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

8 打包词汇资源

8.1 本章目标

在初次安装应用时 SQLite 数据库是没有任何数据的。第 7 章我们通过在 WordLib 的静态块里存放少量词汇数据为数据库 SQLite 添加数据，之后再从数据库中读取数据。这只是让 NEC Vocab 应用尽快运行起来的权宜之计。

实际中，NEC Vocab 应用是要向用户提供一些学习材料的。当然可以让用户从后台服务端下载数据，但服务端的开发超出了本书的范围。这时 assets 就闪亮登场了。利用 assets，我们可以通过资源打包方式，在用户下载 app 时附带上一些基础学习材料，并在用户首次运行时将这些资源存放到本地的 SQLite 数据库中。

本章，我们用 assets 存放图片和 JSON 格式的词汇文件，并介绍如何读取和解析它们，同时将它们存放到 SQLite 数据库中。

8.2 使用 assets 存放资源文件

到目前为止，NEC Vocab 应用所使用的资源，包括"词汇详解"需要显示的图片资源，都存放在 res 文件夹下。对于这种存放在 res 文件夹下资源，系统会在 R 文件里面自动生成该资源的 ID，只需要调用 R.XXX.id 即可访问它们。比如在 res/drawable 目录保存的 dolphin.png 图片，我们就是使用 R.drawable.dolphin 这样的 ID 获取到它的。

为何使用 assets

NEC Vocab 应用将有数千条甚至更多的词汇，要用到成百上千张图片。可以想象，如果使用 Android 资源系统一个个去处理，需要将成百上千张图片复制到 res/drawable 文件夹里，将数千条词汇写入 WordLib.java 的静态块里，这是多么艰巨的任务。要是能把这些文件放在一个文件夹里随应用打包就好了，可惜 Android 资源系统并不支持。

这正是 assets 大显伸手的地方。assets 是随应用打包的文件系统，支持任意层次的文件目录结构。因此，类似游戏这样需要加载大量图片和声音资源的应用通常都会使用它。

放在 assets 文件夹下面的资源不会被 R 文件编译，所以不能像调用 R.XXX.id 那样直接使用。Android 提供了一个工具类 **AssetManager**，以便我们操作 assets 文件夹下的资源。

8.2.1 导入 assets

现在我们来导入 assets。首先创建 **assets** 目录。在 **Project** 视图下，右键单击 **main**，选择 **New → Folder → Assets Folder**，在弹出的如图 8-1 所示对话框中，单击 **Finish** 按钮完成。

