然后，完善 getVocabs()方法：遍历查询取出所有的 WordEntity，返回 WordEntity 数组对象，如代码清单 7-18 所示。为所有记录查询添加代码，行游标，并填充记录列表。

代码清单 7-18 返回 WordEntity 列表 (Wordlib.java)

```

.....
public List<WordEntity> getWordList() {
    List<WordEntity> wordList = new ArrayList<>();
    VocabCursorWrapper cursorWrapper = queryVocabs(null, null);
    try {
        cursorWrapper.moveToFirst();
        while (!cursorWrapper.isAfterLast()){
            wordList.add(cursorWrapper.getVocab());
            cursorWrapper.moveToNext();
        }
    }
}

```

```

    }
    } finally {
        cursorWrapper.close();
    }
    return wordList;
}
.....

```

数据库 `cursor` 之所以被称为 `cursor`，是因为它内部就像有根手指似的，总是指向查询的某个地方。因此，要从 `cursor` 中取出数据，首先要调用 `moveToFirst()` 方法移动虚拟手指指向第一个元素。读取行记录后，再调用 `moveToNext()` 方法，读取下一行记录，直到 `isAfterLast()` 告诉我们没有数据可取为止。

最后，别忘了调用 `Cursor` 的 `close()` 方法关闭它。否则，后果很严重：轻则看到应用报错，重则导致应用崩溃。

`Wordlib.getVocab(UUID)` 方法类似于 `getVocabs()` 方法，唯一区别就是它只需要取出已存在的首条记录，如代码清单 7-19 所示。

代码清单 7-19 重写 `getWordEntity(UUID)` 方法（`Wordlib.java`）

```

.....
public WordEntity getWordEntity(UUID id){
    VocabCursorWrapper cursorWrapper = queryVocabs(
        VocabTable.Cols.UUID + " = ?",
        new String[] {id.toString()}
    );
    try {
        if (cursorWrapper.getCount() == 0){
            return null;
        }
        cursorWrapper.moveToFirst();
        return cursorWrapper.getVocab();
    } finally {
        cursorWrapper.close();
    }
}
.....

```

上述代码的作用如下：

- 现在可以插入记录了。也就是说，点击新建纪录菜单项，实现将 `Vocab` 添加到 `Wordlib` 的代码可以正常工作了。
- 数据库查询没有问题了。`VocabularyItemFragment` 现在能够看见 `Wordlib` 中的所有词汇了。
- `Wordlib.getVocab(UUID)` 方法也能正常工作了。`MainActivity` 托管的 `VocabularyFragment`

终于可以显示真正的 WordEntity 对象了。

现在，可以验证我们的成果了。运行 NEC Vocab 应用，进入某词条的词汇详解，点击勾选“斩”按钮，然后按回退键，确认 VocabularyItemFragment 中对应列表项的“斩”得到更新。

7.6 初始化 SQLITE 数据库

还有个不易被发现的问题。即当我们将 NEC Vocab 应用从内存中清除（不是卸载）后，再次启动，我们会发现，原先的词汇表扩大了一倍。如此重复 N 次，词汇表就会扩大 N 倍，直到系统崩溃。之所以这个问题不易被发现，是因为我们使用了单列初始化数据库，应用在内存中存在多久，单列就存在多久，因此，如果 NEC Vocab 应用没有被清空，一直驻留在内存中，则上述问题不会发生。

一个解决方案是只在用户初次进入 NEC Vocab 应用时初始化数据库。这个如何实现呢？

基本思路就是，在第一次进入词汇表的时候，在本地保存一个布尔数据进行记录。如果是第一次进入，保存为 true，然后进行判断。如果是 true，则初始化数据库，并将其修改为 false。

我们在第三章介绍的 SharedPreferences 就是这样一个很好记录工具。如代码清单 7-20 所示，我们再次修改 Wordlib.java 文件，

代码清单 7-20 判读是否是初次进入“词汇”Tab (Wordlib.java)

```
public class WordLib {
    private final String SHARE_APP_TAG= "firstOpen";
    private Boolean first;
    private SharedPreferences setting;
    .....
    private WordLib(Context context) {
        mContext = context.getApplicationContext();
        setting = context.getSharedPreferences(SHARE_APP_TAG, 0);
        first = setting.getBoolean("FIRST",true);
        Log.i(TAG, "第 1 次进入= " + first );
        if(first){
            setting.edit().putBoolean("FIRST", false).commit();
        }
        mDatabase = new VocabBaseHelper(mContext).getWritableDatabase();
        if(first)
            addVocabList();
    }
    .....
}
```

先将已有的 NEC Vocab 应用卸载，再次安装、运行 NEC Vocab 应用。看看词汇表中的词汇数量，然后退出，并将 NEC Vocab 应用从内存中清除。再次运行，看看词汇表中词汇的数量是否发生变化。

需要指出的是，本章我们几乎没有涉及视图层和控制层（**只是在 VocabularyFragment 中覆盖 onPause()方法时添加了一行代码**），主要工作都在模型层进行。这就是 MVC 设计模式给我们带来的便捷之处。