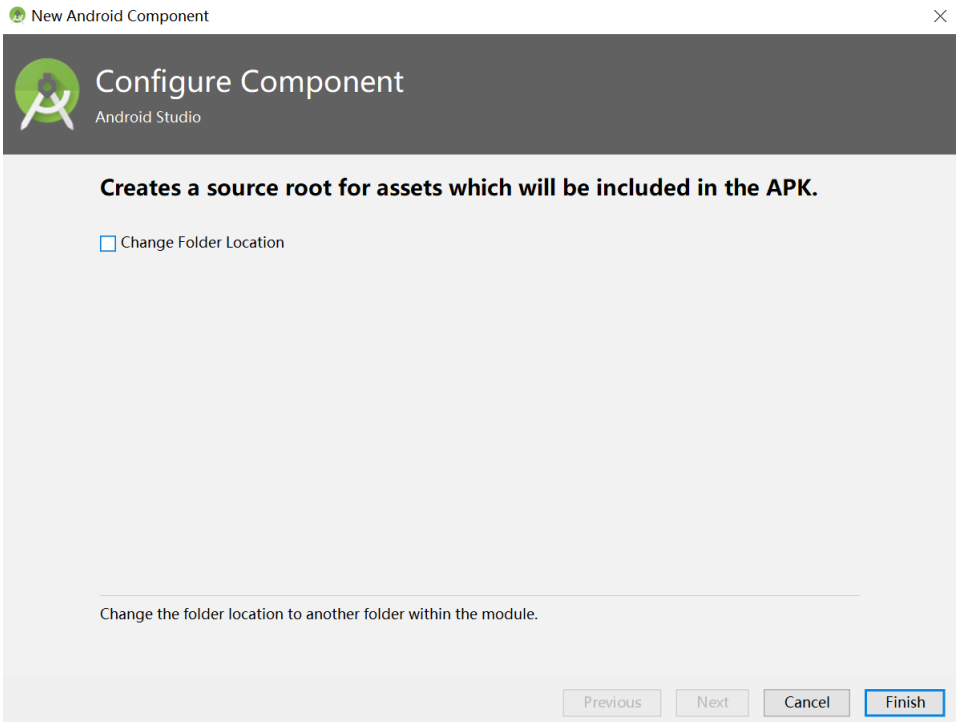

图 8-1 创建 assets 目录

8.2.2 使用 assets

assets 文件夹里面的文件都是保持原始的文件格式，需要用 AssetManager 以**字节流**（主要针对图类资源）或**字符流**（主要针对文本文件）的形式读取文件。其步骤是：

1. 在 Java 里面调用 **getAssets()**获取 **AssetManager** 引用。
2. 用 **AssetManager** 的 **open(String fileName, int accessMode)** 方法则指定读取的文件以及访问模式就能得到输入流 **InputStream**。

3. 用被打开的文件的 **inputStream** 读取文件，
4. 读取完成后关闭输入流 **inputStream.close()**。
5. 最后调用 **AssetManager.close()**关闭 **AssetManager**。

assets 目录下主要存放四种文件：文本文件、图像文件、网页文件（包括 html 中引用的 js/ccs/jpg 等资源）、音频视频文件。本书主要介绍加载 assets 目录下文本文件（这里主要是 JSON 文件）和图片资源。需要注意的是，assets 中的文件只可读取，不能进行写的操作。

8.2.3 用 IO 流读取文件

文件在 Android 程序中通常是以**流**的形式来操作的。**数据在两个存储位置之间的传输称为流**。本书主要介绍字节流与字符流。

字节流与字符流：

1. 读写单位：字节流以字节（1 byte，1byte=8bit）为单位；字符流以字符为单位，根据码表映射字符，一次可能读多个字节。
2. 处理对象：字节流能处理所有类型的数据（如图片、avi 等）；字符流只能处理字符类型的数据。
3. 缓存：字节流在操作的时候本身是不会用到缓冲区的，是文件本身的直接操作的；字符流在操作的时候是会用到缓冲区的，是通过缓冲区来操作文件。

InputStreamReader 类

InputStreamReader 类是从字节流到字符流的桥接器，继承自 Reader 类。

两点说明：

1. 为什么要从字节流到字符流？

- 计算机存储的单位是字节，如尽管 txt 文本中有中文汉字这样的字符，但是对计算机而言，是以字节形式存在的；
- 字节流是单个字节为单位读取的，但是不同字符集解码成字符所使用的字节个数不一定相同，因此以字节流读取有时会出现乱码；
- 用一个流把字节流读取的字节进行缓冲而后再通过字符集解码成字符返回，从形式上看是字符流；
- InputStreamReader 流就是起这个作用，实现从字节流到字符流的转换。

2. 为什么要使用指定的字符集读取字节并将它们解码为字符怎么理解？

字节本质是 8 个二进制位，且不同的字符集对同一字节解码后的字符结果是不同的，因此在读取字符时务必要指定合适的字符集，否则读取的内容会产生乱码。如果不指定字符集编码，该解码过程将使用平台默认的字符编码，如：GBK。

缓冲流：BufferedReader

缓冲流 BufferedReader 类继承自 Reader 类。从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取。BufferedReader 中提供了 **readLine()**方法，可以读取一行文本。

既然缓冲流是对流的功能和读写效率的加强和提高，所以在创建缓冲流的对象时要传入要加强的流（Reader）的对象。通常我们通过向 BufferedReader 传递一个 InputStreamReader 流的对象，来创建一个 BufferedReader 对象。例如通过 Bufferedreader 捕获所输入的语句：

```
BufferedReader bufferedReader =new BufferedReader(new InputStreamReader(System.in));
String text =bufferedReader.readLine();
```

8.2.4 读取 assets 目录下的 json 文件

右键单击 **assets** 目录，选择 **New** → **Directory** 菜单项，为 JSON 文件资源创建 vocab_json 子目录。这样我们就可以通过 AssetManager 来读取该文件夹的 JSON 文件。

右键单击 com.studio.aime.necvocab，选择 **New** → **JavaClass**，在弹出的对话框中的 **Name** 栏填写 data.LoadFile，如图 8-2 所示。

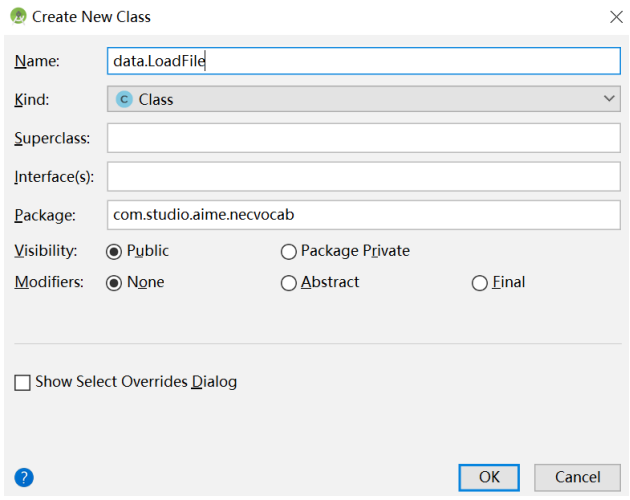

图 8-2 创建 LoadFile 类

这样，便在 com.studio.aime.necvocab 中创建了子包 data，并在 data 包下创建了 LoadFile.java 文件。

如代码清单 8-1 所示，我们在 LoadFile 类中添加一个静态常量 VOCAB_JSON，用于标记 assets 文件夹下的存放 JSON 文件的目录。在静态方法 getJson(...)中，获得 assets 资源管理器，使用 IO 流读取 json 文件内容，并存放在 StringBuilder 对象中，最后返回一个 JSON 字符串。

代码清单 **8-1** 读取 **assets** 目录下的 **json** 文件（**LoadFile.java**）

```
public class LoadFile {
    private static final String VOCAB_JSON = "vocab_json/";

    private static String getJson(Context context, String fileName){
        StringBuilder stringBuilder = new StringBuilder();
        //获得 assets 资源管理器
        AssetManager assetManager = context.getAssets();
        //使用 IO 流读取 json 文件内容，指定字符集编码 utf-8
        try {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
                    assetManager.open(fileName),"utf-8"));
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                stringBuilder.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return stringBuilder.toString();
    }
}
```

8.3 使用 JSON 数据

JSON(JavaScript Object Notation) 是近年流行起来的一种轻量级的数据交换格式。JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯（包括 C、C++、Java、JavaScript、 Python 等）。这使 JSON 成为理想的数据交换语言，尤其适用于 Web 服务。

8.3.1 使用 JSON 格式存放数据

JSON 数据有两种结构：**JSON 对象**是一系列包含在**{ }**中的"**名值对**"；**JSON 数组**是包含在**[ ]**中用逗号隔开的 **JSON 对象列表**。

JSON 对象

JSON 对象是一个无序的“‘键-值’对”集合。一个 JSON 对象以“**{**”（左括号）开始，“**}**”（右括号）结束。每个“键”后跟一个“**:**”（冒号）；“‘键-值’对”之间使用“**,**”（逗号）分隔。

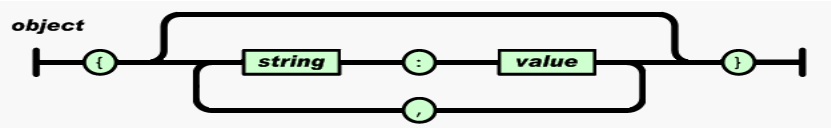

图 8-3 JSON 对象

“键”（key）必须为 **String** 类型， 值（value）可以是 String、Number、Object、Array 等数据类型。

我们可以用一个 JSON 对象描述一个词条：

```
{
   "mEnglish": "beautiful",
   "mPartOfSpeech": "adj.",
   "mChinese": "美丽的",
   "mExample": "a beautiful woman",
   "mImgID": "beautiful"
}
```

JSON 数组

JSON 数组是值（value）的有序集合。一个数组以“**[**”（左中括号）开始，“**]**”（右中括号）结束。值之间使用“**,**”（逗号）分隔。

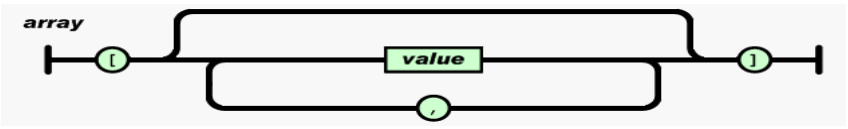

图 8-4 JSON 数组

值（value）可以是双引号括起来的字符串（string）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。

我们可以用一个 JSON 数组描述多条词汇：

```
[
   {
      "mEnglish": "archive",
      "mPartOfSpeech": "v.",
      "mChinese": "存档",
      "mExample": "Scripts can monitor, archive, update, upload, download, and transform data",
```

```
    "mImgID": "archive"
  },
  {
    "mEnglish": "beautiful",
    "mPartOfSpeech": "adj.",
    "mChinese": "美丽的",
    "mExample": "a beautiful woman",
    "mImgID": "beautiful"
  }
]
```

创建 JSON 文件

下面我们用 JSON 文件存放词汇表，以便用户首次运行 NEC Vocab 应用时能将这些词汇装载的 SQLite 数据库中。

右击 vocab_json 子目录新建 **File**，起名"vocabulary.json"，然后我们在文件中写一些 json 格式的词汇数据（如果词汇很多，使用多个文件）：

代码清单 **8-2** 创建 **json** 格式的词汇表（**vocabulary.json**）

```
[
  {
    "mEnglish": "archive",
    "mPartOfSpeech": "v.",
    "mChinese": "存档",
    "mExample": "Scripts can monitor, archive, update, upload, download, and transform data",
    "mImgID": "archive"
  },
  {
    "mEnglish": "beautiful",
    "mPartOfSpeech": "adj.",
    "mChinese": "美丽的",
    "mExample": "a beautiful woman",
    "mImgID": "beautiful"
  },
  ......
  {
    "mEnglish": "sail",
    "mPartOfSpeech": "vi.",
    "mChinese": "起航",
    "mExample": "When does your steamer sail?",
    "mImgID": "sail"
  }
]
```

注意，为了后面使用便于使用 Gson 解析这些词汇，词汇表中每个 Json 对象的"名称"都使用和 WordEntity 类中**成员变量**完全相同的字符串。

那么，怎么读到这些数据呢?

8.3.2 解析 JSON 数据

Android sdk 中为开发者提供有对应 JSON 数据的 Java 对象，如 JSONObject 和 JSONArray。使用 JSONObject 和 JSONArray 两个类可以完成对 JSON 数据的解析。

JSONObject

例如，要解析的 JSON 数据如下：

```
{
   "mEnglish": "beautiful",
   "mPartOfSpeech": "adj.",
   "mChinese": "美丽的",
   "mExample": "a beautiful woman",
   "mImgID": "beautiful"
}
```

使用 JSONObject 解析 JSON 对象：

```
JSONObject jsonObj = new JSONObject(json1);
String english = jsonObj.optString("mEnglish");
String partOfSpeech = jsonObj.optString("mPartOfSpeech");
String chinese = jsonObj.optString("mChinese");
String example = jsonObj.optString("mExample");
String imgID = jsonObj.optString("mImgID");
```

optXXX()方法在解析数据时比 getXXX()方法更安全，如果对应字段不存在，optXXX()方法会返回空值或者 0，而 getXXX()方法会抛出异常。

JSONArray

需要声明一个数组或列表去接收解析出来的 json 数组，否则是无法装载的。拿到 json 数组以后，一般情况是继续为 json 对象进行取值，所以，还需要将 json 数组循环取出放到 json 对象中。

使用 JSONArray 解析 JSON 数组：

```
JSONArray jsonArray = new JSONArray(json2);
for (int i = 0; i < jsonArray.length(); i++) {
      String english = jsonArray. getJSONObject(i).optString("mEnglish");
      String partOfSpeech = jsonArray. getJSONObject(i).optString("mPartOfSpeech");
      String chinese = jsonArray. getJSONObject(i).optString("mChinese");
      String example = jsonArray. getJSONObject(i).optString("mExample");
      String imgID = jsonArray.getJSONObject(i).optString("mImgID");
      ……//存放到某列表中
}
```

8.4 使用 Gson 解析 JSON 文件

json.org API 提供有对应 JSON 数据的 Java 对象，如 JSONObject 和 JSONArray。使用 JSONObject(String)构造函数，可以很方便地把 JSON 数据解析成相应的 Java 对象。但无论什么平台，把 JSON 数据转化为 Java 对象都是应用开发的常见任务。于是，很多开发者就创建了一些工具库，希望能简化 JSON 数据和 Java 对象的互转。Gson 就是这样的一个工具库（https://github.com/google/gson）。不用写任何解析代码，Gson 就能自动把 JSON 数据映射为 Java 对象。因为这个特性，Gson 现在是开发者最喜爱的 JSON 解析库。

使用 GSON，需添加 Gson 依赖项。点击 File → Project Structure，在出现的对话框中选 app → Dependenies，点击右上角的绿色+，在弹出的对话框中选择 GSON(com.google.code.gson:gson:2.X.X)，点击 OK，如图 8-5 所示：

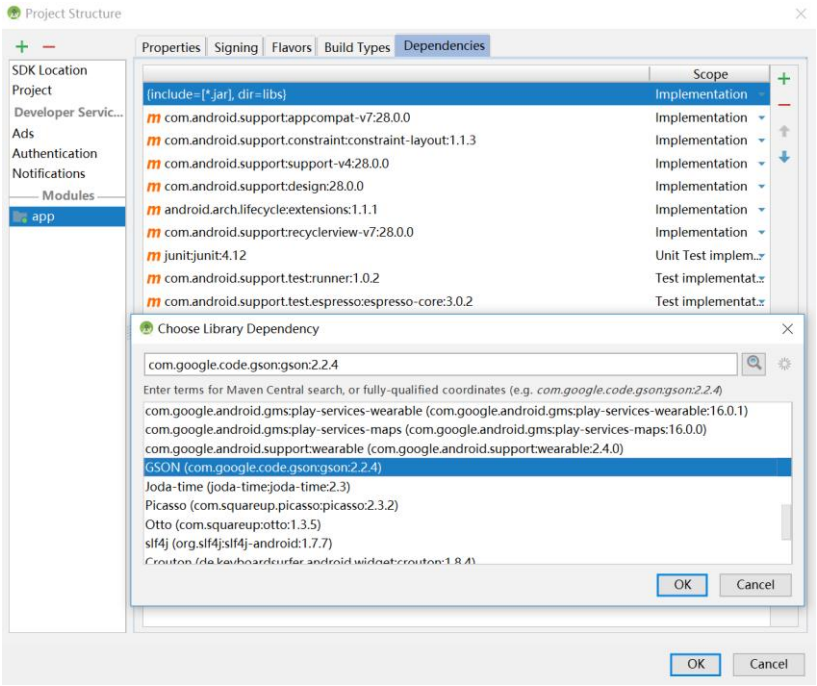

图 8-5 添加 Gson 依赖项

这样准备工作就做好了。

8.4.1 解析词汇 JSON 数据

使用 Gson 库之前，需要创建 JSON 数据所对应的实体类。这里的实体类就是我们已经创

建的 WordEntity 类。需要注意的是实体类中的成员变量名要与 JSON 数据的 key 值一致。当然，所需的一切已经完成，这里我们**需要将 WordEntity 的 mImgID 变量由 int 类型修改为 String 类型**，如代码清单 8-3 所示。

代码清单 8-3 修改 WordEntity 类（WordEntity.java）

```
public class WordEntity implements Serializable {
    private UUID mId;
    boolean mKilled;
    private String mEnglish;
    private String mPartOfSpeech;
    private String mChinese;
    private String mExample;
    private String mImgID;

    public WordEntity(){
        this(UUID.randomUUID());      }

    public WordEntity(UUID uuid){
        mId = uuid;
    }
    public WordEntity(String ve,String vp, String vc, String eg, String imgId){
        mId = UUID.randomUUID();
        mEnglish = ve;
        mPartOfSpeech = vp;
        mChinese = vc;
        mExample = eg;
        mImgID = imgId;
    }
    public UUID getId() {
        return mId;
    }
    public boolean isKilled() {
        return mKilled;
    }
    public void setKilled(boolean killed) {
        mKilled = killed;
    }
    public String getEnglish() {
        return mEnglish;
    }
    public void setEnglish(String english) {
        mEnglish = english;
    }
    public String getPartOfSpeech() {
        return mPartOfSpeech;
    }
    public void setPartOfSpeech(String partOfSpeech) {
        mPartOfSpeech = partOfSpeech;
    }
    public String getChinese() {
        return mChinese;
    }
```

```
    public void setChinese(String chinese) {
        mChinese = chinese;
    }
    public String getExample() {
        return mExample;
    }
    public void setExample(String example) {
        mExample = example;
    }
    public String getImgID() {
        return mImgID;
    }
    public void setImgID(String imgID) {
        mImgID = imgID;
    }
}
```

使用 Gson 解析 JSON 对象示例代码如下：

```
Gson gson = new Gson();
WordEntity word = gson.fromJson(json1, WordEntity.class);
```

使用 Gson 解析 JSON 数组示例代码如下：

```
Gson gson =new Gson();
Type type = new TypeToken<List<WordEntity>>(){}.getType();
return gson.fromJson(mJson,type);
```

如代码清单 8-4 所示，在 LoadFile.java 文件中添加 JSON 解析代码。

代码清单 8-4 利用 GSON 解析 JSON 数据（LoadFile.java）

```
public class LoadFile {
    private static final String VOCAB_JSON = "vocab_json/";
    public static List<WordEntity> getJsonList (Context context) {
        String mJson;
        mJson = getJsonFile(context,VOCAB_JSON + "vocabulary.json");
        Gson gson =new Gson();
        Type type = new TypeToken<List<WordEntity>>(){}.getType();
        return gson.fromJson(mJson,type);
    }
    private static String getJsonFile(Context context, String fileName){
        StringBuilder stringBuilder = new StringBuilder();
        //获得 assets 资源管理器
        AssetManager assetManager = context.getAssets();
        //使用 IO 流读取 json 文件内容
        try {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
                    assetManager.open(fileName),"utf-8"));
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                stringBuilder.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
```

```
            }
            return stringBuilder.toString();
        }
    }
```

接下来，我们在 WordLib.java 文件中来获取所需的 JSON 数据，并将从 JSON 文件中获取的词汇数据插入数据库，如代码清单 8-5 所示。现在我们再也不需要再在 WordLib 中通过静态块向数据库添加数据了，删除静态块中的词汇数据。

代码清单 8-5 将从 JSON 文件中获取的词汇数据插入数据库（WordLib.java）

```
public class WordLib {
    private final String SHARE_APP_TAG= "firstOpen";
    private Boolean first;
    private SharedPreferences setting;
    private static final String TAG = "WordLib";
    private Context mContext;
    private static WordLib sWordLib;
    private List<WordEntity> mWordList;
    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();
        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();
        mWordList = new ArrayList<>();
        mWordList = LoadFile.getJsonList(mContext);
        if(first)
            addVocabList();
    }
    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;
    }
    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();
        }
    }
    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().trim());
        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 : mWordList ~~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 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 new VocabCursorWrapper(cursor);
    }
```

```
    private static List<WordEntity> sWords = new ArrayList<>();
    static{
        for (int i = 1; i <= 20; i++) {
            sWords.add(new WordEntity("beautiful","adj.","美丽的",
                    "a beautiful woman","beautiful"));
            sWords.add(new WordEntity("chat","vi.","谈话",
                    "What were you chatting about?","chat"));
            sWords.add(new WordEntity("classic","adj.","经典的",
                    "a classic novel","classicbook"));
            sWords.add(new WordEntity("dance","v.","跳舞",
                    "They stayed up all night singing and dancing.",""));
            sWords.add(new WordEntity("fly","v.","飞行",
                    "The dog is learning to fly.","fly"));
            sWords.add(new WordEntity("loudly","adv.","大声地",
                    "They were talking loudly.","loudly"));
        //}
    }
}
```

当然，此时在 VocabularyFragment.java 文件中会有一个错误。这是因为我们改变了 ImgID 的属性。**setImageResource(…)**方法必须接受 int 类型的数据，我们需要接收 **Bitmap**（位图）类型数据，因此改用 **setImageBitmap(…)**方法接收。但此时我们还没有获取图片文件，因此暂时先将 setImageBitmap(…)方法的参数设为 null，如代码清单 8-6 所示。

代码清单 8-6 在 VocabularyFragment 中更新数据（VocabularyFragment.java）

```
public class VocabularyFragment extends Fragment{
    private static final String ARG_PARAM = "param";
    ......
    private void updateWords(){
        mKilledCheckBox.setChecked(mWord.isKilled());
        mEnglishTextView.setText(mWord.getEnglish());
        mPartOfSpeech.setText(mWord.getPartOfSpeech());
        mChineseTextView.setText(mWord.getChinese());
        mExampleGratiaView.setText(mWord.getExample());
        mImageView.setImageResource(mWord.getImgID());
        mImageView.setImageBitmap(null);
    }
}
```

现在运行 NEC Vocab 应用，应该能看到图 8-6 所示的没有图片的效果。

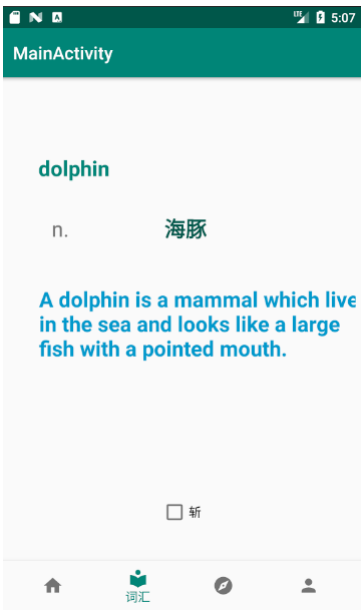

图 8-6 没有图片的效果

8.5 Bitmap 和 BitmapFactory

在添加图片资源之前，我们先来了解一下 Bitmap 和 BitmapFactory。

8.5.1 Bitmap

在 NEC Vocab 应用中我们用到的图片资源为*.png* 格式，我们将以位图 Bitmap 的格式获取它们。

Bitmap（位图）

Bitmap 是图像处理最重要的中转类之一。用它可以获取图像文件信息，借助 Matrix 进行图像剪切、旋转、缩放等操作，再以指定格式保存图像文件。

构造 Bitmap 对象

我们创建一个类的对象通常是通过其构造方法实例化。Bitmap 采用工厂设计模式，不能直接调用其构造方法，而是通过 BitmapFactory 工厂类的 static Bitmap decodeXxx()方法实例化。

8.5.2 BitmapFactory 类

使用 Bitmap(位图)时，我们总是会与 BitmapFactory 类打交道。BitmapFactory 用于从各种

资源、文件、流和字节数组中创建 Bitmap 位图对象。即使用 BitmapFactory 类就创建 Bitmap 位图。下面，看看 BitmapFactory 类提供的用于创建 Bitmap 位图的方法。BitmapFactory 类是一个工具类，提供大量方法，这些方法可用于从不同的数据源解析、创建 Bitmap 位图。

BitmapFactory 类创建 Bitmap 位图的方法说明。

BitmapFactory 类创建 Bitmap 位图的方法很多，我们主要使用**输入流**方法：

```
public static Bitmap decodeStream (InputStream is)
```

这里，参数 is 持有原始数据用于解码位图的输入流，返回解码后的位图。

下面，我们就使用 BitmapFactory 来创建 Bitmap(位图)。

8.5.3 使用 BitmapFactory 创建 Bitmap

从 assets 中获取资源，使用 BitmapFactory 创建 Bitmap 的示例代码如下：

```
try {
        InputStream is=this.getAssets().open("iv_dzdp_iv.png");
         bitmap=BitmapUtils.decodeStream(is);
} catch (IOException e) {
        e.printStackTrace();
}
```

8.6 加载图片资源

为方便组织文件，我们创建一个 vocab_png 子目录。右键单击 assets 目录，选择 **New → Directory** 菜单项，为图片资源创建 vocab_png 子目录，如图 8-7 所示。

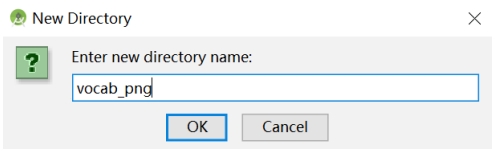

图 8-7 创建 vocab_png 子目录

下载并复制一些 png 格式的图片文件至 assets/vocab_png 目录，如图 8-8 所示（最好为这些图片取个易于辨识的名义，以便在 JSON 文件中使用）。

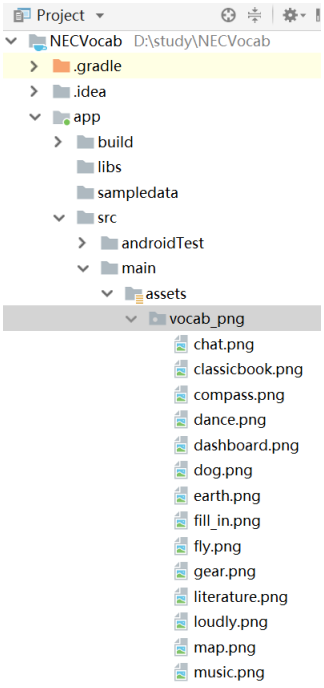

图 8-8 已导入 assets 的图片

重新编译应用，确保一切正常。接下来编写代码，列出所有这些资源并展示给用户。在 LoadFile.java 添加静态常量 VOCAB_PNG 以指示图片所在的文件夹，同时添加静态方法 loadImages(...)，如代码清单 8-7 所示。

代码清单 8-7 加载图片资源（LoadFile.java）

```
public class LoadFile {
    ......
    private static final String VOCAB_PNG = "vocab_png";
    public static Bitmap loadImages(Context context,String image) {
        String imageNames;
        InputStream open = null;
        Bitmap bitmap = null;
        try {
            imageNames = VOCAB_PNG+"/"+ image+ ".png";//mWord.getImgID();// ;
            open = context.getAssets().open(imageNames);
            bitmap = BitmapFactory.decodeStream(open);
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        finally {
            if (open != null) {
                try {
                    open.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
```

```
            }
            return bitmap;
        }
    }
```

最后，因为我们只是在"词汇详解"中才显示图片，因此我们在 VocabularyFragment.java 文件中获取图片，如代码清单 8-8 所示。

代码清单 8-8 在 VocabularyFragment 中跟新数据（VocabularyFragment.java）

```
public class VocabularyFragment extends Fragment{
    private static final String ARG_PARAM = "param";
    ......
    private void updateWords(){
        mKilledCheckBox.setChecked(mWord.isKilled());
        mEnglishTextView.setText(mWord.getEnglish());
        mPartOfSpeech.setText(mWord.getPartOfSpeech());
        mChineseTextView.setText(mWord.getChinese());
        mExampleGratiaView.setText(mWord.getExample());
        mImageView.setImageBitmap(LoadFile.loadImages(getActivity(), mWord.getImgID()));
    }
}
```

现在运行 NECVocab 应用。应该可以看到如图 8-9 所示的结果。

图 8-9 运行结果